

RichFaces Developer Guide

RichFaces framework with a huge library of
rich components and skinnability support

Copyright © 2007 Red Hat

Table of Contents

| | |
|---|----|
| 1. Introduction | 1 |
| 2. Technical Requirements | 3 |
| 2.1. Supported Java Versions | 3 |
| 2.2. Supported JavaServer Faces Implementations | 3 |
| 2.3. Supported Servers | 3 |
| 2.4. Supported Browsers | 4 |
| 3. Getting Started with RichFaces | 5 |
| 3.1. Downloading RichFaces 3.1.0 | 5 |
| 3.2. Installation | 5 |
| 3.3. Simple Ajax Echo Project | 6 |
| 3.3.1. JSP Page | 6 |
| 3.3.2. Data Bean | 7 |
| 3.3.3. faces-config.xml | 7 |
| 3.3.4. Web.xml | 7 |
| 3.3.5. Deployment | 8 |
| 4. Settings for different environments | 9 |
| 4.1. Web Application Descriptor Parameters | 9 |
| 4.2. Sun JSF RI | 11 |
| 4.3. Apache MyFaces | 12 |
| 4.4. Facelets Support | 12 |
| 4.5. JBoss Seam Support | 12 |
| 4.6. Portlet Support | 15 |
| 4.7. Sybase EAServer | 15 |
| 5. Basic concepts of the RichFaces Framework | 17 |
| 5.1. Introduction | 17 |
| 5.2. RichFaces Architecture Overview | 18 |
| 5.3. Limitations and Rules | 21 |
| 5.4. Ajax Request Optimization | 22 |
| 5.4.1. Re-Rendering | 22 |
| 5.4.2. Queue and Traffic Flood Protection | 24 |
| 5.4.3. Data Processing Options | 24 |
| 5.4.4. Action and Navigation | 25 |
| 5.4.5. JavaScript Interactions | 26 |
| 5.4.6. Iteration components Ajax attributes | 27 |
| 5.4.7. Other useful attributes | 27 |
| 5.5. How To... | 28 |
| 5.5.1. Send an Ajax request | 28 |
| 5.5.2. Decide What to Send | 28 |
| 5.5.3. Decide What to Change | 29 |
| 5.6. Filter Configuration | 29 |
| 5.7. Scripts and Styles Load Strategy | 31 |
| 5.8. Request Errors and Session Expiration Handling | 32 |
| 5.8.1. Request Errors Handling | 32 |

| | |
|---|----|
| 5.8.2. Session Expired Handling | 33 |
| 5.9. Skinnability | 33 |
| 5.9.1. Why Skinnability | 33 |
| 5.9.2. Using Skinnability | 34 |
| 5.9.3. Example | 34 |
| 5.9.4. Skin Parameters Tables in RichFaces | 35 |
| 5.9.5. Creating and Using Your Own Skin File | 37 |
| 5.9.6. Built-in skinnability in RichFaces | 37 |
| 6. The RichFaces Components | 39 |
| 6.1. < a4j:ajaxListener > | 39 |
| 6.1.1. Creating on a page | 39 |
| 6.1.2. Dynamical creation of a component from Java code | 39 |
| 6.1.3. Key attributes and ways of usage | 40 |
| 6.1.4. | 40 |
| 6.1.5. Relevant resources links | 40 |
| 6.2. < a4j:keepAlive > | 40 |
| 6.2.1. Creating on a page | 41 |
| 6.2.2. Dynamical creation of a component from Java code | 41 |
| 6.2.3. Key attributes and ways of usage | 41 |
| 6.2.4. Relevant resources links | 42 |
| 6.3. < a4j:actionparam > | 42 |
| 6.3.1. Creating on a page | 43 |
| 6.3.2. Dynamical creation of a component from Java code | 43 |
| 6.3.3. Key attributes and ways of usage | 43 |
| 6.3.4. Relevant resources links | 44 |
| 6.4. < a4j:commandButton > | 44 |
| 6.4.1. Description | 44 |
| 6.4.2. Creating on a page | 47 |
| 6.4.3. Dynamical creation of a component from Java code | 48 |
| 6.4.4. Key attributes and ways of usage | 48 |
| 6.4.5. Relevant resources links | 49 |
| 6.5. < a4j:commandLink > | 49 |
| 6.5.1. Description | 49 |
| 6.5.2. Creating on a page | 53 |
| 6.5.3. Dynamical creation of a component from Java code | 53 |
| 6.5.4. Key attributes and ways of usage | 53 |
| 6.5.5. Relevant resources links | 54 |
| 6.6. < a4j:form > | 54 |
| 6.6.1. Description | 54 |
| 6.6.2. Creating on a page | 57 |
| 6.6.3. Dynamical creation of a component from Java code | 57 |
| 6.6.4. Key attributes and ways of usage | 57 |
| 6.6.5. Relevant resources links | 58 |
| 6.7. < a4j:htmlCommandLink > | 58 |
| 6.7.1. Description | 58 |

| | |
|--|----|
| 6.7.2. Creating on a page | 61 |
| 6.7.3. Dynamical creation of a component from Java code | 61 |
| 6.7.4. Key attributes and ways of usage | 61 |
| 6.7.5. Relevant resources links | 61 |
| 6.8. < a4j:jsFunction > | 62 |
| 6.8.1. Description | 62 |
| 6.8.2. Creating on a page | 64 |
| 6.8.3. Dynamical creation of a component from Java code | 64 |
| 6.8.4. Key attributes and ways of usage | 64 |
| 6.8.5. Relevant resources links | 65 |
| 6.9. < a4j:include > | 65 |
| 6.9.1. Description | 65 |
| 6.9.2. Creating on a page | 66 |
| 6.9.3. Dynamical creation of a component from Java code | 67 |
| 6.9.4. Relevant resources links | 67 |
| 6.10. < a4j:loadBundle > | 68 |
| 6.10.1. Creating on a page | 68 |
| 6.10.2. Dynamical creation of a component from Java code | 68 |
| 6.10.3. Key attributes and ways of usage | 69 |
| 6.10.4. Relevant resources links | 69 |
| 6.11. < a4j:loadScript > | 69 |
| 6.11.1. Description | 69 |
| 6.11.2. Creating on a page | 70 |
| 6.11.3. Dynamical creation of a component from Java code | 70 |
| 6.11.4. Key attributes and ways of usage | 70 |
| 6.11.5. Relevant resources links | 70 |
| 6.12. < a4j:loadStyle > | 70 |
| 6.12.1. Description | 70 |
| 6.12.2. Creating on a page | 71 |
| 6.12.3. Dynamical creation of a component from Java code | 71 |
| 6.12.4. Key attributes and ways of usage | 71 |
| 6.12.5. Relevant resources links | 72 |
| 6.13. < a4j:log > | 72 |
| 6.13.1. Description | 72 |
| 6.13.2. Creating on a page | 73 |
| 6.13.3. Dynamical creation of a component from Java code | 73 |
| 6.13.4. Key attributes and ways of usage | 74 |
| 6.13.5. Relevant resources links | 74 |
| 6.14. < a4j:mediaOutput > | 74 |
| 6.14.1. Description | 74 |
| 6.14.2. Creating on a page | 78 |
| 6.14.3. Dynamical creation of a component from Java code | 79 |
| 6.14.4. Key attributes and ways of usage | 79 |
| 6.14.5. Relevant resources links | 80 |
| 6.15. < a4j:outputPanel > | 80 |

| | |
|--|-----|
| 6.15.1. Description | 80 |
| 6.15.2. Creating on a page | 81 |
| 6.15.3. Dynamical creation of a component from Java code | 82 |
| 6.15.4. Key attributes and ways of usage | 82 |
| 6.15.5. Relevant resources links | 83 |
| 6.16. < a4j:page > | 84 |
| 6.16.1. Description | 84 |
| 6.16.2. Creating on a page | 85 |
| 6.16.3. Dynamical creation of a component from Java code | 85 |
| 6.16.4. Key attributes and ways of usage | 86 |
| 6.16.5. Relevant resources links | 86 |
| 6.17. < a4j:poll > | 86 |
| 6.17.1. Description | 86 |
| 6.17.2. Creating on a page | 88 |
| 6.17.3. Dynamical creation of a component from Java code | 88 |
| 6.17.4. Key attributes and ways of usage | 88 |
| 6.17.5. Relevant resources links | 89 |
| 6.18. < a4j:portlet > | 90 |
| 6.18.1. Description | 90 |
| 6.18.2. Creating on a page | 90 |
| 6.18.3. Dynamical creation of a component from Java code | 90 |
| 6.18.4. Key attributes and ways of usage | 90 |
| 6.18.5. Relevant resources links | 91 |
| 6.19. < a4j:push > | 91 |
| 6.19.1. Description | 91 |
| 6.19.2. Creating on a page | 93 |
| 6.19.3. Dynamical creation of a component from Java code | 93 |
| 6.19.4. Key attributes and ways of usage | 93 |
| 6.19.5. Relevant resources links | 94 |
| 6.20. < a4j:region > | 95 |
| 6.20.1. Description | 95 |
| 6.20.2. Creating on a page | 96 |
| 6.20.3. Dynamical creation of a component from Java code | 96 |
| 6.20.4. Key attributes and ways of usage | 96 |
| 6.20.5. Relevant resources links | 97 |
| 6.21. < a4j:repeat > | 98 |
| 6.21.1. Description | 98 |
| 6.21.2. Creating on a page | 99 |
| 6.21.3. Dynamical creation of a component from Java code | 99 |
| 6.21.4. Key attributes and ways of usage | 99 |
| 6.21.5. Relevant resources links | 100 |
| 6.22. < a4j:status > | 100 |
| 6.22.1. Description | 100 |
| 6.22.2. Creating on a page | 102 |
| 6.22.3. Dynamical creation of a component from Java code | 102 |

| | |
|--|-----|
| 6.22.4. Key attributes and ways of usage | 102 |
| 6.22.5. Relevant resources links | 103 |
| 6.23. < a4j:support > | 103 |
| 6.23.1. Description | 103 |
| 6.23.2. Creating on a page | 106 |
| 6.23.3. Dynamical creation of a component from Java code | 106 |
| 6.23.4. Key attributes and ways of usage | 106 |
| 6.23.5. Relevant resources links | 108 |
| 6.24. < rich:calendar > | 108 |
| 6.24.1. Description | 108 |
| 6.24.2. Key Features | 109 |
| 6.24.3. Creating the Component with a Page Tag | 115 |
| 6.24.4. Creating the Component Dynamically Using Java | 115 |
| 6.24.5. Details of Usage | 115 |
| 6.24.6. JavaScript API | 121 |
| 6.24.7. Look-and-Feel Customization | 122 |
| 6.24.8. Skin Parameters Redefinition | 122 |
| 6.24.9. Definition of Custom Style Classes | 126 |
| 6.24.10. Relevant Resources Links | 131 |
| 6.25. < rich:componentControl > | 131 |
| 6.25.1. Description | 131 |
| 6.25.2. Creating the Component with a Page Tag | 132 |
| 6.25.3. Creating the Component Dynamically Using Java | 133 |
| 6.25.4. Details of Usage | 133 |
| 6.25.5. Look-and-Feel Customization | 134 |
| 6.25.6. Relevant Resources Links | 134 |
| 6.26. < rich:contextMenu > | 134 |
| 6.26.1. Description | 134 |
| 6.26.2. Key Features | 135 |
| 6.26.3. Creating the Component with a Page Tag | 136 |
| 6.26.4. Creating the Component Dynamically Using Java | 136 |
| 6.26.5. Details of Usage | 137 |
| 6.26.6. JavaScript API | 140 |
| 6.26.7. Look-and-Feel Customization | 140 |
| 6.26.8. Skin Parameters Redefinition | 141 |
| 6.26.9. Definition of Custom Style Classes | 141 |
| 6.26.10. Relevant Resources Links | 142 |
| 6.27. < rich:dataFilterSlider > | 142 |
| 6.27.1. Description | 142 |
| 6.27.2. Key Features | 142 |
| 6.27.3. Creating the Component with a Page Tag | 144 |
| 6.27.4. Creating the Component Dynamically Using Java | 144 |
| 6.27.5. Details of Usage | 145 |
| 6.27.6. Look-and-Feel Customization | 145 |
| 6.27.7. Relevant Resources Links | 145 |

| | |
|---|-----|
| 6.28. < rich:datascroller > | 146 |
| 6.28.1. Description | 146 |
| 6.28.2. Key Features | 146 |
| 6.28.3. Creating the Component with a Page Tag | 150 |
| 6.28.4. Creating the Component Dynamically Using Java | 150 |
| 6.28.5. Details of Usage | 150 |
| 6.28.6. Look-and-Feel Customization | 152 |
| 6.28.7. Skin Parameters Redefinition | 152 |
| 6.28.8. Definition of Custom Style Classes | 153 |
| 6.28.9. Relevant Resources Links | 154 |
| 6.29. < rich:columnGroup > | 155 |
| 6.29.1. Description | 155 |
| 6.29.2. Key Features | 155 |
| 6.29.3. Creating the Component with a Page Tag | 157 |
| 6.29.4. Creating the Component Dynamically Using Java | 157 |
| 6.29.5. Details of Usage | 157 |
| 6.29.6. Look-and-Feel Customization | 159 |
| 6.29.7. Skin Parameters Redefinition | 159 |
| 6.29.8. Definition of Custom Style Classes | 160 |
| 6.29.9. Relevant Resources Links | 160 |
| 6.30. < rich:column > | 160 |
| 6.30.1. Description | 160 |
| 6.30.2. Key Features | 160 |
| 6.30.3. Creating the Component with a Page Tag | 162 |
| 6.30.4. Creating the Component Dynamically Using Java | 162 |
| 6.30.5. Details of Usage | 162 |
| 6.30.6. Look-and-Feel Customization | 165 |
| 6.30.7. Skin Parameters Redefinition | 165 |
| 6.30.8. Definition of Custom Style Classes | 165 |
| 6.30.9. Relevant Resources Links | 166 |
| 6.31. < rich:dataGrid > | 166 |
| 6.31.1. Description | 166 |
| 6.31.2. Key Features | 166 |
| 6.31.3. Creating the Component with a Page Tag | 170 |
| 6.31.4. Creating the Component Dynamically Using Java | 171 |
| 6.31.5. Details of Usage | 171 |
| 6.31.6. Look-and-Feel Customization | 173 |
| 6.31.7. Skin Parameters Redefinition | 173 |
| 6.31.8. Definition of Custom Style Classes | 173 |
| 6.31.9. Relevant Resources Links | 173 |
| 6.32. < rich:dataList > | 173 |
| 6.32.1. Description | 173 |
| 6.32.2. Key Features | 174 |
| 6.32.3. Creating the Component with a Page Tag | 175 |
| 6.32.4. Creating the Component Dynamically Using Java | 175 |

| | |
|---|-----|
| 6.32.5. Details of Usage | 175 |
| 6.32.6. Look-and-Feel Customization | 177 |
| 6.32.7. Definition of Custom Style Classes | 177 |
| 6.32.8. Relevant Resources Links | 178 |
| 6.33. < rich:dataOrderedList > | 178 |
| 6.33.1. Description | 178 |
| 6.33.2. Key Features | 178 |
| 6.33.3. Creating the Component with a Page Tag | 180 |
| 6.33.4. Creating the Component Dynamically Using Java | 180 |
| 6.33.5. Details of Usage | 180 |
| 6.33.6. Look-and-Feel Customization | 181 |
| 6.33.7. Definition of Custom Style Classes | 182 |
| 6.33.8. Relevant Resources Links | 182 |
| 6.34. < rich:dataDefinitionList > | 182 |
| 6.34.1. Description | 182 |
| 6.34.2. Key Features | 183 |
| 6.34.3. Creating the Component with a Page Tag | 185 |
| 6.34.4. Creating the Component Dynamically Using Java | 185 |
| 6.34.5. Details of Usage | 185 |
| 6.34.6. Look-and-Feel Customization | 186 |
| 6.34.7. Definition of Custom Style Classes | 187 |
| 6.34.8. Relevant Resources Links | 187 |
| 6.35. < rich:dataTable > | 187 |
| 6.35.1. Description | 187 |
| 6.35.2. Key Features | 188 |
| 6.35.3. Creating the Component with a Page Tag | 192 |
| 6.35.4. Creating the Component Dynamically from Java | 193 |
| 6.35.5. Details of Usage | 193 |
| 6.35.6. Look-and-Feel Customization | 194 |
| 6.35.7. Skin Parameters Redefinition | 195 |
| 6.35.8. Definition of Custom Style Classes | 195 |
| 6.35.9. Relevant Resources Links | 197 |
| 6.36. < rich:subTable > | 197 |
| 6.36.1. Description | 197 |
| 6.36.2. Key Features | 198 |
| 6.36.3. Creating the Component with a Page Tag | 201 |
| 6.36.4. Creating the Component Dynamically Using Java | 201 |
| 6.36.5. Details of Usage | 201 |
| 6.36.6. Look-and-Feel Customization | 202 |
| 6.36.7. Skin Parameters Redefinition | 202 |
| 6.36.8. Definition of Custom Style Classes | 202 |
| 6.37. < rich:dndParam > | 203 |
| 6.37.1. Description | 203 |
| 6.37.2. Creating the Component with a Page Tag | 204 |
| 6.37.3. Creating the Component Dynamically Using Java | 204 |

| | |
|---|-----|
| 6.37.4. Details of Usage | 204 |
| 6.37.5. Look-and-Feel Customization | 205 |
| 6.37.6. Relevant Resources Links | 205 |
| 6.38. < rich:dragIndicator > | 205 |
| 6.38.1. Description | 205 |
| 6.38.2. Key Features | 206 |
| 6.38.3. Creating the Component with a Page Tag | 206 |
| 6.38.4. Creating the Component Dynamically Using Java | 207 |
| 6.38.5. Details of Usage | 207 |
| 6.38.5.1. Macro definitions | 207 |
| 6.38.5.2. Predefined macro definitions | 208 |
| 6.38.5.3. Marker customization | 208 |
| 6.38.6. Look-and-Feel Customization | 209 |
| 6.38.7. Relevant Resources Links | 209 |
| 6.39. < rich:dragSupport > | 209 |
| 6.39.1. Description | 209 |
| 6.39.2. Key Features | 209 |
| 6.39.3. Creating the Component with a Page Tag | 212 |
| 6.39.4. Creating the Component Dynamically Using Java | 212 |
| 6.39.5. Details of Usage | 212 |
| 6.39.6. Look-and-Feel Customization | 214 |
| 6.39.7. Relevant Resources Links | 214 |
| 6.40. < rich:dropSupport > | 214 |
| 6.40.1. Description | 214 |
| 6.40.2. Key Features | 215 |
| 6.40.3. | 217 |
| 6.40.4. Creating the Component with a Page Tag | 217 |
| 6.40.5. Creating the Component Dynamically Using Java | 217 |
| 6.40.6. Details of Usage | 218 |
| 6.40.7. Look-and-Feel Customization | 220 |
| 6.40.8. Relevant Resources Links | 220 |
| 6.41. < rich:dragListener > | 220 |
| 6.41.1. Description | 220 |
| 6.41.2. Key Features | 220 |
| 6.41.3. Creating the Component with a Page Tag | 221 |
| 6.41.4. Creating the Component Dynamically Using Java | 221 |
| 6.41.5. Details of Usage | 221 |
| 6.41.6. Look-and-Feel Customization | 222 |
| 6.42. < rich:dropListener > | 222 |
| 6.42.1. Description | 222 |
| 6.42.2. Key Features | 222 |
| 6.42.3. Creating the Component with a Page Tag | 222 |
| 6.42.4. Creating the Component Dynamically Using Java | 223 |
| 6.42.5. Details of Usage | 223 |
| 6.42.6. Look-and-Feel Customization | 223 |

| | |
|---|-----|
| 6.43. < rich:dropDownMenu > | 224 |
| 6.43.1. Description | 224 |
| 6.43.2. Key Features | 224 |
| 6.43.3. Creating the Component with a Page Tag | 226 |
| 6.43.4. Creating the Component Dynamically Using Java | 226 |
| 6.43.5. Details of Usage | 226 |
| 6.43.6. Look-and-Feel Customization | 229 |
| 6.43.7. Skin Parameters Redefinition | 230 |
| 6.43.8. Definition of Custom Style Classes | 230 |
| 6.43.9. Relevant Resources Links | 232 |
| 6.44. < rich:menuGroup > | 232 |
| 6.44.1. Description | 232 |
| 6.44.2. Key Features | 232 |
| 6.44.3. Creating the Component with a Page Tag | 234 |
| 6.44.4. Creating the Component Dynamically Using Java | 234 |
| 6.44.5. Details of Usage | 234 |
| 6.44.6. Look-and-Feel Customization | 236 |
| 6.44.7. Skin Parameters Redefinition | 236 |
| 6.44.8. Definition of Custom Style Classes | 236 |
| 6.44.9. Relevant Resources Links | 237 |
| 6.45. < rich:menuItem > | 238 |
| 6.45.1. Description | 238 |
| 6.45.2. Key Features | 238 |
| 6.45.3. Creating the Component with a Page Tag | 241 |
| 6.45.4. Creating the Component Dynamically Using Java | 241 |
| 6.45.5. Details of Usage | 241 |
| 6.45.6. Look-and-Feel Customization | 242 |
| 6.45.7. Skin Parameters Redefinition | 243 |
| 6.45.8. Definition of Custom Style Classes | 243 |
| 6.45.9. Relevant Resources Links | 244 |
| 6.46. < rich:menuSeparator > | 244 |
| 6.46.1. Description | 244 |
| 6.46.2. Creating the Component with a Page Tag | 245 |
| 6.46.3. Creating the Component Dynamically Using Java | 245 |
| 6.46.4. Look-and-Feel Customization | 246 |
| 6.46.5. Skin Parameters Redefinition | 246 |
| 6.46.6. Definition of Custom Style Classes | 246 |
| 6.46.7. Relevant Resources Links | 247 |
| 6.47. < rich:effect > | 247 |
| 6.47.1. Description | 247 |
| 6.47.2. Key Features | 247 |
| 6.47.3. Creating the Component with a Page Tag | 248 |
| 6.47.4. Creating the Component Dynamically Using Java | 248 |
| 6.47.5. Details of Usage | 248 |
| 6.47.6. Look-and-Feel Customization | 250 |

| | |
|---|-----|
| 6.47.7. Relevant Resources Links | 250 |
| 6.48. < rich:gmap > | 250 |
| 6.48.1. Description | 250 |
| 6.48.2. Key Features | 251 |
| 6.48.3. Creating the Component with a Page Tag | 253 |
| 6.48.4. Creating the Component Dynamically Using Java | 253 |
| 6.48.5. Details of Usage | 253 |
| 6.48.6. Look-and-Feel Customization | 256 |
| 6.48.7. Definition of Custom Style Classes | 256 |
| 6.48.8. Relevant Resources Links | 256 |
| 6.49. < rich:virtualEarth > | 256 |
| 6.49.1. Description | 256 |
| 6.49.2. Key Features | 256 |
| 6.49.3. Creating the Component with a Page Tag | 258 |
| 6.49.4. Creating the Component Dynamically Using Java | 258 |
| 6.49.5. Details of Usage | 259 |
| 6.49.6. Look-and-Feel Customization | 260 |
| 6.49.7. Definition of Custom Style Classes | 260 |
| 6.49.8. Relevant Resources Links | 260 |
| 6.50. < rich:inputNumberSlider > | 260 |
| 6.50.1. Description | 260 |
| 6.50.2. Key Features | 261 |
| 6.50.3. Creating the Component with a Page Tag | 264 |
| 6.50.4. Creating the Component Dynamically Using Java | 264 |
| 6.50.5. Details of Usage | 265 |
| 6.50.6. Look-and-Feel Customization | 266 |
| 6.50.7. Skin Parameters Redefinition | 266 |
| 6.50.8. Definition of Custom Style Classes | 267 |
| 6.50.9. Relevant Resources Links | 268 |
| 6.51. < rich:inputNumberSpinner > | 269 |
| 6.51.1. Description | 269 |
| 6.51.2. Key Features | 269 |
| 6.51.3. Creating the Component with a Page Tag | 272 |
| 6.51.4. Creating the Component Dynamically Using Java | 272 |
| 6.51.5. Details of Usage | 272 |
| 6.51.6. Look-and-Feel Customization | 273 |
| 6.51.7. Skin Parameters Redefinition | 274 |
| 6.51.8. Definition of Custom Style Classes | 274 |
| 6.51.9. Relevant Resources Links | 275 |
| 6.52. < rich:insert > | 276 |
| 6.52.1. Description | 276 |
| 6.52.2. Key Features | 276 |
| 6.52.3. Creating the Component with a Page Tag | 276 |
| 6.52.4. Creating the Component Dynamically Using Java | 277 |
| 6.52.5. Details of Usage | 277 |

| | |
|---|-----|
| 6.52.6. Look-and-Feel Customization | 278 |
| 6.52.7. Relevant Resources Links | 278 |
| 6.53. < rich:jQuery > | 278 |
| 6.53.1. Description | 278 |
| 6.53.2. Key Features | 278 |
| 6.53.3. Creating the Component with a Page Tag | 279 |
| 6.53.4. Creating the Component Dynamically Using Java | 279 |
| 6.53.5. Details of Usage | 279 |
| 6.53.6. Look-and-Feel Customization | 283 |
| 6.53.7. Relevant Resources Links | 283 |
| 6.54. < rich:listShuttle > | 283 |
| 6.54.1. Description | 283 |
| 6.54.2. Key Features | 283 |
| 6.54.3. Creating the Component with a Page Tag | 286 |
| 6.54.4. Creating the Component Dynamically Using Java | 287 |
| 6.54.5. Details of Usage | 287 |
| 6.54.6. JavaScript API | 290 |
| 6.54.7. Look-and-Feel Customization | 290 |
| 6.54.8. Skin Parameters Redefinition | 291 |
| 6.54.9. Definition of Custom Style Classes | 293 |
| 6.54.10. Relevant Resources Links | 296 |
| 6.55. < rich:message > | 296 |
| 6.55.1. Description | 296 |
| 6.55.2. Key Features | 297 |
| 6.55.3. Creating the Component with a Page Tag | 299 |
| 6.55.4. Creating the Component Dynamically Using Java | 299 |
| 6.55.5. Details of Usage | 299 |
| 6.55.6. Look-and-Feel Customization | 300 |
| 6.55.7. Definition of Custom Style Classes | 300 |
| 6.55.8. Relevant Resources Links | 301 |
| 6.56. < rich:messages > | 301 |
| 6.56.1. Description | 301 |
| 6.56.2. Key Features | 301 |
| 6.56.3. Creating the Component with a Page Tag | 303 |
| 6.56.4. Creating the Component Dynamically Using Java | 304 |
| 6.56.5. Details of Usage | 304 |
| 6.56.6. Look-and-Feel Customization | 305 |
| 6.56.7. Definition of Custom Style Classes | 305 |
| 6.56.8. Relevant Resources Links | 305 |
| 6.57. < rich:modalPanel > | 306 |
| 6.57.1. Description | 306 |
| 6.57.2. Key Features | 306 |
| 6.57.3. Creating the Component with a Page Tag | 309 |
| 6.57.4. Creating the Component Dynamically Using Java | 309 |
| 6.57.5. Details of Usage | 309 |

| | |
|---|-----|
| 6.57.6. JavaScript API | 313 |
| 6.57.7. Look-and-Feel Customization | 313 |
| 6.57.8. Skin Parameters Redefinition | 313 |
| 6.57.9. Definition of Custom Style Classes | 314 |
| 6.57.10. Relevant Resources Links | 315 |
| 6.58. < rich:orderingList > | 315 |
| 6.58.1. Description | 315 |
| 6.58.2. Key Features | 316 |
| 6.58.3. Creating the Component with a Page Tag | 318 |
| 6.58.4. Creating the Component Dynamically Using Java | 318 |
| 6.58.5. Details of Usage | 319 |
| 6.58.6. JavaScript API | 322 |
| 6.58.7. Look-and-Feel Customization | 322 |
| 6.58.8. Skin Parameters Redefinition | 322 |
| 6.58.9. Definition of Custom Style Classes | 325 |
| 6.58.10. Relevant Resources Links | 328 |
| 6.59. < rich:paint2D > | 328 |
| 6.59.1. Description | 328 |
| 6.59.2. Key Features | 328 |
| 6.59.3. Creating the Component with a Page Tag | 331 |
| 6.59.4. Creating the Component Dynamically Using Java | 331 |
| 6.59.5. Details of Usage | 331 |
| 6.59.6. Look-and-Feel Customization | 332 |
| 6.59.7. Relevant Resources Links | 332 |
| 6.60. < rich:panel > | 333 |
| 6.60.1. Description | 333 |
| 6.60.2. Key Features | 333 |
| 6.60.3. Creating the Component with a Page Tag | 334 |
| 6.60.4. Creating the Component Dynamically Using Java | 334 |
| 6.60.5. Details of Usage | 335 |
| 6.60.6. | 335 |
| 6.60.7. Look-and-Feel Customization | 336 |
| 6.60.8. Skin Parameters Redefinition | 336 |
| 6.60.9. Definition of Custom Style Classes | 337 |
| 6.60.10. Relevant Resources Links | 338 |
| 6.61. < rich:panelBar > | 338 |
| 6.61.1. Description | 338 |
| 6.61.2. Key Features | 339 |
| 6.61.3. Creating the Component with a Page Tag | 341 |
| 6.61.4. Creating the Component Dynamically Using Java | 341 |
| 6.61.5. Details of Usage | 341 |
| 6.61.6. Look-and-Feel Customization | 341 |
| 6.61.7. Skin Parameters Redefinition | 342 |
| 6.61.8. Definition of Custom Style Classes | 342 |
| 6.61.9. Relevant Resources Links | 343 |

| | |
|---|-----|
| 6.62. < rich:panelBarItem > | 344 |
| 6.62.1. Description | 344 |
| 6.62.2. Key Features | 344 |
| 6.62.3. Creating the Component with a Page Tag | 345 |
| 6.62.4. Creating the Component Dynamically Using Java | 345 |
| 6.62.5. Details of Usage | 345 |
| 6.62.6. Look-and-Feel Customization | 346 |
| 6.62.7. Skin Parameters Redefinition | 346 |
| 6.62.8. Definition of Custom Style Classes | 346 |
| 6.63. < rich:panelMenu > | 348 |
| 6.63.1. Description | 348 |
| 6.63.2. Key Features | 348 |
| 6.63.3. Creating the Component with a Page Tag | 353 |
| 6.63.4. Creating the Component Dynamically Using Java | 353 |
| 6.63.5. Details of Usage | 353 |
| 6.63.6. JavaScript API | 355 |
| 6.63.7. Look-and-Feel Customization | 355 |
| 6.63.8. Definition of Custom Style Classes | 355 |
| 6.63.9. Relevant Resources Links | 355 |
| 6.64. < rich:panelMenuGroup > | 355 |
| 6.64.1. Description | 355 |
| 6.64.2. Key Features | 356 |
| 6.64.3. Creating the Component with a Page Tag | 361 |
| 6.64.4. Creating the Component Dynamically Using Java | 361 |
| 6.64.5. Details of Usage | 361 |
| 6.64.6. JavaScript API | 363 |
| 6.64.7. Look-and-Feel Customization | 363 |
| 6.64.8. Skin Parameters Redefinition | 363 |
| 6.64.9. Definition of Custom Style Classes | 364 |
| 6.64.10. Relevant resources links | 365 |
| 6.65. < rich:panelMenuItem > | 365 |
| 6.65.1. Description | 365 |
| 6.65.2. Key Features | 366 |
| 6.65.3. Creating the Component with a Page Tag | 369 |
| 6.65.4. Creating the Component Dynamically Using Java | 369 |
| 6.65.5. Details of Usage | 370 |
| 6.65.6. Look-and-Feel Customization | 371 |
| 6.65.7. Skin Parameters Redefinition | 371 |
| 6.65.8. Definition of Custom Style Classes | 372 |
| 6.65.9. Relevant resources links | 373 |
| 6.66. < rich:scrollableDataTable > | 373 |
| 6.66.1. Description | 373 |
| 6.66.2. Key Features | 373 |
| 6.66.3. Creating the Component with a Page Tag | 377 |
| 6.66.4. Creating the Component Dynamically Using Java | 378 |

| | |
|---|-----|
| 6.66.5. Details of Usage | 378 |
| 6.66.6. JavaScript API | 381 |
| 6.66.7. Look-and-Feel Customization | 381 |
| 6.66.8. Skin Parameters Redefinition | 381 |
| 6.66.9. Definition of Custom Style Classes | 382 |
| 6.66.10. Relevant Resources Links | 384 |
| 6.67. < rich:separator > | 384 |
| 6.67.1. Description | 384 |
| 6.67.2. Key Features | 384 |
| 6.67.3. Creating the Component with a Page Tag | 386 |
| 6.67.4. Creating the Component Dynamically Using Java | 386 |
| 6.67.5. Details of Usage | 386 |
| 6.67.6. Look-and-Feel Customization | 387 |
| 6.67.7. Definition of Custom Style Classes | 387 |
| 6.67.8. Relevant Resources Links | 387 |
| 6.68. < rich:simpleTogglePanel > | 387 |
| 6.68.1. Description | 387 |
| 6.68.2. Key Features | 388 |
| 6.68.3. Creating the Component with a Page Tag | 391 |
| 6.68.4. Creating the Component Dynamically Using Java | 391 |
| 6.68.5. Details of Usage | 391 |
| 6.68.6. Look-and-Feel Customization | 392 |
| 6.68.7. Skin Parameters Redefinition | 392 |
| 6.68.8. Definition of Custom Style Classes | 393 |
| 6.68.9. Relevant Resources Links | 394 |
| 6.69. < rich:spacer > | 394 |
| 6.69.1. Description | 394 |
| 6.69.2. Key Features | 394 |
| 6.69.3. Creating the Component with a Page Tag | 396 |
| 6.69.4. Creating the Component Dynamically Using Java | 396 |
| 6.69.5. Details of Usage | 396 |
| 6.69.6. Look-and-Feel Customization | 396 |
| 6.69.7. Relevant Resources Links | 397 |
| 6.70. < rich:suggestionbox > | 397 |
| 6.70.1. Description | 397 |
| 6.70.2. Key Features | 397 |
| 6.70.3. Creating the Component with a Page Tag | 403 |
| 6.70.4. Creating the Component Dynamically Using Java | 403 |
| 6.70.5. Details of Usage | 404 |
| 6.70.6. | 405 |
| 6.70.7. Look-and-Feel Customization | 406 |
| 6.70.8. Skin Parameters Redefinition | 407 |
| 6.70.9. Definition of Custom Style Classes | 408 |
| 6.70.10. Relevant Resources Links | 409 |
| 6.71. < rich:tabPanel > | 409 |

| | |
|---|-----|
| 6.71.1. Description | 409 |
| 6.71.2. Key Features | 409 |
| 6.71.3. Creating the Component with a Page Tag | 412 |
| 6.71.4. Creating the Component Dynamically Using Java | 412 |
| 6.71.5. Details of Usage | 412 |
| 6.71.6. Look-and-Feel Customization | 414 |
| 6.71.7. Skin Parameters Redefinition | 414 |
| 6.71.8. Definition of Custom Style Classes | 415 |
| 6.71.9. Relevant Resources Links | 416 |
| 6.72. < rich:tab > | 416 |
| 6.72.1. Description | 416 |
| 6.72.2. Key Features | 416 |
| 6.72.3. Creating the Component with a Page Tag | 419 |
| 6.72.4. Creating the Component Dynamically Using Java | 420 |
| 6.72.5. Details of Usage | 420 |
| 6.72.6. Look-and-Feel Customization | 421 |
| 6.72.7. Skin Parameters Redefinition | 422 |
| 6.72.8. Definition of Custom Style Classes | 423 |
| 6.73. < rich:togglePanel > | 423 |
| 6.73.1. Description | 423 |
| 6.73.2. Key Features | 424 |
| 6.73.3. Creating the Component with a Page Tag | 426 |
| 6.73.4. Creating the Component Dynamically Using Java | 426 |
| 6.73.5. Details of Usage | 427 |
| 6.73.6. Look-and-Feel Customization | 428 |
| 6.73.7. Definition of Custom Style Classes | 428 |
| 6.73.8. Relevant Resources Links | 428 |
| 6.74. < rich:toggleControl > | 428 |
| 6.74.1. Description | 428 |
| 6.74.2. Key Features | 429 |
| 6.74.3. Creating the Component with a Page Tag | 432 |
| 6.74.4. Creating the Component Dynamically Using Java | 432 |
| 6.74.5. Details of Usage | 433 |
| 6.74.6. Look-and-Feel Customization | 433 |
| 6.74.7. Definition of Custom Style Classes | 434 |
| 6.75. < rich:toolBar > | 434 |
| 6.75.1. Description | 434 |
| 6.75.2. Key Features | 434 |
| 6.75.3. Creating the Component with a Page Tag | 436 |
| 6.75.4. Creating the Component Dynamically Using Java | 436 |
| 6.75.5. Details of Usage | 436 |
| 6.75.6. Look-and-Feel Customization | 437 |
| 6.75.7. Skin Parameters Redefinition | 437 |
| 6.75.8. Definition of Custom Style Classes | 437 |
| 6.75.9. Relevant Resources Links | 438 |

| | |
|---|-----|
| 6.76. < rich:toolBarGroup > | 438 |
| 6.76.1. Description | 438 |
| 6.76.2. Key Features | 438 |
| 6.76.3. | 439 |
| 6.76.4. Creating the Component with a Page Tag | 439 |
| 6.76.5. Creating the Component Dynamically Using Java | 440 |
| 6.76.6. Details of Usage | 440 |
| 6.76.7. Look-and-Feel Customization | 441 |
| 6.76.8. Skin Parameters Redefinition | 441 |
| 6.77. < rich:toolTip > | 441 |
| 6.77.1. Description | 441 |
| 6.77.2. Key Features | 441 |
| 6.77.3. Creating the Component with a Page Tag | 443 |
| 6.77.4. Creating the Component Dynamically Using Java | 444 |
| 6.77.5. Details of Usage | 444 |
| 6.77.6. JavaScript API | 446 |
| 6.77.7. Look-and-Feel Customization | 446 |
| 6.77.8. Skin Parameters Redefinition | 446 |
| 6.77.9. Definition of Custom Style Classes | 447 |
| 6.77.10. Relevant Resources Links | 447 |
| 6.78. < rich:tree > | 447 |
| 6.78.1. Description | 447 |
| 6.78.2. Key Features | 447 |
| 6.78.3. Creating the Component with a Page Tag | 453 |
| 6.78.4. Creating the Component Dynamically Using Java | 453 |
| 6.78.5. Details of Usage | 453 |
| 6.78.6. Built-In Drag and Drop | 457 |
| 6.78.7. Events handling | 459 |
| 6.78.8. Look-and-Feel Customization | 460 |
| 6.78.9. Skin Parameters Redefinition: | 460 |
| 6.78.10. Definition of Custom Style Classes | 460 |
| 6.78.11. Relevant Resources Links | 460 |
| 6.79. < rich:treeNode > | 460 |
| 6.79.1. Description | 460 |
| 6.79.2. Key Features | 461 |
| 6.79.3. Creating the Component with a Page Tag | 464 |
| 6.79.4. Creating the Component Dynamically Using Java | 465 |
| 6.79.5. Details of Usage | 465 |
| 6.79.6. Built-in Drag and Drop | 466 |
| 6.79.7. Events Handling | 466 |
| 6.79.8. Look-and-Feel Customization | 466 |
| 6.79.9. Skin Parameters Redefinition | 467 |
| 6.79.10. Definition of Custom Style Classes | 467 |
| 6.79.11. Relevant Resources Links | 469 |
| 6.80. < rich:changeExpandListener > | 469 |

| | |
|---|-----|
| 6.80.1. Description | 469 |
| 6.80.2. Key Features | 469 |
| 6.80.3. Creating the Component with a Page Tag | 470 |
| 6.80.4. Creating the Component Dynamically Using Java | 470 |
| 6.80.5. Details of usage | 470 |
| 6.80.6. Look-and-Feel Customization | 471 |
| 6.81. < rich:nodeSelectListener > | 471 |
| 6.81.1. Description | 471 |
| 6.81.2. Key Features | 471 |
| 6.81.3. Creating the Component with a Page Tag | 471 |
| 6.81.4. Creating the Component Dynamically Using Java | 472 |
| 6.81.5. Details of usage | 472 |
| 6.81.6. Look-and-Feel Customization | 473 |
| 6.82. < rich:recursiveTreeNodesAdaptor > | 473 |
| 6.82.1. Description | 473 |
| 6.82.2. Key Features | 473 |
| 6.82.3. Creating the Component with a Page Tag | 474 |
| 6.82.4. Creating the Component Dynamically Using Java | 475 |
| 6.82.5. Details of Usage | 475 |
| 6.83. < rich:treeNodesAdaptor > | 476 |
| 6.83.1. Description | 476 |
| 6.83.2. Key Features | 476 |
| 6.83.3. Creating the Component with a Page Tag | 477 |
| 6.83.4. Creating the Component Dynamically Using Java | 477 |
| 6.83.5. Details of Usage | 477 |
| 6.83.6. Relevant Resources Links | 478 |
| 7. IDE Support | 479 |
| 8. Links to information resources | 480 |

Introduction

Rich Faces is an open source framework that adds Ajax capability into existing JSF applications without resorting to JavaScript.

Rich Faces leverages JavaServer Faces framework including lifecycle, validation, conversion facilities and management of static and dynamic resources. Rich Faces components with built-in Ajax support and a highly customizable look-and-feel can be easily incorporated into JSF applications.

Rich Faces allows to:

- Intensify the whole set of JSF benefits while working with Ajax. Rich Faces is fully integrated into the JSF lifecycle. While other frameworks only give you access to the managed bean facility, Rich Faces advantages the action and value change listeners, as well as invokes server-side validators and converters during the Ajax request-response cycle.
- Add Ajax capability to the existing JSF applications. Framework provides two components libraries (Core Ajax and UI). The Core library sets Ajax functionality into existing pages, so there is no need to write any JavaScript code or to replace existing components with new Ajax ones. Rich Faces enables page-wide Ajax support instead of the traditional component-wide support and it gives the opportunity to define the event on the page. An event invokes an Ajax request and areas of the page which become synchronized with the JSF Component Tree after changing the data on the server by Ajax request in accordance with events fired on the client.
- Create quickly complex View basing on out of the box components. Rich Faces UI library contains components for adding rich user interface features to JSF applications. It extends the Rich Faces framework to include a large (and growing) set of powerful rich Ajax-enabled components that come with extensive skins support. In addition, RichFaces components are designed to be used seamlessly with other 3d-party component libraries on the same page, so you have more options for developing your applications.
- Write your own custom rich components with built-in Ajax support. We're always working on improvement of Component Development Kit (CDK) that was used for Rich Faces UI library creation. The CDK includes a code-generation facility and a templating facility using a JSP-like syntax. These capabilities help to avoid a routine process of a component creation. The component factory works like a well-oiled machine allowing the creation of first-class rich components with built-in Ajax functionality even more easily than the creation of simpler components by means of the traditional coding approach.
- Package resources with application Java classes. In addition to its core, Ajax functionality of Rich Faces provides an advanced support for the different resources management: pictures, JavaScript code, and

CSS stylesheets. The resource framework makes possible to pack easily these resources into Jar files along with the code of your custom components.

- Easily generate binary resources on-the-fly. Resource framework can generate images, sounds, Excel spreadsheets etc.. on-the-fly so that it becomes for example possible to create images using the familiar approach of the "Java Graphics2D" library.
- Create a modern rich user interface look-and-feel with skins-based technology. Rich Faces provides a skinnability feature that allows easily define and manage different color schemes and other parameters of the UI with the help of named skin parameters. Hence, it is possible to access the skin parameters from JSP code and the Java code (e.g. to adjust generated on-the-fly images based on the text parts of the UI). RichFaces comes with a number of predefined skins to get you started, but you can also easily create your own custom skins.
- Test and create the components, actions, listeners, and pages at the same time. An automated testing facility is in our roadmap for the near future. This facility will generate test cases for your component as soon as you develop it. The testing framework will not just test the components, but also any other server-side or client-side functionality including JavaScript code. What is more, it will do all of this without deploying the test application into the Servlet container.

Rich Faces UI components come ready to use out-of-the-box, so developers save their time and immediately gain the advantage of the mentioned above features in Web applications creation. As a result, usage experience can be faster and easily obtained.

Technical Requirements

RichFaces was developed with an open architecture to be compatible with the widest possible variety of environments.

This is what you need to start working with RichFaces 3.1.0:

- Java
- JavaServer Faces
- Java application server or servlet container
- Browser (on client side)
- Richfaces framework

2.1. Supported Java Versions

- JDK 1.4 and higher

2.2. Supported JavaServer Faces Implementations

- Sun JSF 1.1 RI - 1.2
- MyFaces 1.1.1 - 1.2
- Facelets JSF 1.1.1 - 1.2
- Seam 1.2. - 2.0

2.3. Supported Servers

- Apache Tomcat 4.1 - 6.0
- IBM WebSphere 5.1 - 6.0
- BEA WebLogic 8.1 - 9.0
- Oracle AS/OC4J 10.1.3
- Resin 3.0

- Jetty 5.1.X
- Sun Application Server 8 (J2EE 1.4)
- Glassfish (J2EE 5)
- JBoss 3.2 - 4.2.x
- Sybase EAServer 6.0.1

2.4. Supported Browsers

- Internet Explorer 6.0 - 7.0
- Firefox 1.5 - 2.0
- Opera 8.5 - 9.0
- Netscape 7.0
- Safari 2.0

Note:

Safari 3.0 Beta is not supported.

This list is composed basing on reports received from our users. We assume the list can be incomplete and absence of your environment in the list doesn't mean incompatibility.

We appreciate your feedback on platforms and browsers that aren't in the list but are compatible with RichFaces. It helps us to keep the list up-to-date.

Getting Started with RichFaces

3.1. Downloading RichFaces 3.1.0

The latest release of RichFaces is available for download at:

<http://labs.jboss.com/jbossrichfaces/downloads>
in the RichFaces project area under JBoss.

3.2. Installation

- Unzip "*richfaces-ui-3.1.0-bin.zip*" file to the chosen folder.
- Copy "*richfaces-api-3.1.0.jar*", "*richfaces-impl-3.1.0.jar*", "*richfaces-ui-3.1.0.jar*" files into the "*WEB-INF/lib*" folder of your application.
- Add the following content into the "*WEB-INF/web.xml*" file of your application:

```
...  
<context-param>  
  <param-name>org.richfaces.SKIN</param-name>  
  <param-value>blueSky</param-value>  
</context-param>  
<filter>  
  <display-name>RichFaces Filter</display-name>  
  <filter-name>richfaces</filter-name>  
  <filter-class>org.ajax4jsf.Filter</filter-class>  
</filter>  
<filter-mapping>  
  <filter-name>richfaces</filter-name>  
  <servlet-name>Faces Servlet</servlet-name>  
  <dispatcher>REQUEST</dispatcher>  
  <dispatcher>FORWARD</dispatcher>  
  <dispatcher>INCLUDE</dispatcher>  
</filter-mapping>
```

- Add the following lines for each JSP page of your application.

```
<%@ taglib uri="http://richfaces.org/a4j" prefix="a4j"%>  
<%@ taglib uri="http://richfaces.org/rich" prefix="rich"%>
```

For XHTML pages:

```
<xmlns:a4j="http://richfaces.org/a4j">
```

```
<xmlns:rich="http://richfaces.org/rich">
```

Note:

The previous namespaces URIs (<https://ajax4jsf.dev.java.net/ajax> and <http://richfaces.ajax4jsf.org/rich>) are also available for backward compatibility.

3.3. Simple Ajax Echo Project

In our JSF project you need only one JSP page that has a form with a couple of child tags: **<h:inputText>** and **<h:outputText>**.

This simple application let you input some text into the **<h:inputText>**, send data to the server, and see the server response as a value of **<h:outputText>**.

3.3.1. JSP Page

Here is the necessary page (echo.jsp):

```
<%@ taglib uri="http://richfaces.org/a4j" prefix="a4j"%>
<%@ taglib uri="http://richfaces.org/rich" prefix="rich"%>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<html>
  <head>
    <title>repeater </title>
  </head>
  <body>
    <f:view>
      <h:form>
        <rich:panel header="Simple Echo">
          <h:inputText size="50" value="#{bean.text}" >
            <a4j:support event="onkeyup" reRender="rep"/>
          </h:inputText>
          <h:outputText value="#{bean.text}" id="rep"/>
        </rich:panel>
      </h:form>
    </f:view>
  </body>
</html>
```

Only two tags distinguish this page from a "regular" JSF one. There are **<rich:panel>** and **<a4j:support>**.

The **<rich:panel>** allows to place the page elements in rectangle panel that can be skinned.

The **<a4j:support>** with corresponding attributes (as it was shown in the previous example) adds an Ajax support to the parent **<h:inputText>** tag. This support is bound to "onkeyup" JavaScript event, so that each time when this event is fired on the parent tag, our application sends an Ajax request to the server. It means that the text field pointed to our managed bean property contains up-to-date value of our input.

The value of *"reRender"* attribute of the `<a4j:support>` tag defines which part(s) of our page is (are) to be updated. In this case, the only part of the page to update is the `<h:outputText>` tag because its ID value matches to the value of *"reRender"* attribute. As you see, it's not difficult to update multiple elements on the page, only list their IDs as the value of *"reRender"*.

3.3.2. Data Bean

In order to build this application, you should create a managed bean:

```
package demo;

public class Bean {
    private String text;
    public Bean() {
    }
    public String getText() {
        return text;
    }
    public void setText(String text) {
        this.text = text;
    }
}
```

3.3.3. faces-config.xml

Next, it's necessary to register your bean inside of the `faces-config.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE faces-config PUBLIC "-//Sun Microsystems, Inc.//DTD JavaServer Faces
Config 1.1//EN"
"http://java.sun.com/dtd/web-facesconfig_1_1.dtd">
<faces-config>
    <managed-bean>
        <managed-bean-name>bean</managed-bean-name>
        <managed-bean-class>demo.Bean</managed-bean-class>
        <managed-bean-scope>request</managed-bean-scope>
        <managed-property>
            <property-name>text</property-name>
            <value/>
        </managed-property>
    </managed-bean>
</faces-config>
```

Note:

Nothing that relates directly to RichFaces is required in the configuration file.

3.3.4. Web.xml

It is also necessary to add jar files (see installation chapter) and modify the `"web.xml"` file:

```
<?xml version="1.0"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
```

```

    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">
    <display-name>a4jEchoText</display-name>
    <context-param>
        <param-name>org.richfaces.SKIN</param-name>
        <param-value>blueSky</param-value>
    </context-param>
    <context-param>
        <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
        <param-value>server</param-value>
    </context-param>
    <filter>
        <display-name>RichFaces Filter</display-name>
        <filter-name>richfaces</filter-name>
        <filter-class>org.ajax4jsf.Filter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>richfaces</filter-name>
        <servlet-name>Faces Servlet</servlet-name>
        <dispatcher>REQUEST</dispatcher>
        <dispatcher>FORWARD</dispatcher>
        <dispatcher>INCLUDE</dispatcher>
    </filter-mapping>
    <listener>
        <listener-class>com.sun.faces.config.ConfigureListener</listener-class>
    </listener>

    <!-- Faces Servlet -->
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <!-- Faces Servlet Mapping -->
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.jsf</url-pattern>
    </servlet-mapping>
    <login-config>
        <auth-method>BASIC</auth-method>
    </login-config>
</web-app>

```

Now your application should work.

3.3.5. Deployment

Finally, you should be able to place this application on your Web server. To start your project, point your browser at <http://localhost:8080/a4jEchoText/echo.jsf>

Settings for different environments

RichFaces comes with support for all tags (components) included in the JavaServer Faces specification. To add RichFaces capabilities to the existing JSF project you should just put the RichFaces libraries into the lib folder of the project and add filter mapping. The behavior of the existing project doesn't change just because of RichFaces.

4.1. Web Application Descriptor Parameters

RichFaces doesn't require any parameters to be defined in your web.xml. But the RichFaces parameters listed below may help with development and may increase the flexibility of RichFaces usage.

Table 4.1. Initialization Parameters

| Name | Default | Description |
|----------------------------------|---------|--|
| org.richfaces.SKIN | DEFAULT | Is a name of a skin used in an application. It can be a literal string with a skin name, or the <i>EL</i> expression (<code>#{...}</code>) pointed to a <i>String</i> property (skin name) or a property of a <code>org.richfaces.framework.skin</code> type. Skin in last case, this instance is used as a current skin |
| org.richfaces.LoadScriptStrategy | DEFAULT | Defines how the RichFaces script files are loaded to application. Possible values are: ALL, DEFAULT, NONE. For more information see "Scripts and Styles Load Strategy". |
| org.richfaces.LoadStyleStrategy | DEFAULT | Defines how the RichFaces style files are loaded to application. Possible values are: ALL, DEFAULT, NONE. For more information see "Scripts and Styles Load Strategy". |
| org.ajax4jsf.LOGFILE | none | |

| Name | Default | Description |
|------------------------------------|---------|---|
| | | Is an URL to an application or a container log file (if possible). If this parameter is set, content from the given URL is shown on a <i>Debug</i> error page in the <i>iframe</i> window |
| org.ajax4jsf.VIEW_HANDLERS | none | Is a comma-separated list of custom <i>ViewHandler</i> instances for inserting in chain. Handlers are inserted BEFORE RichFaces viewhandlers in the given order. For example, in facelets application this parameter must contain com.sun.facelets.FaceletViewHandler, instead of declaration in faces-config.xml |
| org.ajax4jsf.CONTROL_COMPONENTS | none | Is a comma-separated list of names for a component as a special control case, such as messages bundle loader, alias bean components, etc. Is a type of component got by a reflection from the static field <i>COMPONENT_TYPE</i> . For components with such types encode methods always are called in rendering Ajax responses, even if a component isn't in an updated part |
| org.ajax4jsf.ENCRYPT_RESOURCE_DATA | False | For generated resources, such as encrypt generation data, it's encoded in the resource URL. For example, URL for an image generated from the <i>mediaOutput</i> component contains a name of a generation method, since for a hacker attack, it is possible to create a request for any JSF baked beans or other attributes. To prevent such attacks, set this parameter to "true" in critical applications (works with JRE > 1.4) |

| Name | Default | Description |
|-------------------------------|---------|---|
| org.ajax4jsf.ENCRYPT_PASSWORD | Random | Is a password for encryption of resources data. If isn't set, a random password is used |
| org.ajax4jsf.COMPRESS_SCRIPT | true | It doesn't allow framework to reformat JavaScript files (makes it impossible to debug) |

Note:

org.richfaces.SKIN is used in the same way as org.ajax4jsf.SKIN

Table 4.2. org.ajax4jsf.Filter Initialization Parameters

| Name | Default | Description |
|-----------------|---------|---|
| log4j-init-file | - | Is a path (relative to web application context) to the <i>log4j.xml</i> configuration file, it can be used to setup per-application custom logging |
| enable-cache | true | Enable caching of framework-generated resources (JavaScript, CSS, images, etc.). For debug purposes development custom JavaScript or Style prevents to use old cached data in a browser |
| forceparser | true | Force parsing by a filter <i>HTML</i> syntax checker on any JSF page. If "false", only Ajax responses are parsed to syntax check and conversion to well-formed XML. Setting to "false" improves performance, but can provide visual effects on Ajax updates |

4.2. Sun JSF RI

RichFaces works with any implementation of JSF (both JSF 1.1 and JSF 1.2) and with most JSF component libraries without any additional settings. For more information look at:

java.sun.com [<http://java.sun.com/javaee/javaxserverfaces/>]

4.3. Apache MyFaces

RichFaces works with all Apache MyFaces versions (1.1.1 - 1.1.6) including specific libraries like TOMAHAWK Sandbox and Trinidad (the previous ADF Faces). However, there are some considerations to take into account for configuring applications to work with MyFaces and RichFaces.

There are some problems with different filters defined in the web.xml file clashing. To avoid these problems, the RichFaces filter must be the first one among other filters in the web.xml configuration file.

For more information look at: <http://myfaces.apache.org>

There's one more problem while using MyFaces + Seam. If you use this combination you should use `<a4j:page>` inside `<f:view>` (right after it in your code) wrapping another content inside your pages because of some problems in realization of `<f:view>` in myFaces.

The problem is to be overcome in the nearest future.

4.4. Facelets Support

A high-level support for Facelets is one of our main support features. When working with RichFaces, there is no difference what release of Facelets is used.

You should also take into account that some JSF frameworks such as Facelets use their own ViewHandler and need to have it first in the chain of ViewHandlers and the RichFaces AjaxViewHandler is not an exception. At first RichFaces installs its ViewHandler in any case, so in case of two frameworks, for example RichFaces + Facelets, no changes in settings are required. Although, when more then one framework (except RichFaces) is used, it's possible to use the VIEW_HANDLERS parameter defining these frameworks view handlers according to its usage order in it. For example, the declaration:

Example:

```
<context-param>
  <param-name>org.ajax4jsf.VIEW_HANDLERS</param-name>
  <param-value>com.sun.facelets.FaceletViewHandler</param-value>
</context-param>
```

says that Facelets will officially be the first, however AjaxViewHandler will be a little ahead temporarily to do some small, but very important job.

Note:

In this case you don't have to define FaceletViewHandler in the WEB-INF/faces-config.xml.

4.5. JBoss Seam Support

RichFaces now works out-of-the-box with JBoss Seam and Facelets running inside JBoss AS 4.0.4 and higher. There is no more shared JAR files needed. You just have to package the RichFaces library with your application.

Your web.xml for Seam 1.2 must be like this:

```

<?xml version="1.0" ?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
         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="2.4">

    <!-- richfaces -->

    <filter>
        <display-name>RichFaces Filter</display-name>
        <filter-name>richfaces</filter-name>
        <filter-class>org.ajax4jsf.Filter</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>richfaces</filter-name>
        <url-pattern>*.seam</url-pattern>
    </filter-mapping>

    <context-param>
        <param-name>org.ajax4jsf.VIEW_HANDLERS</param-name>
        <param-value>org.jboss.seam.ui.facelet.SeamFaceletViewHandler</param-value>
    </context-param>

    <!-- Seam -->

    <listener>
        <listener-class>org.jboss.seam.servlet.SeamListener</listener-class>
    </listener>

    <servlet>
        <servlet-name>Seam Resource Servlet</servlet-name>
        <servlet-class>org.jboss.seam.servlet.ResourceServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>Seam Resource Servlet</servlet-name>
        <url-pattern>/seam/resource/*</url-pattern>
    </servlet-mapping>

    <filter>
        <filter-name>Seam Filter</filter-name>
        <filter-class>org.jboss.seam.web.SeamFilter</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>Seam Filter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <!-- MyFaces -->

    <listener>

        <listener-class>org.apache.myfaces.webapp.StartupServletContextListener</listener-
class>

```

```

</listener>

<!-- JSF -->

<context-param>
    <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
    <param-value>client</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>*.seam</url-pattern>
</servlet-mapping>

</web-app>

```

Seam 2 supports RichFaces Filter. Thus your web.xml for Seam 2 must be like this:

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    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-app_2_5.xsd">

    <context-param>
        <param-name>org.ajax4jsf.VIEW_HANDLERS</param-name>
        <param-value>com.sun.facelets.FaceletViewHandler</param-value>
    </context-param>

    <!-- Seam -->

    <listener>
        <listener-class>org.jboss.seam.servlet.SeamListener</listener-class>
    </listener>

    <servlet>
        <servlet-name>Seam Resource Servlet</servlet-name>
        <servlet-class>org.jboss.seam.servlet.SeamResourceServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>Seam Resource Servlet</servlet-name>
        <url-pattern>/seam/resource/*</url-pattern>
    </servlet-mapping>

```



```

<filter>
  <filter-name>Seam Filter</filter-name>
  <filter-class>org.jboss.seam.servlet.SeamFilter</filter-class>
</filter>

<filter-mapping>
  <filter-name>Seam Filter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>

<!-- JSF -->

<context-param>
  <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
  <param-value>.xhtml</param-value>
</context-param>

<context-param>
  <param-name>facelets.DEVELOPMENT</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>*.seam</url-pattern>
</servlet-mapping>

</web-app>

```

Only one issue still persists while using Seam with MyFaces. Look at myFaces part of this section.

4.6. Portlet Support

JBoss Portlets have support since version Ajax4jsf 1.1.1. This support is improved in Richfaces 3.1.0. Provide your feedback on compatible with RichFaces if you face some problems.

4.7. Sybase EAServer

The load-on-startup for the Faces Servlet had to be set to 0 in web.xml.

Example:

```

...
  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>0</load-on-startup>
  </servlet>

```

```
</servlet>  
...
```

This is because, EAServer calls servlet init() before the ServletContextInitializer. Not an EAServer bug, this is in Servlet 2.3 spec.

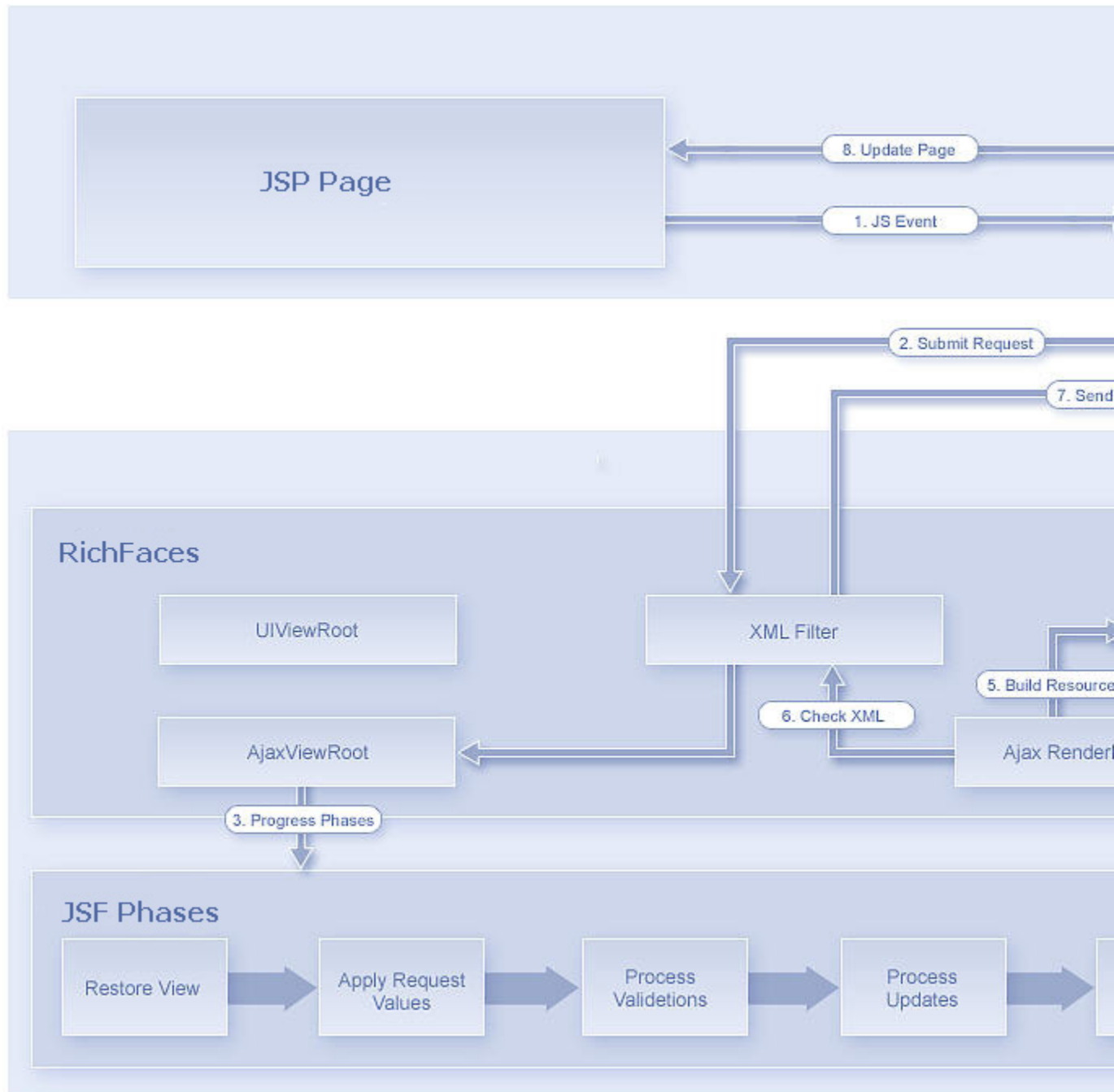
Basic concepts of the RichFaces Framework

5.1. Introduction

The framework is implemented as a component library which adds Ajax capability into existing pages, so you don't need to write any JavaScript code or to replace existing components with new Ajax widgets. RichFaces enables page-wide Ajax support instead of the traditional component-wide support. Hence, you can define the event on the page that invokes an Ajax request and the areas of the page that should be synchronized with the JSF Component Tree after the Ajax request changes the data on the server according to the events fired on the client.

Next Figure shows how it works:

Figure 5.1. Request Processing flow

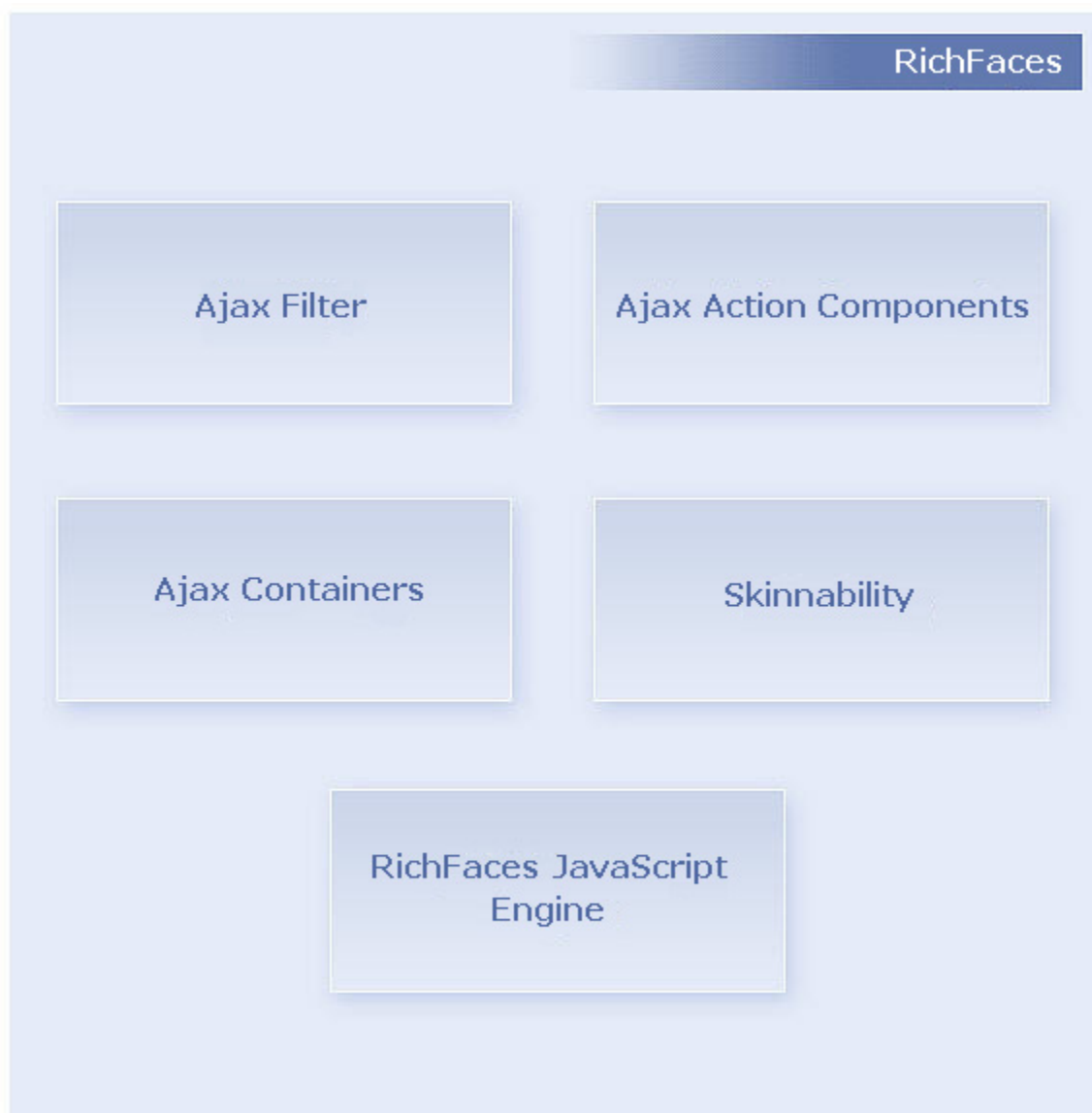


RichFaces allows to define (by means of JSF tags) different parts of a JSF page you wish to update with an Ajax request and provide a few options to send Ajax requests to the server. Also JSF page doesn't change from a "regular" JSF page and you don't need to write any JavaScript or XMLHttpRequest objects by hands, everything is done automatically.

5.2. RichFaces Architecture Overview

Next figure lists several important elements of the RichFaces framework

Figure 5.2. Core Ajax component structure

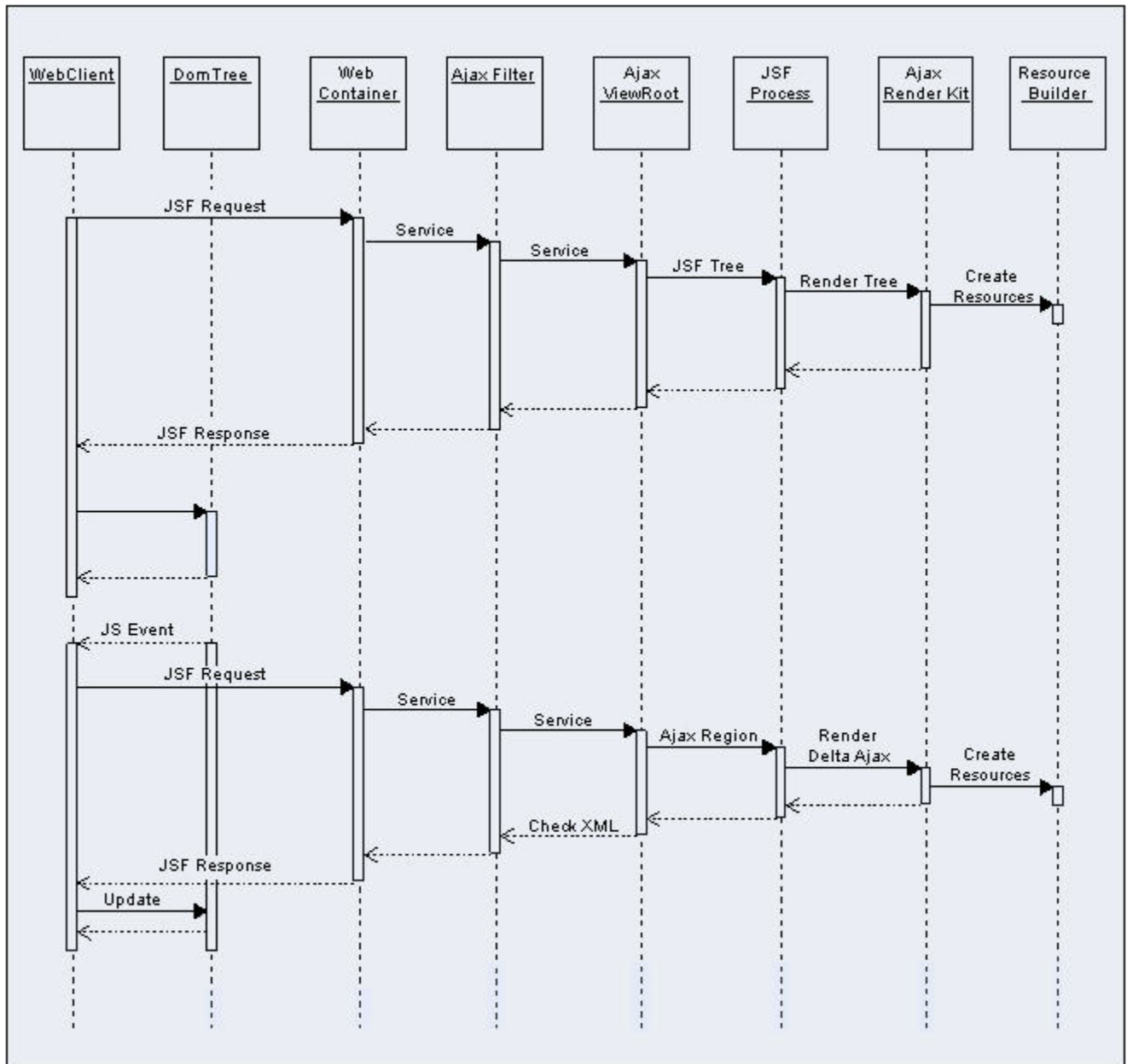


Ajax Filter. To get all benefits of RichFaces, you should register a Filter in web.xml file of your application. The Filter recognizes multiple request types. Necessary information about Filter configuration can be found in the "Filter configuration" section. The sequence diagram on Figure 3 shows the difference in processing of a "regular" JSF request and an Ajax request.

In the first case the whole JSF tree will be encoded, in the second one option it depends on the "size" of the Ajax region. As you can see, in the second case the filter parses the content of an Ajax response before sending it to the client side.

Have a look at the next picture to understand these two ways:

Figure 5.3. Request Processing sequence diagram

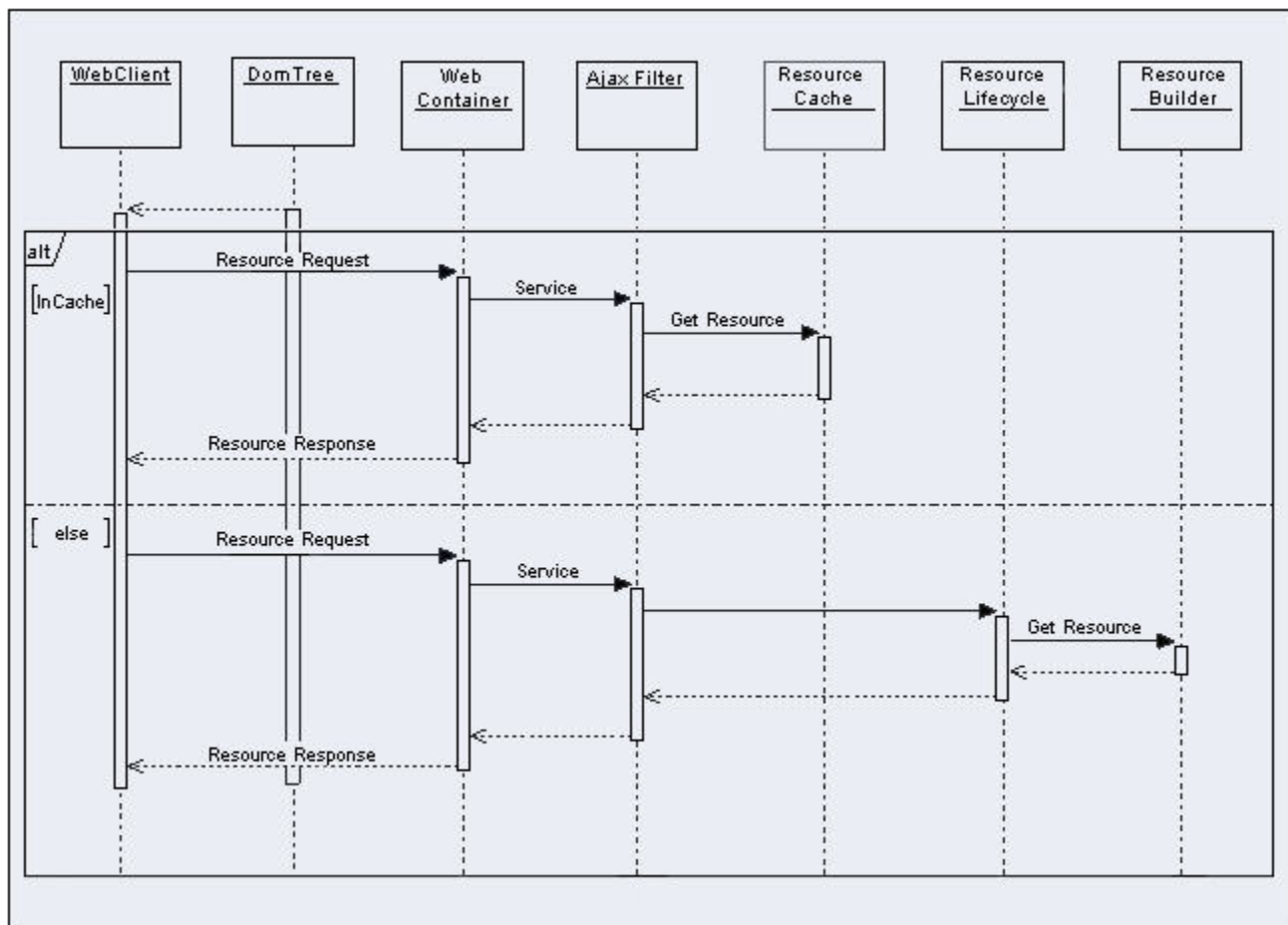


In both cases, the information about required static or dynamic resources that your application requests is registered in the ResourceBuilder class.

When a request for a resource comes (Figure 4), the RichFaces filter checks the Resource Cache for this resource and if it is there, the resource is sent to the client. Otherwise, the filter searches for the resource among those that are registered by the ResourceBuilder. If the resource is registered, the RichFaces filter will send a request to the ResourceBuilder to create (deliver) the resource.

Next Figure shows the ways of resource request processing.

Figure 5.4. Resource request sequence diagram



AJAX Action Components. There are Ajax Action Components: AjaxCommandButton, AjaxCommandLink, AjaxPoll and AjaxSupport and etc. You can use them to send Ajax requests from the client side.

AJAX Containers. AjaxContainer is an interface that describes an area on your JSF page that should be decoded during an Ajax request. AjaxViewRoot and AjaxRegion are implementations of this interface.

JavaScript Engine. RichFaces JavaScript Engine runs on the client-side. It knows how to update different areas on your JSF page based on the information from the Ajax response. Do not use this JavaScript code directly, as it is available automatically.

5.3. Limitations and Rules

In order to create RichFaces applications properly, keep the following points in mind:

- Any Ajax framework should not append or delete, but only replace elements on the page. For successful updates, an element with the same ID as in the response must exist on the page. If you'd like to append any code to a page, put in a placeholder for it (any empty element). For the same reason, it's recommended to place messages in the *"AjaxOutput"* component (as no messages is also a message).
- Don't use `<f:verbatim>` for self-rendered containers, since this component is transient and not saved in the tree.

- Ajax requests are made by XMLHttpRequest functions in XML format, but this XML bypasses most validations and the corrections that might be made in a browser. Thus, create only a strict standards-compliant code for HTML and XHTML, without skipping any required elements or attributes. Any necessary XML corrections are automatically made by the XML filter on the server, but lot's of unexpected effects can be produced by an incorrect HTML code.

5.4. Ajax Request Optimization

5.4.1. Re-Rendering

Ajax attributes are common for Ajax components such as `<a4j:support>` , `<a4j:commandButton>` , `<a4j:jsFunction>` , `<a4j:poll>` , `<a4j:push>` and so on. Also, most RichFaces components with built-in Ajax support have these attributes for a similar purpose. Ajax components attributes help RichFaces to expose its features. Most of the attributes have default values. Thus, you can start working with RichFaces without knowing the usage of these attribute. However, their usage allows to tune the required Ajax behavior very smoothly.

"reRender" is a key attribute. The attribute allows to point to area(s) on a page that should be updated as a response on Ajax interaction. The value of the *"reRender"* attribute is an id of the JSF component or an id list.

A simple example is placed below:

```
...
    <a4j:commandButton value="update" reRender="infoBlock" />
    ...
    <h:panelGrid id="infoBlock">
    ...
    </h:panelGrid>
...
```

The value of *"reRender"* attribute of the `<a4j:commandButton>` tag defines which part(s) of your page is (are) to be updated. In this case, the only part of the page to update is the `<h:panelGrid>` tag because its ID value matches to the value of *"reRender"* attribute. As you see, it's not difficult to update multiple elements on the page, only list their IDs as the value of *"reRender"* .

"reRender" uses `UIComponent.findComponent()` algorithm [[http://java.sun.com/javaee/javaxserverfaces/1.2_MR1/docs/api/javax/faces/component/UIComponent.html#findComponent\(java.lang.String\)](http://java.sun.com/javaee/javaxserverfaces/1.2_MR1/docs/api/javax/faces/component/UIComponent.html#findComponent(java.lang.String))] (with some additional exceptions) to find the component in the component tree. As can you see, the algorithm presumes several steps. Each other step is used if the previous step is not successful. Therefore, you can define how fast the component is found mentioning it more precisely. The following example shows the difference in approaches (both buttons will work successfully):

```
...
    <h:form id="form1">
    ...
    <a4j:commandButton value="Usual Way" reRender="infoBlock, infoBlock2" />
    <a4j:commandButton value="Shortcut" reRender=":infoBlock1,sv:infoBlock2" />
    ...
```



```

</h:form>
<h:panelGrid id="infoBlock">
    ...
</h:panelGrid>
...
<f:subview id="sv">
    <h:panelGrid id="infoBlock2">
        ...
    </h:panelGrid>
    ...
</f:subview>
...

```

It's also possible to use JSF EL expression as a value of the `reRender` attribute. It might be a property of types `Set`, `Collection`, `Array` or simple `String`. The EL for `reRender` is resolved right before the `Render Response` phase. Hence, you can calculate what should be re-rendered on any previous phase during the Ajax request processing.

Most common problem with using `reRender` is pointing it to the component that has a `"reRender"` attribute. Note, that JSF does not mark the place in the browser DOM where the outcome of the component should be placed in case the `"rendered"` condition returns false. Therefore, after the component becomes rendered during the Ajax request, RichFaces delivers the rendered code to the client, but does not update a page, because the place for update is unknown. You need to point to one of the parent components that has no `"rendered"` attribute. As an alternative, you can wrap the component with `<a4j:outputPanel>` `layout="none"`.

`"ajaxRendered"` attribute of the `<a4j:outputPanel>` set to `"true"` allows to define the area of the page that will be re-rendered even if it is not pointed in the `reRender` attribute explicitly. It might be useful if you have an area on a page that should be updated as a response on any Ajax request. For example, the following code allows to output error messages regardless of what Ajax request causes the `Validation` phase failed.

```

...
    <a4j:outputPanel ajaxRendered="true">
        <h:messages />
    </a4j:outputPanel>
...

```

`"limitToList"` attribute allows to dismiss the behavior of the `<a4j:outputPanel>` `"ajaxRendered"` attribute. `"limitToList" = "false"` means to update only the area(s) that mentioned in the `"reRender"` attribute explicitly. All output panels with `"ajaxRendered"="true"` is ignored. An example is placed below:

```

...
    <h:form>
        <h:inputText value="#{person.name}">
            <a4j:support event="onkeyup" reRender="test" limitToList="true"/>
        </h:inputText>
        <h:outputText value="#{person.name}" id="test"/>
    </form>
...

```

5.4.2. Queue and Traffic Flood Protection

"eventsQueue" attribute defines the name of the queue that will be used to order upcoming Ajax requests. By default, RichFaces does not queue Ajax requests. If events are produced simultaneously, they will come to the server simultaneously. JSF implementations (especially, the very first ones) does not guaranty that the request that comes first will be served or passed into the JSF lifecycle first. The order how the server side data will be modified in case of simultaneous request might be unpredictable. Usage of *eventsQueue* attribute allows to avoid possible mess. Define the queue name explicitly, if you expect intensive Ajax traffic in your application.

The next request posted in the same queue will wait until the previous one is not processed and Ajax Response is returned back if the *"eventsQueue"* attribute is defined. In addition, RichFaces starts to remove from the queue "similar" requests. "Similar" requests are the requests produced by the same event. For example, according to the following code, only the newest request will be sent to the server if you type very fast and has typed the several characters already before the previous Ajax Response is back.

```
...
    <h:inputText value="#{userBean.name}">
        <a4j:support event="onkeyup" eventsQueue="foo" reRender="bar" />
    </h:inputText>
...
```

"requestDelay" attribute defines the time (in ms) that the request will be wait in the queue before it is ready to send. When the delay time is over, the request will be sent to the server or removed if the newest "similar" request is in a queue already .

"ignoreDupResponses" attribute orders to ignore the Ajax Response produced by the request if the newest "similar" request is in a queue already. *"ignoreDupResponses"="true"* does not cancel the request while it is processed on the server, but just allows to avoid unnecessary updates on the client side if the response loses the actuality.

Defining the *"eventsQueue"* along with *"requestDelay"* allows to protect against unnecessary traffic flood and synchronizes Ajax requests order. If you have several sources of Ajax requests, you can define the same queue name there. This might be very helpful if you have Ajax components that invoke request asynchronously from the ones produced by events from users. For example, **<a4j:poll>** or **<a4j:push>** . In case the requests from such components modify the same data, the synchronization might be very helpful.

More information can be found on the RichFaces Users Forum [<http://jboss.com/index.html?module=bb&op=viewtopic&t=105766>].

"timeout" attribute is used for setting response waiting time on a particular request. If a response is not received during this time, the request is aborted.

5.4.3. Data Processing Options

RichFaces uses form based approach for Ajax request sending. This means each time, when you click an Ajax button or **<a4j:poll>** produces an asynchronous request, the data from the closest JSF form is submitted with the XMLHttpRequest object. The form data contains the values from the form input element and auxiliary information such as state saving data.

When *"ajaxSingle"* attribute value is *"true"*, it orders to include only a value of the current component (along with *<f:param>* or *<a4j:action>* param values if any) to the request map. In case of *<a4j:support>*, it is a value of the parent component. An example is placed below:

```
...
<h:form>
  <h:inputText value="#{person.name}">
    <a4j:support event="onkeyup" reRender="test" ajaxSingle="true"/>
  </h:inputText>
  <h:inputText value="#{person.middleName}" />
</form>
...
```

In this example the request contains only the input component causes the request generation, not all the components contained on a form, because of *"ajaxSingle"="true"* usage.

Note, that *"ajaxSingle"="true"* reduces the upcoming traffic, but does not prevent decoding other input components on the server side. Some JSF components, such as *<h:selectOneMenu>* do recognize the missing data in the request map value as a null value and try to pass the validation process with a failed result. Thus, use *<a4j:region>* to limit a part of the component tree that will be processed on the server side when it is required.

"immediate" attribute has the same purpose as any other non-JSF component. The default *"ActionListener"* should be executed immediately (i.e. during the Apply Request Values phase of a request processing lifecycle), rather than waiting until the Invoke Application phase. Using *immediate="true"* is one of the ways to have some data model values updated when other cannot be updated because of a problem with passing the Validation phase successfully. This might be important inside the *<h:dataTable>* like components where using *<a4j:region>* is impossible due to the *<h:dataTable>* component architecture.

"bypassUpdates" attribute allows to bypass the Update Model phase. It might be useful if you need to check your input against the available validator, but not to update the model with those data. Note, that an action will be invoked at the end of the Validation phase only if the Validation phase is passed successfully. The listeners of the Application phase will not be invoked in any case.

5.4.4. Action and Navigation

Ajax component is similar to any other non-Ajax JSF component like *<h:commandButton>*. It allows to submit the form. You can use *"action"* and *"actionListener"* attribute to invoke the action method and define the action event.

"action" method must return null if you want to have an Ajax Response with a partial page update. This is regular mode called "Ajax request generates Ajax Response". In case of action does not return null, but the action outcome that matches one of navigation rules, RichFaces starts to work in "Ajax request generates Non-Ajax Response" mode. This mode might be helpful in two major cases:

- RichFaces allows to organize a page flow inside the *<a4j:include>* component. This is a typical scenario for Wizard like behavior. The new content is rendered inside the *<a4j:include>* area. The content is taken from the navigation rule of the faces configuration file (usually, the faces-config.xml). Note, that the content of the "wizard" is not isolated from the rest of the page. The included page should

not have own `<f:view>` (it does not matter if you use facelets). You need to have an Ajax component inside the `<a4j:include>` to navigate between the wizard pages. Otherwise, the whole page update will be performed.

- If you want to involve the server side validators and navigate to the next page only if the Validation phase is passed successfully, you can replace `<h:commandButton>` with `<a4j:commandButton>` and point to the action method that navigates to the next page. If Validation process fails, the partial page update will occur and you will see an error message. Otherwise, the application proceeds to the next page. Make sure, you define `<redirect/>` option for the navigation rule to avoid memory leaks.

5.4.5. JavaScript Interactions

RichFaces allows writing Ajax-enabled JSF application without writing any Javascript code. However, you can still invoke the javascript code if you need. There are several ajax attributes that helps to do it.

`"onsubmit"` attribute allows to invoke JavaScript code before an Ajax request is sent. If `"onsubmit"` returns "false", the Ajax request is canceled. The code of `"onsubmit"` is inserted before the RichFaces Ajax call. Hence, the `"onsubmit"` should not has a "return" statement if you want the Ajax request to be sent. If you are going to invoke a JavaScript function that returns "true" or "false", use the conditional statement to return something only when you need to cancel the request. For example:

```
...
    onsubmit="if (mynosendfunct()==false){return false}"
...
```

`"onclick"` attribute is similar to the `"onsubmit"`, but for clickable components such as `<a4j:commandLink>` and `<a4j:commandButton>`. If it returns "false", the Ajax request is canceled also.

`"oncomplete"` attribute allows to invoke the JavaScript code right after the Ajax Response is returned back and the DOM tree of the browser is updated. Richfaces registers the code for further invocation of XMLHttpRequest object before an Ajax request is sent. This means the code will not be changed during processing of the request on the server if you use JSF EL value binding. Also, you cannot use "this" inside the code, because it will not point the component where Ajax request was initiated.

`"onbeforedomupdate"` attribute defines JavaScript code for call after Ajax response receiving and before updating DOM on a client side.

`"data"` attribute allows to get the additional data from the server during an Ajax call. You can use JSF EL to point the property of the managed bean and its value will be serialized in JSON format and be available on the client side. You can refer to it using the `"data"` variable. For example:

```
...
    <a4j:commandButton value="Update" data="#{userBean.name}"
    oncomplete="showTheName(data.name)" />
...
```

Richfaces allows to serialize not only primitive types into JSON format, but also complex types including arrays and collections. The beans should be serializable to be referred with `"data"`.

5.4.6. Iteration components Ajax attributes

"ajaxKeys" attribute defines strings that are updated after an Ajax request. It provides possibility to update several child components separately without updating the whole page.

```
...
<a4j:poll interval="1000" action="{repeater.action}" reRender="text">
  <table>
    <tbody>
      <a4j:repeat value="{bean.props}" var="detail"
ajaxKeys="{repeater.ajaxedRowsSet}">
        <tr>
          <td>
            <h:outputText value="detail.someProperty" id="text"/>
          </td>
        </tr>
      </a4j:repeat>
    </tbody>
  </table>
</a4j:poll>
...
```

5.4.7. Other useful attributes

"status" attribute for Ajax components (such as **<a4j:commandButton>** , **<a4j:poll>** , etc.) points to an ID of **<a4j:status>** component. Use this attribute if you want to share **<a4j:status>** component between different Ajax components from different regions. The following example shows it.

```
...
<a4j:region id="extr">
  <h:form>
    <h:outputText value="Status:" />
    <a4j:status id="commonstatus" startText="In Progress...." stopText="" />
    <h:panelGrid columns="2">
      <h:outputText value="Name" />
      <h:inputText id="name" value="{userBean.name}">
        <a4j:support event="onkeyup" reRender="out" />
      </h:inputText>
      <h:outputText value="Job" />
      <a4j:region id="intr">
        <h:inputText id="job" value="{userBean.job}">
          <a4j:support event="onkeyup" reRender="out"
status="commonstatus"/>
        </h:inputText>
      </a4j:region>
    </h:panelGrid>

    <a4j:region>
      <h:outputText id="out" value="Name: {userBean.name}, Job:
#{userBean.job}" />
      <br />
      <a4j:commandButton ajaxSingle="true" value="Clean Up Form"
reRender="name, job, out" status="commonstatus">
        <a4j:actionparam name="n" value="" assignTo="{userBean.name}"
/>
    </a4j:region>
  </h:form>
</a4j:region>
```

```

        <a4j:actionparam name="j" value="" assignTo="#{userBean.job}" />
    </a4j:commandButton>
</a4j:region>
</h:form>
</a4j:region>
...

```

In the example `<a4j:support>` and `<a4j:commandButton>` are defined in different regions. Values of `"status"` attribute for these components points to an ID of `<a4j:support>`. Thus, the `<a4j:support>` component is shared between two components from different regions.

More information could be found here [<http://livedemo.exadel.com/richfaces-demo/richfaces/status.jsf?c=status>].

Other useful attribute is `"focus"`. It points to an ID of a component where focus will be set after an Ajax request.

5.5. How To...

5.5.1. Send an Ajax request

There are different ways to send Ajax requests from your JSF page. For example you can use `<a4j:commandButton>`, `<a4j:commandLink>`, `<a4j:poll>` or `<a4j:support>` tags or any other.

All these tags hide the usual JavaScript activities that are required for an XMLHttpRequest object building and an Ajax request sending. Also, they allow you to decide which components of your JSF page are to be re-rendered as a result of the Ajax response (you can list the IDs of these components in the `"reRender"` attribute).

`<a4j:commandButton>` and `<a4j:commandLink>` tags are used to send an Ajax request on "onclick" JavaScript event.

`<a4j:poll>` tag is used to send an Ajax request periodically using a timer.

The `<a4j:support>` tag allows you to add Ajax functionality to standard JSF components and send Ajax request onto a chosen JavaScript event: "onkeyup", "onmouseover", etc.

5.5.2. Decide What to Send

You may describe a region on the page you wish to send to the server, in this way you can control what part of the JSF View is decoded on the server side when you send an Ajax request.

The easiest way to describe an Ajax region on your JSF page is to do nothing, because the content between the `<f:view>` and `</f:view>` tags is considered the default Ajax region.

You may define multiple Ajax regions on the JSF page (they can even be nested) by using the `<a4j:region>` tag.

If you wish to render the content of an Ajax response outside of the active region then the value of the `"renderRegionOnly"` attribute should be set to `"false"` (`"false"` is default value). Otherwise, your Ajax updates are limited to elements of the active region.

5.5.3. Decide What to Change

Using IDs in the "reRender" attribute to define "AJAX zones" for update works fine in many cases.

But you can not use this approach if your page contains, e.g. a `<f:verbatim>` tag and you wish to update its content on an Ajax response.

The problem with the `<f:verbatim/>` tag as described above is related to the value of the transientFlag of JSF components. If the value of this flag is true, the component must not participate in state saving or restoring of process.

In order to provide a solution to this kind of problems, RichFaces uses the concept of an output panel that is defined by the `<a4j:outputPanel>` tag. If you put a `<f:verbatim>` tag inside of the output panel, then the content of the `<f:verbatim/>` tag and content of other panel's child tags could be updated on Ajax response. There are two ways to control this:

- By setting the "ajaxRendered" attribute value to "true".
- By setting the "reRender" attribute value of an Action Component to the output panel ID.

5.6. Filter Configuration

RichFaces uses a filter for a correction of code received on an Ajax request. In case of a "regular" JSF request a browser makes correction independently. In case of Ajax request in order to prevent layout destruction it's needed to use a filter, because a received code could differ from a code validated by a browser and a browser doesn't make any corrections.

An example of how to set a Filter in a web.xml file of your application is placed below.

Example:

```
...
    <filter>
        <display-name>RichFaces Filter</display-name>
        <filter-name>richfaces</filter-name>
        <filter-class>org.ajax4jsf.Filter</filter-class>
    </filter>
...
```

Note:

Fast Filter is deprecated and available only for backward compatibility with previous RichFaces versions. Fast Filter usage isn't recommended, because there is another way to use its functionality by means of Neko filter type [30].

In RichFaces 3.1 filter configuration becomes more flexible. It's possible to configure different filters for different sets of pages for the same application.

The possible filter types are:

- TIDY

"TIDY" filter type based on the Tidy parser. This filter is recommended for applications with complicated or non-standard markup when all necessary code corrections are made by the filter when a response comes from the server.

- NEKO

"NEKO" filter type corresponds to the former "Fast Filter" and it's based on the Neko parser. In case of using this filter code isn't strictly verified. Use this one if you are sure that your application markup is really strict for this filter. Otherwise it could cause lot's of errors and corrupt a layout as a result. This filter considerably accelerates all Ajax requests processing.

- NONE

No correction.

An example of configuration is placed below.

Example:

```
...
<context-param>
  <param-name>org.ajax4jsf.xmlparser.ORDER</param-name>
  <param-value>NONE,NEKO,TIDY</param-value>
</context-param>

<context-param>
  <param-name>org.ajax4jsf.xmlparser.NONE</param-name>
  <param-value>/pages/performance\.xhtml,/pages/default.*\.xhtml</param-value>
</context-param>

<context-param>
  <param-name>org.ajax4jsf.xmlparser.NEKO</param-name>
  <param-value>/pages/repeat\.xhtml</param-value>
</context-param>

<filter>
  <display-name>RichFaces Filter</display-name>
  <filter-name>richfaces</filter-name>
  <filter-class>org.ajax4jsf.Filter</filter-class>
</filter>

<filter-mapping>
  <filter-name>richfaces</filter-name>
  <servlet-name>Faces Servlet</servlet-name>
  <dispatcher>FORWARD</dispatcher>
  <dispatcher>REQUEST</dispatcher>
  <dispatcher>INCLUDE</dispatcher>
</filter-mapping>
...
```

The example shows that ORDER parameter defines the order in which particular filter types are used for pages code correction.

First of all "NONE" type is specified for the filter. Then two different sets of pages are defined for which two filter types (NONE and NEKO) are used correspondingly. If a page relates to the first set that is defined in the following way:

```
<param-value>/pages/performance\.xhtml,/pages/default.*\.xhtml</param-value> ,
```

it's not corrected, because filter type for this page is defined as "NONE". If a page is not from the first set, then "NEKO" type is set.

If a page relates to the second set that is defined in the following way:

```
<param-value>/pages/repeat\.xhtml</param-value> ,
```

then "NEKO" filter type is used for correction. If it's not related to the second set, "TIDY" type is set for the filter ("TIDY" filter type is used for code correction).

5.7. Scripts and Styles Load Strategy

Before the version 3.1.3, RichFaces loaded styles and script on demand. I.e. files are loaded only if they are required on a particular page. Since RichFaces 3.1.3, it's possible to manage how the RichFaces script and style files are loaded to application.

org.richfaces.LoadScriptStrategy

The following declaration in your web.xml allows loading the integrated script files.

```
...
<context-param>
  <param-name>org.richfaces.LoadScriptStrategy</param-name>
  <param-value>ALL</param-value>
</context-param>
...
```

If you do not declare the org.richfaces.LoadScriptStrategy in the web.xml, it equals to:

```
...
<context-param>
  <param-name>org.richfaces.LoadScriptStrategy</param-name>
  <param-value>DEFAULT</param-value>
</context-param>
...
```

The third possible value is "NONE". You have no a special reason to use it unless you obtain the newest (or modified) version of the script and want to include it manually in a page header.

Note

If you use "ALL" value of Scripts Load Strategy, make sure you turn the JavaScript files compression off:

```
...
<context-param>
  <param-name>org.ajax4jsf.COMPRESS_SCRIPT</param-name>
  <param-value>>false</param-value>
</context-param>
...
```

org.richfaces.LoadStyleStrategy

The following declaration allows to load only one integrated style sheet file.

```
...
  <context-param>
    <param-name>org.richfaces.LoadStyleStrategy</param-name>
    <param-value>ALL</param-value>
  </context-param>
...
```

The integrated style sheet contains style for all shipped components. The skinnability feature still works.

The "DEFAULT" value is a classical on-demand variant.

The "NONE" stops loading the styles at all. The earlier introduced plain skin resets all color and font parameters to null. The "NONE" value for `org.richfaces.LoadStyleStrategy` means that predefined styles for RichFaces are not used.

For more information see RichFaces User Forum [<http://www.jboss.com/index.html?module=bb&op=viewtopic&p=4114033>].

5.8. Request Errors and Session Expiration Handling

RichFaces allows to redefine standard handlers responsible for processing of different exceptional situations. It helps to define own JavaScript, which is executed when these situations occur.

5.8.1. Request Errors Handling

To execute your own code on the client in case of an error during Ajax request, it's necessary to redefine the standard "A4J.AJAX.onError" method:

```
A4J.AJAX.onError = function(req,status,message) {
    // Custom Developer Code
};
```

The function defined this way accepts as parameters:

- req - a params string of a request that calls an error
- status - the number of an error returned by the server
- message - a default message for the given error

Thus, it's possible to create your own handler that is called on timeouts, inner server errors, and etc.

5.8.2. Session Expired Handling

It's possible to redefine also the *"onExpired"* framework method that is called on the *"Session Expiration"* event.

Example:

```
A4J.AJAX.onExpired = function(loc,expiredMsg){  
    // Custom Developer Code  
};
```

Here the function receives in params:

- loc - URL of the current page (on demand can be updated)
- expiredMsg - a default message on *"Session Expiration"* event.

5.9. Skinnability

5.9.1. Why Skinnability

If you have a look at a CSS file in an enterprise application, for example, the one you're working on now, you'll see how often the same color is noted in it. Standard CSS has no way to define a particular color abstractly for defining as a panel header color, a background color of an active pop-up menu item, a separator color, etc. To define common interface styles, you have to copy the same values over and over again and the more interface elements you have the more copy-and-paste activity that needs to be performed.

Hence, if you want to change the application palette, you have to change all interrelating values, otherwise your interface can appear a bit clumsy. The chances of such an interface coming about is very high, as CSS editing usually becomes the duty of a general developer who doesn't necessarily have much knowledge of user interface design.

Moreover, if a customer wishes to have an interface look-and-feel that can be adjusted on-the-fly by an end user, your work is multiplied, as you have to deal with several CSS files variants, each of which contains the same values repeated numerous times.

These problems can be solved with the skinnability system built into the RichFaces project and realized fully in RichFaces. Every named skin has some skin-parameters for the definition of a palette and the other parameters of the user interface. By changing just a few parameters, you can alter the appearance of dozens of components in an application in a synchronized fashion without messing up user interface consistency.

The skinnability feature can't completely replace standard CSS and certainly doesn't eliminate its usage. Skinnability is a high-level extension of standard CSS, which can be used together with regular CSS declarations. You can also refer to skin parameters in CSS via JSF Expression Language. You have the complete ability to synchronize the appearance of all the elements in your pages.

5.9.2. Using Skinnability

RichFaces skinnability is designed for mixed usage with:

- Skin parameters defined in the RichFaces framework
- Predefined CSS classes for components
- User style classes

The color scheme of the component can be applied to its elements using any of three style classes:

- A default style class inserted into the framework

This contains style parameters linked to some constants from a skin. It is defined for every component and specifies a default representation level. Thus, an application interface could be modified by changing the values of skin parameters.

- A style class of skin extension

This class name is defined for every component element and inserted into the framework to allow defining a class with the same name into its CSS files. Hence, the appearance of all components that use this class is extended.

- User style class

It's possible to use one of the styleClass parameters for component elements and define your own class in it. As a result, the appearance of one particular component is changed according to a CSS style parameter specified in the class.

5.9.3. Example

Here is a simple panel component:

Example:

```
<rich:panel>
...
</rich:panel>
```

The code generates a panel component on a page, which consists of two elements: a wrapper `<div>` element and a `<div>` element for the panel body with the particular style properties. The wrapper `<div>` element looks like:

Example:

```
<div class="dr-pnl rich-panel">
...
</div>
```

dr-pnl is a CSS class specified in the framework via skin parameters:

- background-color is defined with `generalBackgroundColor`
- border-color is defined with `panelBorderColor`

It's possible to change all colors for all panels on all pages by changing these skin parameters.

However, if a **<rich-panel>** class is specified somewhere on the page, its parameters are also acquired by all panels on this page.

A developer may also change the style properties for a particular panel. The following definition:

Example:

```
<rich:panel styleClass="customClass">
...
</rich:panel>
```

could add some style properties from `customClass` to one particular panel, as a result we get three styles:

Example:

```
<div class="dr_pnl rich-panel customClass">
...
</div>
```

5.9.4. Skin Parameters Tables in RichFaces

RichFaces provides eight predefined skin parameters (skins) at the simplest level of common customization:

- DEFAULT
- plain
- emeraldTown
- blueSky
- wine
- japanCherry
- ruby
- classic
- deepMarine

To plug one in, it's necessary to specify a skin name in the *"org.richfaces.SKIN"* context-param.

Here is an example of a table with values for one of the main skins, "blueSky".

Table 5.1. Colors

| Parameter name | Default value |
|---------------------------|----------------------------|
| headerBackgroundColor | #BED6F8 |
| headerGradientColor | #F2F7FF |
| headTextColor | #000000 |
| headerWeightFont | bold |
| generalBackgroundColor | #FFFFFF |
| generalTextColor | #000000 |
| generalSizeFont | 11px |
| generalFamilyFont | Arial, Verdana, sans-serif |
| controlTextColor | #000000 |
| controlBackgroundColor | #ffffff |
| additionalBackgroundColor | #ECF4FE |
| shadowBackgroundColor | #000000 |
| shadowOpacity | 1 |
| panelBorderColor | #BED6F8 |
| subBorderColor | #ffffff |
| tabBackgroundColor | #C6DEFF |
| tabDisabledTextColor | #8DB7F3 |
| trimColor | #D6E6FB |
| tipBackgroundColor | #FAE6B0 |
| tipBorderColor | #E5973E |
| selectControlColor | #E79A00 |
| generalLinkColor | #0078D0 |
| hoverLinkColor | #0090FF |
| visitedLinkColor | #0090FF |

Table 5.2. Fonts

| Parameter name | Default value |
|------------------|----------------------------|
| headerSizeFont | 11px |
| headerFamilyFont | Arial, Verdana, sans-serif |

| Parameter name | Default value |
|-------------------------------|----------------------------|
| tabSizeFont | 11px |
| tabFamilyFont | Arial, Verdana, sans-serif |
| buttonSizeFont | 11px |
| buttonFamilyFont | Arial, Verdana, sans-serif |
| tableBackgroundColor | #FFFFFF |
| tableFooterBackgroundColor | #cccccc |
| tableSubfooterBackgroundColor | #f1f1f1 |
| tableBorderColor | #C0C0C0 |

Skin "plain" was added from 3.0.2 version. It doesn't have any parameters. It's necessary for embedding RichFaces components into existing projects which have its own styles.

To get detailed information on particular parameter possibilities, see the chapter where each component has skin parameters described corresponding to its elements.

5.9.5. Creating and Using Your Own Skin File

In order to create your own skin file, do the following:

- Create a file and define in it skin constants which are used by style classes (see section "Skin Parameters Tables in RichFaces"). The name of skin file should correspond to the following format: <name>.skin.properties. As an example of such file you can see RichFaces predefined skin parameters (skins): blueSky, classic, deepMarine, etc. These files are located in the richfaces-impl-xxxxx.jar inside the /META-INF/skins folder.
- Add a skin definition <context-param> to the web.xml of your application. An example is placed below:

Example:

```
...
<context-param>
  <param-name>org.richfaces.SKIN</param-name>
  <param-value>name</param-value>
</context-param>
...
```

- Put your <name>.skin.properties file in one of the following classpath elements: META-INF/skins/ or classpath folder (e.g. WEB-INF/classes).

5.9.6. Built-in skinnability in RichFaces

RichFaces gives an opportunity to incorporate skinnability into UI design. With this framework you can easily use named skin parameters in properties files to control the appearance of the skins that are applied consistently to a whole set of components. You can look at examples of predefined skins at:

<http://livedemo.exadel.com/richfaces-demo/>

You may simply control the look-and-feel of your application by using the skinnability service of the RichFaces framework. With the means of this service you can define the same style for rendering standard JSF components and custom JSF components built with the help of RichFaces.

To find out more on skinnability possibilities, follow these steps:

- Create a custom render kit and register it in the faces-config.xml like this:

```
<render-kit>
    <render-kit-id>NEW_SKIN</render-kit-id>
    <render-kit-class>
        org.ajax4jsf.framework.renderer.ChameleonRenderKitImpl
    </render-kit-class>
</render-kit>
```

- Then you need to create and register custom renderers for the component based on the look-and-feel predefined variables:

```
<renderer>
    <component-family>javax.faces.Command</component-family>
    <renderer-type>javax.faces.Link</renderer-type>
    <renderer-class>
        newskin.HtmlCommandLinkRenderer
    </renderer-class>
</renderer>
```

- Finally, you need to place a properties file with skin parameters into the class path root. There are two requirements for the properties file:
 - The file must be named **<skinName>**.skin.properties, in this case, it would be called newskin.skin.properties.
 - The first line in this file should be render.kit= **<render-kit-id>**, in this case, it would be called render.kit=NEW_SKIN.

Extra information on custom renderers creation can be found at:

<http://java.sun.com/javaee/javaserverfaces/reference/docs/index.html>

The RichFaces Components

The library encompasses ready-made components built based on the *Rich Faces CDK*.

6.1. <a4j:ajaxListener >

The <a4j:ajaxListener> component is the same one as *"actionListener"* or *"ValueChangeListener"*, but for an Ajax container.

Table 6.1. a4j : ajaxListener attributes

| Attribute Name | Description |
|----------------|--|
| type | Fully qualified Java class name of an AjaxListener to be created and registered. |

Table 6.2. Component identification parameters

| Name | Value |
|----------------|--|
| listener-class | org.ajax4jsf.framework.ajax.AjaxListener |
| event-class | org.ajax4jsf.framework.ajax.AjaxEvent |
| tag-class | org.ajax4jsf.taglib.html.jsp.AjaxListenerTag |

6.1.1. Creating on a page

Simple Component definition on a page:

Example:

```
...
<a4j:ajaxListener type="demo.Bean"/>
...
```

6.1.2. Dynamical creation of a component from Java code

Example:

```
package demo;

public class ImplBean implements import org.ajax4jsf.component.html.AjaxListener{
    ...
}
```

```

}

import demo.ImplBean;
...
ImplBean myListener = new ImplBean();
...

```

6.1.3. Key attributes and ways of usage

Additional to the listeners provided by JSF specification, RichFaces add one more: ajax Listener (**<a4j:ajaxListener>**). Ajax Listener is invoked before the Render Response phase. Instead of action Listener of Value Change Listener which are not invoked when Validation of Update Model phases failed, ajax Listener is guaranteed to be invoked for each Ajax response. Thus, it is a good place for update the list of re-rendered components, for example. Ajax Listener is not invoked for non-Ajax request and when RichFaces works in "Ajax Request generates Non-Ajax Response" mode. Therefore, ajax Listener invocation is a good indicator that Ajax response is going to be processed. Attribute "type" described in the following chapter. defines the fully qualified Java class name for listener. This class should implement `org.ajax4jsf.framework.ajax.ajaxListener` interface. You can access to the source of the event (Ajax component) using `event.getSource()` call.

Example:

```

...
<a4j:commandLink id="cLink" value="Click it To Send Ajax Request">
  <a4j:ajaxListener type="demo.Bean"/>
</a4j:commandLink>
...

```

Example:

```

package demo;

import org.ajax4jsf.framework.ajax.AjaxEvent;

public class Bean implements org.ajax4jsf.framework.ajax.AjaxListener{
  ...
  public void processAjax(AjaxEvent arg0){
    //Custom Developer Code
  }
  ...
}

```

6.1.5. Relevant resources links

Some additional information about usage of component can be found here. [<http://livedemo.exadel.com/richfaces-demo/richfaces/ajaxListener.jsf?c=ajaxListener>]

6.2. < a4j:keepAlive >

The **<a4j:keepAlive>** component allows to keep a state of each bean between requests.

Table 6.3. a4j : keepAlive attributes

| Attribute Name | Description |
|----------------|---|
| ajaxOnly | if true, bean value restored in ajax requests only. |
| beanName | name of bean for EL-expressions. |

Table 6.4. Component identification parameters

| Name | Value |
|------------------|---------------------------------------|
| component-type | org.ajax4jsf.components.KeepAlive |
| component-family | org.ajax4jsf.components.AjaxKeepAlive |
| component-class | org.ajax4jsf.components.AjaxKeepAlive |

6.2.1. Creating on a page

Simple Component definition on a page:

Example:

```
<a4j:keepAlive beanName = "#{myClass.testBean}" />
```

6.2.2. Dynamical creation of a component from Java code

Example:

```
import org.ajax4jsf.ajax.AjaxKeepAlive;
...
AjaxKeepAlive myKeepAlive = new AjaxKeepAlive();
...
```

6.2.3. Key attributes and ways of usage

If a managed bean is declared with 'request' scope in the configuration file with the help of 'managed-bean-scope' tag then the life-time of this bean instance is valid only for the current request. Any attempts to make a reference to the bean instance after the request end will throw in Illegal Argument Exception by the server. To avoid these kinds of Exception, component **<a4j:keepAlive>** is used to maintain the state of the whole bean object among subsequent request.

Example:

```
<a4j:keepAlive beanName = "#{myClass.testBean}" />
```

Note that the attribute 'beanName' must point to a legal jsf EL expression which resolves to a managed bean instance. For example for the above code the class definition may look like this:

```

class MyClass{
...
private TestBean testBean;
// Getters and Setters for testBean.
...
}

```

6.2.4. Relevant resources links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/keepAlive.jsf?c=keepAlive>] you can see the example of `<a4j:keepAlive>` usage and sources for the given example.

Some additional information about usage of component can be found here. [<http://jboss.com/index.html?module=bb&op=viewtopic&t=104989>]

6.3. `<a4j:actionparam>`

The `<a4j:actionParam>` component combines the functionality of both JSF components: `<f:param>` and `<f:actionListener>`.

More information about `<f:param>` and `<f:actionListener>` can be found here [<http://java.sun.com/javasee/javaxserverfaces/1.2/docs/tlddocs/index.html>].

Table 6.5. a4j : actionparam attributes

| Attribute Name | Description |
|----------------|---|
| assignTo | EL expression for updatable bean property. This property will be updated if the parent command component performs an actionEvent. |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| converter | ID of a converter to be used or a reference to a converter. |
| id | Every component may have a unique id that is automatically created if omitted |
| name | A name of this parameter |
| noEscape | If set to true, the value will not enclosed within single quotes and there will be no escaping of characters. This allows the use of the value as JavaScript code for calculating value on the client-side. This doesn't work with non-AJAX components. |
| value | An initial value or a value binding |

Table 6.6. Component identification parameters

| Name | Value |
|-----------------|---|
| component-type | org.ajax4jsf.ActionParameter |
| component-class | org.ajax4jsf.component.html.HtmlActionParameter |

6.3.1. Creating on a page

Simple component definition example:

Example:

```
<a4j:actionParam noEscape="true" name="param1" value="getMyValue()"
  assignTo="#{bean.prop1}" />
```

6.3.2. Dynamical creation of a component from Java code

Example:

```
import org.ajax4jsf.component.html.HtmlActionParameter;
...
HtmlActionParameter myActionParameter = new HtmlActionParameter();
...
```

6.3.3. Key attributes and ways of usage

The component **<a4j:actionParam>** is a combination of the functionality of two JSF tags: **<f:param>** and **<f:actionListener>**.

At the render phase, it's decoded by parent component (**<h:commandLink>** or like) as usual. At the process request phase, if the parent component performs an action event, update the value specified in the "assignTo" attribute as its value. If a converter attribute is specified, use it to encode and decode the value to a string stored in the html parameter.

<a4j:actionParam> has a *"noEscape"* attribute. If it is set to "true", the value will be evaluated as a JavaScript code.

Example:

```
...
    <script>
        ...
        var foo = "bar";
        ...
    </script>
    ...
    <a4j:actionParam noEscape="true" name="param1" value="foo"
  assignTo="#{bean.prop1}" />
...
```

The **<a4j:param>** extends **<f:param>**, so the "name" attribute is mandatory. Otherwise, the value will be missing due missing the request parameter name for it.

6.3.4. Relevant resources links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/actionparam.jsf?c=actionparam>] you can see the example of `<a4j:actionParam>` usage and sources for the given example.

More information can be found on the Ajax4jsf Users Forum [<http://www.jboss.com/index.html?module=bb&op=viewtopic&p=4063764>].

6.4. <a4j:commandButton >

6.4.1. Description

The `<a4j:commandButton>` component is very similar to the `<h:commandButton>` component, the only difference is that an Ajax form submit is generated on a click and it allows dynamic rerendering after a response comes back. It's not necessary to plug any support into the component, as Ajax support is already built in.

Table 6.7. a4j : commandButton attributes

| Attribute Name | Description |
|------------------|---|
| accesskey | This attribute assigns an access key to an element. An access key is a single character from the document character set. Note: Authors should consider the input method of the expected reader when specifying an accesskey |
| action | MethodBinding pointing at the application action to be invoked, if this UIComponent is activated by the user, during the Apply Request Values or Invoke Application phase of the request processing lifecycle, depending on the value of the immediate property |
| actionExpression | The action method binding expression. |
| actionListener | MethodBinding pointing at method accepting an ActionEvent with return type void |
| ajaxSingle | if "true", submits ONLY one field/link, instead of all form controls |
| alt | Alternate textual description of the element rendered by this component. |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| bypassUpdates | If "true", after process validations phase it skips updates of model beans on a force render response. It can be used for validating components input |

| Attribute Name | Description |
|--------------------|---|
| data | Serialized (on default with JSON) data passed on the client by a developer on AJAX request. It's accessible via "data.foo" syntax |
| dir | Direction indication for text that does not inherit directionality. Valid values are "LTR" (left-to-right) and "RTL" (right-to-left) |
| disabled | When set for a form control, this boolean attribute disables the control for user input |
| eventsQueue | Name of requests queue to avoid send next request before complete other from same event. Can be used to reduce number of requests of frequently events (key press, mouse move etc.) |
| focus | id of element to set focus after request completed on client side |
| id | Every component may have a unique id that is automatically created if omitted |
| ignoreDupResponses | Attribute allows to ignore an Ajax Response produced by a request if the newest 'similar' request is in a queue already. ignoreDupResponses="true" does not cancel the request while it is processed on the server, but just allows to avoid unnecessary updates on the client side if the response isn't actual now. |
| image | Absolute or relative URL of the image to be displayed for this button. If specified, this "input" element will be of type "image". Otherwise, it will be of the type specified by the "type" property with a label specified by the "value" property. |
| immediate | True means, that the default ActionListener should be executed immediately (i.e. during Apply Request Values phase of the request processing lifecycle), rather than waiting until the Invoke Application phase |
| lang | Code describing the language used in the generated markup for this component |
| limitToList | If "true", updates on client side ONLY elements from this 'reRender' property. If "false" (default) updates all rendered by ajax region components |
| onbeforedomupdate | JavaScript code for call before DOM has been updated on client side |

| Attribute Name | Description |
|----------------|---|
| onblur | HTML: script expression; the element lost the focus |
| onchange | HTML: script expression; the element value was changed |
| onclick | HTML: a script expression; a pointer button is clicked |
| oncomplete | JavaScript code for call after request completed on client side |
| ondblclick | HTML: a script expression; a pointer button is double-clicked |
| onfocus | HTML: script expression; the element got the focus |
| onkeydown | HTML: a script expression; a key is pressed down |
| onkeypress | HTML: a script expression; a key is pressed and released |
| onkeyup | HTML: a script expression; a key is released |
| onmousedown | HTML: script expression; a pointer button is pressed down |
| onmousemove | HTML: a script expression; a pointer is moved within |
| onmouseout | HTML: a script expression; a pointer is moved away |
| onmouseover | HTML: a script expression; a pointer is moved onto |
| onmouseup | HTML: script expression; a pointer button is released |
| rendered | If "false", this component is not rendered |
| requestDelay | Attribute defines the time (in ms.) that the request will be wait in the queue before it is ready to send. When the delay time is over, the request will be sent to the server or removed if the newest 'similar' request is in a queue already |
| reRender | Id[s] (in format of call <code>UIComponent.findComponent()</code> of components, rendered in case of <code>AjaxRequest</code> caused by this component. Can be single id, comma-separated list of Id's, or EL Expression with array or Collection |
| size | This attribute tells the user agent the initial width of the control. The width is given in pixels except when type attribute has the value "text" or "password". In |

| Attribute Name | Description |
|----------------|--|
| | that case, its value refers to the (integer) number of characters |
| status | ID (in format of call <code>UIComponent.findComponent()</code>) of Request status component |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| tabindex | This attribute specifies the position of the current element in the tabbing order for the current document. This value must be a number between 0 and 32767. User agents should ignore leading zeros |
| timeout | Timeout (in ms) for request. |
| title | Advisory title information about markup elements generated for this component |
| type | submit reset image button This attribute specifies a type of control to create. The default value for this attribute is "submit" |
| value | The current value for this component |

Table 6.8. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | org.ajax4jsf.CommandButton |
| component-family | javax.faces.Command |
| component-class | org.ajax4jsf.component.html.HtmlAjaxCommandButton |
| renderer-type | org.ajax4jsf.components.AjaxCommandButtonRenderer |

6.4.2. Creating on a page

`<a4j:commandButton>` is used in the same way as `<h:commandButton>`, but with definition of the area that is updated after the response comes back from the server.

```
<a4j:commandButton reRender="someData" action="#{bean.action1}" value="Link" />
```

This definition of the component provides a link, a click on the link causes an Ajax form submit on the server, "action1" method performance, and rendering of the component with `"someData"` id after the response comes back from the server.

6.4.3. Dynamical creation of a component from Java code

Example:

```
import org.ajax4jsf.component.html.HtmlAjaxCommandButton;
...
HtmlAjaxCommandButton myButton = new HtmlAjaxCommandButton();
...
```

6.4.4. Key attributes and ways of usage

The component `<a4j:commandButton>` placed on a page generates the following HTML code:

```
<input type="submit" onclick="A4J.AJAX.Submit(...request parameters);return false;"
value="sort" />
```

Hence, the utility method "A4J.AJAX.Submit" is called on a click, the method performs Ajax request as the `<a4j:support>` component

Note:

AJAX support is built in and it's not necessary to add nested `<a4j:support>` to the component.

The usage of the keyword 'this' in JavaScript code in the *"oncomplete"* attribute depends on the location of `<a4j:commandButton>`. If the commandButton is situated outside the re-rendered region you can use keyword 'this' as in the following example:

```
...
<h:form id="form">
  <a4j:commandButton id="cbutton" action="director.rollCamera"
    onclick="this.disabled=true"
    oncomplete="this.disabled=false" />
</h:form>
...
```

Otherwise if the commandButton contained in re-rendered region the *"oncomplete"* attribute has a problem obtaining a reference of the commandButton object when using the keyword 'this'. In this case you can use the *"oncomplete"* attribute as in the following example:

```
...
<h:form id="form">
<a4j:commandButton id="cbutton" action="director.rollCamera"
  onclick="this.disabled=true"
  oncomplete="document.getElementById('form:cbutton').disabled=false" />
</h:form>
...
```

Common JSF navigation could be performed after an Ajax submit and partial rendering, but Navigation Case must be defined as `<redirect/>` in order to avoid problems with some browsers.

As any Core Ajax component sending Ajax requests and processing server responses `<a4j:commandButton>` has all attributes described above (see `<a4j:support>` chapter) that provide the required behavior of requests sending (delay, limitation of submit area and rendering, and etc.)

6.4.5. Relevant resources links

Here [\[http://livedemo.exadel.com/richfaces-demo/richfaces/commandButton.jsf?c=commandButton\]](http://livedemo.exadel.com/richfaces-demo/richfaces/commandButton.jsf?c=commandButton) you can see the example of `<a4j:commandButton>` usage and sources for the given example.

6.5. `<a4j:commandLink>`

6.5.1. Description

The `<a4j:commandLink>` component is very similar to the `<h:commandLink>` component, the only difference is that an Ajax form submit is generated on a click and it allows dynamic rerendering after a response comes back. It's not necessary to plug any support into the component, as Ajax support is already built in.

Table 6.9. a4j : commandLink attributes

| Attribute Name | Description |
|------------------|---|
| accesskey | This attribute assigns an access key to an element. An access key is a single character from the document character set. Note: Authors should consider the input method of the expected reader when specifying an accesskey |
| action | MethodBinding pointing at the application action to be invoked, if this UIComponent is activated by the user, during the Apply Request Values or Invoke Application phase of the request processing lifecycle, depending on the value of the immediate property |
| actionExpression | The action method binding expression. |
| actionListener | MethodBinding pointing at method accepting an ActionEvent with return type void |
| ajaxSingle | if "true", submits ONLY one field/link, instead of all form controls |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| bypassUpdates | If "true", after process validations phase it skips updates of model beans on a force render response. It can be used for validating components input |
| charset | |

| Attribute Name | Description |
|--------------------|---|
| | The character encoding of a resource designated by this hyperlink |
| coords | This attribute specifies the position and shape on the screen. The number and order of values depends on the shape being defined. Possible combinations: * rect: left-x, top-y, right-x, bottom-y. * circle: center-x, center-y, radius. Note. When the radius value is percentage value, user agents should calculate the final radius value based on the associated object's width and height. The radius should be the smaller value of the two. * poly: x1, y1, x2, y2, ..., xN, yN. The first x and y coordinate pair and the last should be the same to close the polygon. When these coordinate values are not the same, user agents should infer an additional coordinate pair to close the polygon. Coordinates are relative to the top, left corner of the object. All values are lengths. All values are separated by commas |
| data | Serialized (on default with JSON) data passed on the client by a developer on AJAX request. It's accessible via "data.foo" syntax |
| dir | Direction indication for text that does not inherit directionality. Valid values are "LTR" (left-to-right) and "RTL" (right-to-left) |
| eventsQueue | Name of requests queue to avoid send next request before complete other from same event. Can be used to reduce number of requests of frequently events (key press, mouse move etc.) |
| focus | id of element to set focus after request completed on client side |
| hreflang | Base language of a resource specified with the href attribute; hreflang may only be used with href |
| id | Every component may have a unique id that is automatically created if omitted |
| ignoreDupResponses | Attribute allows to ignore an Ajax Response produced by a request if the newest 'similar' request is in a queue already. ignoreDupResponses="true" does not cancel the request while it is processed on the server, but just allows to avoid unnecessary updates on the client side if the response isn't actual now |

| Attribute Name | Description |
|-------------------|---|
| immediate | True means, that the default ActionListener should be executed immediately (i.e. during Apply Request Values phase of the request processing lifecycle), rather than waiting until the Invoke Application phase |
| lang | Code describing the language used in the generated markup for this component |
| limitToList | If "true", updates on client side ONLY elements from this 'reRender' property. If "false" (default) updates all rendered by ajax region components |
| onbeforedomupdate | JavaScript code for call before DOM has been updated on client side |
| onblur | JavaScript code. The onblur event occurs when an element loses focus either by the pointing device or by tabbing navigation. It may be used with the same elements as onfocus |
| onclick | HTML: a script expression; a pointer button is clicked |
| oncomplete | JavaScript code for call after request completed on client side |
| ondblclick | HTML: a script expression; a pointer button is double-clicked |
| onfocus | JavaScript code. The onfocus event occurs when an element gets focus |
| onkeydown | HTML: a script expression; a key is pressed down |
| onkeypress | HTML: a script expression; a key is pressed and released |
| onkeyup | HTML: a script expression; a key is released |
| onmousedown | HTML: script expression; a pointer button is pressed down |
| onmousemove | HTML: a script expression; a pointer is moved within |
| onmouseout | HTML: a script expression; a pointer is moved away |
| onmouseover | HTML: a script expression; a pointer is moved onto |
| onmouseup | HTML: script expression; a pointer button is released |

| Attribute Name | Description |
|----------------|--|
| rel | The relationship from the current document to the anchor specified by this hyperlink. The value of this attribute is a space-separated list of link types |
| rendered | If "false", this component is not rendered |
| requestDelay | Attribute defines the time (in ms.) that the request will be wait in the queue before it is ready to send. When the delay time is over, the request will be sent to the server or removed if the newest 'similar' request is in a queue already |
| reRender | Id[s] (in format of call <code>UIComponent.findComponent()</code>) of components, rendered in case of AjaxRequest caused by this component. Can be single id, comma-separated list of Id's, or EL Expression with array or Collection |
| rev | A reverse link from the anchor specified by this hyperlink to the current document. The value of this attribute is a space-separated list of link types |
| shape | default rect circle poly [CI] This attribute specifies the shape of a region. Possible values: * default: Specifies the entire region. * rect: Define a rectangular region. * circle: Define a circular region. * poly: Define a polygonal region. |
| status | ID (in format of call <code>UIComponent.findComponent()</code>) of Request status component |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| tabindex | This attribute specifies the position of the current element in the tabbing order for the current document. This value must be a number between 0 and 32767. User agents should ignore leading zeros |
| target | This attribute specifies the name of a frame where a document is to be opened. By assigning a name to a frame via the name attribute, authors can refer to it as the "target" of links defined by other elements |
| timeout | Timeout (in ms) for request. |

| Attribute Name | Description |
|----------------|---|
| title | Advisory title information about markup elements generated for this component |
| type | The content type of the resource designated by this hyperlink |
| value | The current value for this component |

Table 6.10. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | org.ajax4jsf.CommandLink |
| component-family | javax.faces.Command |
| component-class | org.ajax4jsf.component.html.HtmlAjaxCommandLink |
| renderer-type | org.ajax4jsf.components.AjaxCommandLinkRenderer |

6.5.2. Creating on a page

`<a4j:commandLink>` is used in the same way as `<h:commandLink>`, but with definition of the area that is updated after the response comes back from the server.

```
<a4j:commandLink reRender="someData" action="#{bean.action1}" value="Link"/>
```

This definition of the component provides a link, and a click on the link causes an Ajax form submit on the server, "action1" method performance, and rendering of the component with *"someData"* id after the response comes back from the server.

6.5.3. Dynamical creation of a component from Java code

Example:

```
import org.ajax4jsf.component.html.HtmlAjaxCommandLink;
...
HtmlAjaxCommandLink myLink = new HtmlAjaxCommandLink();
...
```

6.5.4. Key attributes and ways of usage

The component `<a4j:commandLink>` placed on a page generates the following HTML code:

```
<a href="#" onclick="A4J.AJAX.Submit(?request parameters);
return
<a href="#" onclick="A4J.AJAX.Submit(?request parameters);
return false;">
<span style="color: black;">Link Value</span>
</a>
```

Hence, the utility method "A4J.AJAX.Submit" is called on a click, the method performs Ajax request as the **<a4j:support>** component

Note:

AJAX support is built in and it's not necessary to add nested **<a4j:support>** to the component.

Common JSF navigation could be performed after Ajax submit and partial rendering, but Navigation Case must be defined as **<redirect/>** in order to avoid problems with some browsers.

As any Core Ajax component sending Ajax requests and processing server responses **<a4j:commandLink>** has all attributes described above (see **<a4j:support>** chapter) that provide the required behavior of requests sending (delay, limitation of submit area and rendering, etc.)

6.5.5. Relevant resources links

Here [http://livedemo.exadel.com/richfaces-demo/richfaces/commandLink.jsf?c=commandLink] you can see the example of **<a4j:commandLink>** usage and sources for the given example

6.6. <a4j:form >

6.6.1. Description

The **<a4j:form>** component is very similar to the same component from the JSF HTML library, the only slight difference is in generation of links inside and possibility of Ajax by-default submission.

Table 6.11. a4j : form attributes

| Attribute Name | Description |
|----------------|--|
| accept | This attribute specifies a comma-separated list of content types that a server processing this form will handle correctly. User agents may use this information to filter out non-conforming files when prompting a user to select files to be sent to the server (cf. the INPUT element when type="file") |
| acceptCharset | This attribute specifies the list of character encodings for input data that is accepted by the server processing this form. The value is a space- and/or comma-delimited list of charset values. The client must interpret this list as an exclusive-or list, i.e., the server is able to accept any single character encoding per entity received. The default value for this attribute is the reserved string "UNKNOWN". User agents may interpret this value as the character encoding that was used to transmit the document containing this FORM element |
| ajaxSingle | |

| Attribute Name | Description |
|--------------------|--|
| | if "true", submits ONLY one field/link, instead of all form controls |
| ajaxSubmit | If true, it becomes possible to set AJAX submission way for any components inside . |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| bypassUpdates | If "true", after process validations phase it skips updates of model beans on a force render response. It can be used for validating components input |
| data | Serialized (on default with JSON) data passed on the client by a developer on AJAX request. It's accessible via "data.foo" syntax |
| enctype | This attribute specifies the content type used to submit the form to the server (when the value of method is "post"). The default value for this attribute is "application/x-www-form-urlencoded". The value "multipart/form-data" should be used in combination with the INPUT element, type="file" |
| eventsQueue | Name of requests queue to avoid send next request before complete other from same event. Can be used to reduce number of requests of frequently events (key press, mouse move etc.) |
| focus | id of element to set focus after request completed on client side |
| id | Every component may have a unique id that is automatically created if omitted |
| ignoreDupResponses | Attribute allows to ignore an Ajax Response produced by a request if the newest 'similar' request is in a queue already. ignoreDupResponses="true" does not cancel the request while it is processed on the server, but just allows to avoid unnecessary updates on the client side if the response isn't actual now |
| limitToList | If "true", updates on client side ONLY elements from this 'reRender' property. If "false" (default) updates all rendered by ajax region components |
| onbeforedomupdate | JavaScript code for call before DOM has been updated on client side |

| Attribute Name | Description |
|----------------|---|
| oncomplete | JavaScript code for call after request completed on client side |
| onreset | The onreset event occurs when a form is reset. It only applies to the FORM element |
| onsubmit | The onsubmit event occurs when a form is submitted. It only applies to the FORM element |
| prependId | The flag indicating whether or not this form should prepend its id to its descendent id during the clientId generation process. If this flag is not set, the default value is true. |
| rendered | If "false", this component is not rendered |
| requestDelay | Attribute defines the time (in ms.) that the request will be wait in the queue before it is ready to send. When the delay time is over, the request will be sent to the server or removed if the newest 'similar' request is in a queue already |
| reRender | Id[s] (in format of call <code>UIComponent.findComponent()</code>) of components, rendered in case of <code>AjaxRequest</code> caused by this component. Can be single id, comma-separated list of Id's, or EL Expression with array or Collection |
| status | ID (in format of call <code>UIComponent.findComponent()</code>) of Request status component |
| target | This attribute specifies the name of a frame where a document is to be opened. By assigning a name to a frame via the name attribute, authors can refer to it as the "target" of links defined by other elements |
| timeout | Timeout (in ms) for request. |

Table 6.12. Component identification parameters

| Name | Value |
|------------------|--------------------------------------|
| component-type | org.ajax4jsf.Form |
| component-family | javax.faces.Form |
| component-class | org.ajax4jsf.component.html.AjaxForm |
| renderer-type | org.ajax4jsf.FormRenderer |

6.6.2. Creating on a page

Component definition on a page is similar to definition of the original component from JSF HTML library.

```
<a4j:form>
  <!--...Some content to be submitted.-->
</a4j:form>
```

6.6.3. Dynamical creation of a component from Java code

Example:

```
import org.ajax4jsf.component.html.AjaxForm;
...
AjaxForm myForm = new AjaxForm();
...
```

6.6.4. Key attributes and ways of usage

The difference with the original component is that all hidden fields required for command links are always rendered and it doesn't depend on links rendering on the initial page. It solves the problem with invalid links that weren't rendered on a page immediately, but after some Ajax request.

Beginning with release 1.0.5 additional attributes that make this form variant universal have appeared.

If *"ajaxSubmit"* attribute is true, it becomes possible to set Ajax submission way for any components inside, i.e. not a page URL is used as an *"action"* attribute, but the `javascript:A4J.AJAX.Submit(...)` call. In this case, the *"reRender"* attribute contains a list of Ids of components defined for re-rendering. If you have **<h:commandButton>** or **<h:commandLink>** inside the form, they work as **<a4j:commandButton>** .

Example:

```
<a4j:form id="helloForm" ajaxSubmit="true" reRender="table">
  ...
  <t:dataTable id="table"... >
    ...
  </t:dataTable>
  ...
  <t:datascroller for="table"... >
    ...
  </t:datascroller>
  ...
</a4j:form>
```

This example shows that in order to make **<t:datascroller>** submissions to be Ajax ones it's required only to place this **<t:datascroller>** into **<a4j:form>** . In the other case it is necessary to redefine renders for its child links elements that are defined as **<h:commandLink>** and can't be made Ajax ones with using e.g. **<a4j:support>** .

With the help of *"limitToList"* attribute you can limit areas, which are updated after the responses. If *"limitToList"* is true, only the reRender attribute is taken in account. Therefore, if you use blocks of text wrapped with **<a4j:outputPanel>** and *"ajaxRendered"* = true, blocks of text are ignored.

6.6.5. Relevant resources links

Here, [<http://livedemo.exadel.com/richfaces-demo/richfaces/form.jsf?c=form>] you can see the example of `<a4j:form>` usage and sources for the given example.

6.7. < a4j:htmlCommandLink >

6.7.1. Description

The `<a4j:htmlCommandLink>` component is very similar to the same component from the JSF HTML library, the only slight difference is in links generation and problem solving that occurs when an original component is used.

Table 6.13. a4j : htmlCommandLink attributes

| Attribute Name | Description |
|------------------|---|
| accesskey | This attribute assigns an access key to an element. An access key is a single character from the document character set. Note: Authors should consider the input method of the expected reader when specifying an accesskey |
| action | MethodBinding pointing at the application action to be invoked, if this UIComponent is activated by the user, during the Apply Request Values or Invoke Application phase of the request processing lifecycle, depending on the value of the immediate property |
| actionExpression | The action method binding expression. |
| actionListener | MethodBinding pointing at method accepting an ActionEvent with return type void |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| charset | The character encoding of a resource designated by this hyperlink |
| coords | This attribute specifies the position and shape on the screen. The number and order of values depends on the shape being defined. Possible combinations: * rect: left-x, top-y, right-x, bottom-y. * circle: center-x, center-y, radius. Note. When the radius value is percentage value, user agents should calculate the final radius value based on the associated object's width and height. The radius should be the smaller value of the two. * poly: x1, y1, x2, y2, ..., xN, yN. The first x and y coordinate pair and the last should be the same to close |

| Attribute Name | Description |
|----------------|---|
| | the polygon. When these coordinate values are not the same, user agents should infer an additional coordinate pair to close the polygon. Coordinates are relative to the top, left corner of the object. All values are lengths. All values are separated by commas |
| dir | Direction indication for text that does not inherit directionality. Valid values are "LTR" (left-to-right) and "RTL" (right-to-left) |
| disabled | When set for a form control, this boolean attribute disables the control for user input. |
| hreflang | Base language of a resource specified with the href attribute; hreflang may only be used with href |
| id | Every component may have a unique id that is automatically created if omitted |
| immediate | True means, that the default ActionListener should be executed immediately (i.e. during Apply Request Values phase of the request processing lifecycle), rather than waiting until the Invoke Application phase |
| lang | Code describing the language used in the generated markup for this component |
| onblur | JavaScript code. The onblur event occurs when an element loses focus either by the pointing device or by tabbing navigation. It may be used with the same elements as onfocus |
| onclick | HTML: a script expression; a pointer button is clicked |
| ondblclick | HTML: a script expression; a pointer button is double-clicked |
| onfocus | JavaScript code. The onfocus event occurs when an element gets focus |
| onkeydown | HTML: a script expression; a key is pressed down |
| onkeypress | HTML: a script expression; a key is pressed and released |
| onkeyup | HTML: a script expression; a key is released |
| onmousedown | HTML: script expression; a pointer button is pressed down |
| onmousemove | HTML: a script expression; a pointer is moved within |

| Attribute Name | Description |
|----------------|--|
| onmouseout | HTML: a script expression; a pointer is moved away |
| onmouseover | HTML: a script expression; a pointer is moved onto |
| onmouseup | HTML: script expression; a pointer button is released |
| rel | The relationship from the current document to the anchor specified by this hyperlink. The value of this attribute is a space-separated list of link types |
| rendered | If "false", this component is not rendered |
| rev | A reverse link from the anchor specified by this hyperlink to the current document. The value of this attribute is a space-separated list of link types |
| shape | default rect circle poly [CI] This attribute specifies the shape of a region. Possible values: * default: Specifies the entire region. * rect: Define a rectangular region. * circle: Define a circular region. * poly: Define a polygonal region. |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| tabindex | This attribute specifies the position of the current element in the tabbing order for the current document. This value must be a number between 0 and 32767. User agents should ignore leading zeros |
| target | This attribute specifies the name of a frame where a document is to be opened. By assigning a name to a frame via the name attribute, authors can refer to it as the "target" of links defined by other elements |
| title | Advisory title information about markup elements generated for this component |
| type | The content type of the resource designated by this hyperlink |
| value | The current value for this component |

Table 6.14. Component identification parameters

| Name | Value |
|----------------|-----------------------------|
| component-type | javax.faces.HtmlCommandLink |

| Name | Value |
|------------------|--|
| component-family | javax.faces.Command |
| component-class | javax.faces.component.html.HtmlCommandLink |
| renderer-type | org.ajax4jsf.HtmlCommandLinkRenderer |

6.7.2. Creating on a page

Component definition on a page is the same as for the original component from the JSF HTML library.

Example:

```
<a4j:htmlCommandLink value="value" action="action" />
```

6.7.3. Dynamical creation of a component from Java code

Example:

```
import javax.faces.component.html.HtmlCommandLink;
...
HtmlCommandLink myCommandLink = new HtmlCommandLink();
...
```

6.7.4. Key attributes and ways of usage

The difference with the original component is that all hidden fields required for command links with the child **<f:param>** elements are always rendered and it doesn't depend on links rendering on the initial page. It solves the problem with invalid links that weren't rendered on a page immediately, but after some Ajax request.

Example:

```
<a4j:form>
...
<a4j:htmlComandLink action="action" value="link" rendered="#{bean.rendered}">
  <f:param ...>
<a4j:htmlComandLink>
...
</a4j:form>
```

In this example **<a4j:htmlCommandLink>** works as standard **<h:commandLink>**, but here hidden fields required for correct functionality are rendered before the first downloading of a page, though it doesn't happen if its attribute isn't set to "false".

6.7.5. Relevant resources links

Here [\[http://livedemo.exadel.com/richfaces-demo/richfaces/htmlCommandLink.jsf?c=htmlCommandLink\]](http://livedemo.exadel.com/richfaces-demo/richfaces/htmlCommandLink.jsf?c=htmlCommandLink) you can see the example of **<a4j:htmlCommandLinks>** usage and sources for the given example.

6.8. <a4j:jsFunction >

6.8.1. Description

The <a4j:jsFunction> component allows to invoke the server side data and return it in a JSON format to use in a client JavaScript calls.

Table 6.15. a4j : jsFunction attributes

| Attribute Name | Description |
|--------------------|---|
| action | MethodBinding pointing at the application action to be invoked, if this UIComponent is activated by the user, during the Apply Request Values or Invoke Application phase of the request processing lifecycle, depending on the value of the immediate property |
| actionExpression | The action method binding expression. |
| actionListener | MethodBinding pointing at method accepting an ActionEvent with return type void |
| ajaxSingle | if "true", submits ONLY one field/link, instead of all form controls |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| bypassUpdates | If "true", after process validations phase it skips updates of model beans on a force render response. It can be used for validating components input |
| data | Serialized (on default with JSON) data passed on the client by a developer on AJAX request. It's accessible via "data.foo" syntax |
| eventsQueue | Name of requests queue to avoid send next request before complete other from same event. Can be used to reduce number of requests of frequently events (key press, mouse move etc.) |
| focus | id of element to set focus after request completed on client side |
| id | Every component may have a unique id that is automatically created if omitted |
| ignoreDupResponses | Attribute allows to ignore an Ajax Response produced by a request if the newest 'similar' request is in a queue already. ignoreDupResponses="true" does not cancel the request while it is processed on the server, but just |

| Attribute Name | Description |
|-------------------|---|
| | allows to avoid unnecessary updates on the client side if the response isn't actual now |
| immediate | True means, that the default ActionListener should be executed immediately (i.e. during Apply Request Values phase of the request processing lifecycle), rather than waiting until the Invoke Application phase |
| limitToList | If "true", updates on client side ONLY elements from this 'reRender' property. If "false" (default) updates all rendered by ajax region components |
| name | Name of generated JavaScript function definition |
| onbeforedomupdate | JavaScript code for call before DOM has been updated on client side |
| oncomplete | JavaScript code for call after request completed on client side |
| rendered | If "false", this component is not rendered |
| requestDelay | Attribute defines the time (in ms.) that the request will be wait in the queue before it is ready to send. When the delay time is over, the request will be sent to the server or removed if the newest 'similar' request is in a queue already |
| reRender | Id[s] (in format of call <code>UIComponent.findComponent()</code>) of components, rendered in case of AjaxRequest caused by this component. Can be single id, comma-separated list of Id's, or EL Expression with array or Collection |
| status | ID (in format of call <code>UIComponent.findComponent()</code>) of Request status component |
| timeout | Response waiting time on a particular request. If a response is not received during this time, the request is aborted |

Table 6.16. Component identification parameters

| Name | Value |
|------------------|--|
| component-type | org.ajax4jsf.Function |
| component-family | org.ajax4jsf.components.ajaxFunction |
| component-class | org.ajax4jsf.component.html.HtmlajaxFunction |

| Name | Value |
|---------------|--|
| renderer-type | org.ajax4jsf.components.ajaxFunctionRenderer |

6.8.2. Creating on a page

Simple component definition example:

Example:

```
...
<head>
  <script>
    <!--There is some script named "myScript" that uses parameters which will be taken
    from server-->
  </script>
</head>
<body>
  ...
  <a4j:jsFunction data="#{bean.someProperty}" name="callScript"
    oncomplete="myScript(data.subProperty1, data.subProperty2)"/>
  ...

```

The script *"myScript"* will be called after bean.someProperty data will be returned from server(e.g. It'll be object with two subproperties).

6.8.3. Dynamical creation of a component from Java code

Example:

```
import org.ajax4jsf.component.html.HtmlajaxFunction;
...
HtmlajaxFunction myFunction = new HtmlajaxFunction();
...

```

6.8.4. Key attributes and ways of usage

As the component uses Ajax request to get data from server - it has all common Ajax Action attributes. Hence, "action" and "actionListener" can be invoked, and reRendering some parts of the page fired after calling function.

When using the **<a4j:jsFunction>** it's possible to initiate the Ajax request from the JavaScript and perform partial update of a page and/or invoke the JavaScript function with data returned by Ajax response.

```
...
<body onload="callScript()">
  ...
  <h:form>
  ...
  <a4j:jsFunction name="callScript" data="#{bean.someProperty1 }

```

```

        "reRender="someComponent" onComplete="myScript(data.subProperty1,
data.subProperty2)">
        <a4j:actionParam name="param_name" assignTo="#{bean.someProperty2}">
        </a4j:actionParam>
    </a4j:jsFunction>
    ...
</h:form>
    ...
</body>
    ...

```

The **<a4j:jsFunction>** allows to use **<a4j:actionParam>** or pure **<f:param>** for passing any number of parameters of the JavaScript function into Ajax request. **<a4j:jsFunction>** is similar to **<a4j:commandButton>**, but it could be activated from the JavaScript code. It allows to invoke some server side functionality and use the returned data in the JavaScript function invoked from *"oncomplete"* attribute. Hence it's possible to use **<a4j:jsFunction>** instead of **<a4j:commandButton>**. You can put it anywhere, just don't forget to use **<h:form> ... </h:form>** around it.

6.8.5. Relevant resources links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/jsFunction.jsf?c=jsFunction>] you can see the example of **<a4j:jsFunction>** usage and sources for the given example.

6.9. <a4j:include >

6.9.1. Description

The **<a4j:include>** component is used for page areas update after an Ajax request according to the faces-config Navigation Rules and for implementation of wizard-like parts work in Ajax mode.

Table 6.17. a4j : include attributes

| Attribute Name | Description |
|----------------|---|
| ajaxRendered | Defines, whether the content of this component must be (or not) included in AJAX response created by parent AJAX Container, even if it is not forced by reRender list of ajax action. Ignored if component marked to output by Ajax action. default false |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| dir | Direction indication for text that does not inherit directionality. Valid values are "LTR" (left-to-right) and "RTL" (right-to-left) |
| id | Every component may have a unique id that is automatically created if omitted |
| keepTransient | |

| Attribute Name | Description |
|----------------|--|
| | Flag for mark all child components to non-transient. If true, all children components will be set to non-transient state and keep in saved components tree. For output in self-renderer region all content (By default, all content in <f:verbatim> tags and non-jsf elements in facelets, marked as transient - since, self-rendered ajax regions don't plain output for ajax processing). |
| lang | Code describing the language used in the generated markup for this component |
| layout | HTML layout for generated markup. Possible values: "block" for generating an HTML <div> element, "inline" for generating an HTML element, and "none" for generating no HTML element. There is a minor exception for the "none" case where a child element has the property "rendered" set to "false". In this case, we create an empty element with same ID as the child element to use as a placeholder for later processing. |
| rendered | If "false", this component is not rendered |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| title | Advisory title information about markup elements generated for this component |
| viewId | viewId for included page. |

Table 6.18. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | org.ajax4jsf.Include |
| component-family | javax.faces.Output |
| component-class | org.ajax4jsf.component.html.Include |
| renderer-type | org.ajax4jsf.components.AjaxIncludeRenderer |

6.9.2. Creating on a page

To use the component, it's necessary to place the following strings on a page:

Example:

```
<h:panelGroup id="wizard">
  <a4j:include viewId="/pages/include/first.xhtml" />
</h:panelGroup>
```

For navigation inside a page defined in viewId any components responsible for Ajax requests to the server generation are used.

For example, the following component on a page "/pages/include/first.xhtml"

Example:

```
...
<a4j:commandButton action="next" reRender="wizard" />
...
```

And in faces-config it's defined:

Example:

```
<navigation-rule>
  <from-view-id>/pages/include/first.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>next</from-outcome>
    <to-view-id>/pages/include/second.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```

In this case after a click on a button defined inside "first.xhtml" view, navigation is performed after an Ajax request (the same as standard JSF one) only inside this view.

6.9.3. Dynamical creation of a component from Java code

```
<import org.ajax4jsf.component.html.Include;
...
Include myInclude = new Include();
...
```

If **<a4j:include>** is defined this way, any Ajax request returning outcome inside generates navigation with this **<a4j:include>** .

Ajax Action for navigation implementation inside view must be placed inside **<a4j:include>** pages. Navigation defined by these pages is applied to the **<a4j:include>** element current for them.

As in the general case for Ajax Action component, if the **<a4j:action>** component inside **<a4j:include>** returns outcome defined as **<redirect/>**, Ajax submit is performed with navigation of the whole page and not only of the current view.

6.9.4. Relevant resources links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/include.jsf?c=include>] you can see the example of **<a4j:include>** usage and sources for the given example.

Some additional information can be found on the Ajax4Jsf Users Forum. [<http://jboss.com/index.html?module=bb&op=viewtopic&t=104158>]

6.10. <a4j:loadBundle>

The <a4j:loadBundle> component is similar to the same component from the JSF Core library. The component loads a resource bundle localized for the Locale of the current view and exposes it (as a Map) in the request attributes of the current request.

Table 6.19. a4j : loadBundle attributes

| Attribute Name | Description |
|----------------|---|
| basename | Base name of the resource bundle to be loaded. |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| id | Every component may have a unique id that is automatically created if omitted |
| rendered | If "false", this component is not rendered |
| var | Name of a request scope attribute under which the resource bundle will be exposed as a Map. |

Table 6.20. Component identification parameters

| Name | Value |
|------------------|--|
| component-type | org.ajax4jsf.Bundle |
| component-family | org.ajax4jsf.Bundle |
| component-class | org.ajax4jsf.component.html.AjaxLoadBundle |

6.10.1. Creating on a page

Simple component definition on a page:

Example:

```
<a4j:loadBundle baseName="demo.bundle.Messages" var="Message"/>
```

6.10.2. Dynamical creation of a component from Java code

Example:

```
import org.ajax4jsf.component.html.AjaxLoadBundle;
...
AjaxLoadBundle myBundle = new AjaxLoadBundle();
```

...

6.10.3. Key attributes and ways of usage

`<a4j:loadBundle>` allows to use reference to bundle messages during the Ajax re-rendering. `<a4j:loadBundle>` is a substitute for the `<f:loadBundle>` in JSF 1.1 which is not a JSF component originally. `<f:loadBundle>` is a jsp tag that load the bundle messages into the request scope when page is rendered. As soon as each Ajax request works in own request scope, the bundles loaded with `<f:loadBundle>` are unavailable. Instead of `<f:loadBundle>` that might be located anywhere on a page, the `<a4j:loadBundle>` should be declared inside the `<f:view>` (this does not matter in case on using Facelets) JSF 1.2 introduces the bundle registered in the faces-config.xml. This fixed the problem with `<f:loadBundle>`. Therefore, you can use this JSF 1.2 way to declare your bundles.

6.10.4. Relevant resources links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/bundle.jsf?c=loadBundle>] you can see the example of `<a4j:loadBundle>` usage and sources for the given example.

6.11. < a4j:loadScript >

6.11.1. Description

Inserts script links to the head element. Render the value of the component as the value of the `"src"` attribute, after passing it to the `getResourceURL()` method of the `ViewHandler` for this application, and passing the result through the `encodeResourceURL()` method of the `ExternalContext`.

Table 6.21. a4j : loadScript attributes

| Attribute Name | Description |
|----------------|---|
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| id | Every component may have a unique id that is automatically created if omitted |
| rendered | If "false", this component is not rendered |
| src | name of JavaScript resource to load. |

Table 6.22. Component identification parameters

| Name | Value |
|------------------|--|
| component-type | org.ajax4jsf.LoadScript |
| component-family | org.ajax4jsf.LoadScript |
| component-class | org.ajax4jsf.component.html.HtmlLoadScript |

| Name | Value |
|---------------|---------------------------------|
| renderer-type | org.ajax4jsf.LoadScriptRenderer |

6.11.2. Creating on a page

Simple Component definition on a page:

Example:

```
<a4j:loadScript src="scripts/someScript.js"/>
```

6.11.3. Dynamical creation of a component from Java code

Example:

```
import org.ajax4jsf.component.html.HtmlLoadScript;
...
HtmlLoadScript myScript = new HtmlLoadScript();
...
```

6.11.4. Key attributes and ways of usage

As it was mentioned above this component returns its value passing it to the `getResourceUR()` method of the `ViewHandler` for this application, and passing the result through the `encodeResourceURL()` method of the `ExternalContext`.

It means that the Context will be inserts automatically to the link. And calls like `resource://` will be properly handled.

Except this - you may be free to put your script links right from the child page while using facelets templates

.

6.11.5. Relevant resources links

Here [http://livedemo.exadel.com/richfaces-demo/richfaces/script.jsf?c=loadScript] you can see the example of `<a4j:loadScript>`s usage and sources for the given example.

6.12. < a4j:loadStyle >

6.12.1. Description

Inserts stylesheet links to the head element. Render the value of the component as the value of the `"src"` attribute, after passing it to the `getResourceURL()` method of the `ViewHandler` for this application, and passing the result through the `encodeResourceURL()` method of the `ExternalContext`.

Table 6.23. a4j : loadStyle attributes

| Attribute Name | Description |
|----------------|-------------|
| binding | |

| Attribute Name | Description |
|----------------|---|
| | The attribute takes a value-binding expression for a component property of a backing bean |
| id | Every component may have a unique id that is automatically created if omitted |
| rendered | If "false", this component is not rendered |
| src | name of JavaScript resource to load. |

Table 6.24. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | org.ajax4jsf.LoadStyle |
| component-family | org.ajax4jsf.LoadStyle |
| component-class | org.ajax4jsf.component.html.HtmlLoadStyle |
| renderer-type | org.ajax4jsf.LoadStyleRenderer |

6.12.2. Creating on a page

Simple Component definition on a page:

Example:

```
<a4j:loadStyle src="styles/style.css"/>
```

6.12.3. Dynamical creation of a component from Java code

Example:

```
import org.ajax4jsf.component.html.HtmlLoadStyle;
...
HtmlLoadScript myStyle = new HtmlLoadStyle();
...
```

6.12.4. Key attributes and ways of usage

As it was mentioned above this component returns its value passing it to the `getResourceUR()` method of the `ViewHandler` for this application, and passing the result via the `encodeResourceURL()` method of the `ExternalContext`.

It means that the Context will be inserts automatically to the link. And calls like `resource://` will be properly handled.

Except this - you may be free to put your stylesheet links right from the child page while using facelets templates.

6.12.5. Relevant resources links

Some additional information about usage of component can be found here. [<http://livedemo.exadel.com/richfaces-demo/richfaces/style.jsf?c=loadStyle>]

6.13. < a4j:log >

6.13.1. Description

The <a4j:log > component generates JavaScript for opening of the window with client-side debug information on an Ajax request.

Table 6.25. a4j : log attributes

| Attribute Name | Description |
|----------------|--|
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| dir | Direction indication for text that does not inherit directionality. Valid values are "LTR" (left-to-right) and "RTL" (right-to-left) |
| height | height of pop-up |
| hotkey | Keyboard key for activate (in combination with CTRL+SHIFT) log window. |
| id | Every component may have a unique id that is automatically created if omitted |
| lang | Code describing the language used in the generated markup for this component |
| level | log level, possible values : FATAL,ERROR,WARN,INFO,DEBUG,ALL. Component set level 'ALL' by default. |
| name | name of pop-up window |
| onclick | HTML: a script expression; a pointer button is clicked |
| ondblclick | HTML: a script expression; a pointer button is double-clicked |
| onkeydown | HTML: a script expression; a key is pressed down |
| onkeypress | HTML: a script expression; a key is pressed and released |
| onkeyup | HTML: a script expression; a key is released |
| onmousedown | |

| Attribute Name | Description |
|----------------|---|
| | HTML: script expression; a pointer button is pressed down |
| onmousemove | HTML: a script expression; a pointer is moved within |
| onmouseout | HTML: a script expression; a pointer is moved away |
| onmouseover | HTML: a script expression; a pointer is moved onto |
| onmouseup | HTML: script expression; a pointer button is released |
| popup | Render log as popup-window or as div element in page |
| rendered | If "false", this component is not rendered |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| title | Advisory title information about markup elements generated for this component |
| width | width of pop-up. |

Table 6.26. Component identification parameters

| Name | Value |
|------------------|-------------------------------------|
| component-type | org.ajax4jsf.Log |
| component-family | org.ajax4jsf.Log |
| component-class | org.ajax4jsf.component.html.AjaxLog |
| renderer-type | org.ajax4jsf.LogRenderer |

6.13.2. Creating on a page

To use the component, it's necessary to place the following string on a page:

```
<a4j:log/>
```

Then, in order to open a log window, press "CTRL+SHIFT+L" on a page with the component.

6.13.3. Dynamical creation of a component from Java code

Example:

```
import org.ajax4jsf.component.html.AjaxLog;
...
AjaxLog myLog = new AjaxLog();
...
```

6.13.4. Key attributes and ways of usage

Usage of the appropriate component attributes could change a representation level of debug information as well as the hot key for a window opening.

The hot key could be changed with the *"hotkey"* attribute, where it's necessary to define one letter that together with "CTRL+SHIFT" opens a window.

The *"level"* attribute with several possible values (FATAL, ERROR, WARN, INFO, ALL) could change a logging level.

The log could be generated not only in a new window, but also on the current page in a separate **<div>**, this is also controlled with the *"popup"* attribute on the component.

Example:

```
<a4j:log level="ALL" popup="false" width="400" height="200" />
```

The component defined this way is decoded on a page as **<div>** inside a page, where all the information beginning with informational message is generated.

Note:

<a4j:log> is getting renewed automatically after execution of Ajax requests. Don't renew **<a4j:log>** by using reRender!

6.13.5. Relevant resources links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/log.jsf?c=log>] you can see the example of **<a4j:log>** usage and sources for the given example.

6.14. < a4j:mediaOutput >

6.14.1. Description

The **<a4j:mediaOutput>** component implements one of the basic features specified in the framework. The component is a facility for generating images, video, sounds and other binary resources defined by you on-the-fly.

Table 6.27. a4j : mediaOutput attributes

| Attribute Name | Description |
|----------------|---|
| accesskey | This attribute assigns an access key to an element. An access key is a single character from the document character set. Note: Authors should consider the input method of the expected reader when specifying an accesskey |
| align | bottom middle top left right Deprecated. This attribute specifies the position of an IMG, OBJECT, or APPLET |

| Attribute Name | Description |
|----------------|---|
| | with respect to its context. The following values for align concern the object's position with respect to surrounding text: * bottom: means that the bottom of the object should be vertically aligned with the current baseline. This is the default value. * middle: means that the center of the object should be vertically aligned with the current baseline. * top: means that the top of the object should be vertically aligned with the top of the current text line |
| archive | space-separated list of URIs |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| border | Deprecated. This attribute specifies the width of an IMG or OBJECT border, in pixels. The default value for this attribute depends on the user agent |
| cacheable | If "true", the resource is cached (on the server and the client sides). |
| charset | The character encoding of a resource designated by this hyperlink |
| classid | identifies an implementation |
| codebase | base URI for classid, data, archive |
| codetype | content type for code |
| converter | ID of a converter to be used or a reference to a converter. |
| coords | This attribute specifies the position and shape on the screen. The number and order of values depends on the shape being defined. Possible combinations: * rect: left-x, top-y, right-x, bottom-y. * circle: center-x, center-y, radius. Note. When the radius value is percentage value, user agents should calculate the final radius value based on the associated object's width and height. The radius should be the smaller value of the two. * poly: x1, y1, x2, y2, ..., xN, yN. The first x and y coordinate pair and the last should be the same to close the polygon. When these coordinate values are not the same, user agents should infer an additional coordinate pair to close the polygon. Coordinates are relative to the top, left corner of the object. All values are lengths. All values are separated by commas |

| Attribute Name | Description |
|----------------|---|
| createContent | Method call expression to send generated resource to OutputStream. It must have two parameter with a type of java.io.OutputStream and java.lang.Object (deserialized value of data attribute) |
| declare | declare but don't instantiate flag |
| dir | Direction indication for text that does not inherit directionality. Valid values are "LTR" (left-to-right) and "RTL" (right-to-left) |
| element | Name of html element for resource link - may be <a> <object> <applet> <script> or <link> |
| expires | The attribute allows to manage caching and defines the period after which a resource is reloaded. |
| hreflang | Base language of a resource specified with the href attribute; hreflang may only be used with href |
| hspace | Deprecated. This attribute specifies the amount of white space to be inserted to the left and right of an IMG, APPLET, or OBJECT. The default value is not specified, but is generally a small, non-zero length |
| id | Every component may have a unique id that is automatically created if omitted |
| ismap | use server-side image map |
| lang | Code describing the language used in the generated markup for this component |
| lastModified | The attribute allows to manage caching. A browser can send request with the header "If-Modified-Since" for necessity of object reloading. If time of modification is earlier, then the framework doesn't call generation and return code 304. |
| contentType | Generated content mime-type for append to response header ('image/jpeg' etc) |
| onblur | JavaScript code. The onblur event occurs when an element loses focus either by the pointing device or by tabbing navigation. It may be used with the same elements as onfocus |
| onclick | HTML: a script expression; a pointer button is clicked |

| Attribute Name | Description |
|----------------|--|
| ondblclick | HTML: a script expression; a pointer button is double-clicked |
| onfocus | JavaScript code. The onfocus event occurs when an element gets focus |
| onkeydown | HTML: a script expression; a key is pressed down |
| onkeypress | HTML: a script expression; a key is pressed and released |
| onkeyup | HTML: a script expression; a key is released |
| onmousedown | HTML: script expression; a pointer button is pressed down |
| onmousemove | HTML: a script expression; a pointer is moved within |
| onmouseout | HTML: a script expression; a pointer is moved away |
| onmouseover | HTML: a script expression; a pointer is moved onto |
| onmouseup | HTML: script expression; a pointer button is released |
| rel | The relationship from the current document to the anchor specified by this hyperlink. The value of this attribute is a space-separated list of link types |
| rendered | If "false", this component is not rendered |
| rev | A reverse link from the anchor specified by this hyperlink to the current document. The value of this attribute is a space-separated list of link types |
| session | If "true", a session for an object generation is restored. |
| shape | default rect circle poly [CI] This attribute specifies the shape of a region. Possible values: * default: Specifies the entire region. * rect: Define a rectangular region. * circle: Define a circular region. * poly: Define a polygonal region. |
| standby | message to show while loading |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| tabindex | This attribute specifies the position of the current element in the tabbing order for the current document. |

| Attribute Name | Description |
|----------------|--|
| | This value must be a number between 0 and 32767. User agents should ignore leading zeros |
| target | This attribute specifies the name of a frame where a document is to be opened. By assigning a name to a frame via the name attribute, authors can refer to it as the "target" of links defined by other elements |
| title | Advisory title information about markup elements generated for this component |
| type | The content type of the resource designated by this hyperlink |
| uriAttribute | Name of attribute for resource-link attribute ('href' for <a>, 'src' for or <script>, etc |
| usemap | use client-side image map |
| value | Data value calculated at render time and stored in URI (also as part of cache Key), at generation time passed to send method. Can be used for update cache at change of generating conditions, and for creating beans as "Lightweight" pattern components (request scope). IMPORTANT: Since serialized data stored in URI, avoid using big objects. |
| vspace | Deprecated. This attribute specifies the amount of white space to be inserted above and below an IMG, APPLET, or OBJECT. The default value is not specified, but is generally a small, non-zero length |

Table 6.28. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | org.ajax4jsf.MediaOutput |
| component-family | org.ajax4jsf.Resource |
| component-class | org.ajax4jsf.component.html.MediaOutput |
| renderer-type | org.ajax4jsf.MediaOutputRenderer |

6.14.2. Creating on a page

To use the component it's necessary to define it on a page and set Java methods for data keeping and data transmission to output stream.

Component definition on a page for graphical data output

Example:

```
...
<a4j:mediaOutput element="img" cacheable="false" session="true"
    createContent="#{paintBean.paint}" value="#{paintData}"
    mimeType="image/jpeg" />
...
```

Here is the content of `paintData` that is a bean containing output data

Example:

```
package demo;

public class PaintData implements Serializable{
    private static final long serialVersionUID = 1L;
    Integer width=100;
    Integer weight=50;
    ...
}
```

The `Paint` method of the `paintBean` class is a method transmitting graphical data into output stream.

Example:

```
public void paint(OutputStream out, Object data) throws IOException{
    <!--...Some code that puts binary data to "out" Stream-->
}
```

6.14.3. Dynamical creation of a component from Java code

Example:

```
import org.ajax4jsf.component.html.MediaOutput;
...
MediaOutput myMedia = new MediaOutput ();
...
```

6.14.4. Key attributes and ways of usage

As it was shown in the example above there are two main components:

- `"createContent"` specifies a method accepting 2 parameters. The first (of `java.io.OutputStream` type) defines a stream, where any binary data is output. The second (of `java.lang.Object` type) contains deserialized object with data specified in the `"value"` attribute.
- `Value` specifies a bean class keeping data for transmitting into a method that transmits it into a stream.

Note:

A bean class transmitted into `value` should implement `Serializable` interface.

Hence, when using the component it's possible to output your data of any type on a page with Ajax requests.

6.14.5. Relevant resources links

Here [\[http://livedemo.exadel.com/richfaces-demo/richfaces/mediaOutput.jsf?c=mediaOutput\]](http://livedemo.exadel.com/richfaces-demo/richfaces/mediaOutput.jsf?c=mediaOutput) you can see the example of `<a4j:mediaOutput>` usage and sources for the given example.

6.15. < a4j:outputPanel >

6.15.1. Description

The component is used for components grouping in the Ajax output area, which offers several additional output opportunities such as inserting of non-present in tree components, saving of transient elements after Ajax request and some others.

Table 6.29. a4j : outputPanel attributes

| Attribute Name | Description |
|----------------|---|
| ajaxRendered | Defines, whether the content of this component must be (or not) included in AJAX response created by parent AJAX Container, even if it is not forced by reRender list of ajax action. Ignored if component marked to output by Ajax action. default false |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| dir | Direction indication for text that does not inherit directionality. Valid values are "LTR" (left-to-right) and "RTL" (right-to-left) |
| id | Every component may have a unique id that is automatically created if omitted |
| keepTransient | Flag for mark all child components to non-transient. If true, all children components will be set to non-transient state and keep in saved components tree. For output in self-renderer region all content (By default, all content in <f:verbatim> tags and non-jsf elements in facelets, marked as transient - since, self-rendered ajax regions don't plain output for ajax processing). |
| lang | Code describing the language used in the generated markup for this component |
| layout | HTML layout for generated markup. Possible values: "block" for generating an HTML <div> element, "inline" for generating an HTML element, and "none" for generating no HTML element. There is a minor exception for the "none" case where a child element has the property "rendered" set to "false". In |

| Attribute Name | Description |
|----------------|--|
| | this case, we create an empty element with same ID as the child element to use as a placeholder for later processing. |
| onclick | HTML: a script expression; a pointer button is clicked |
| ondblclick | HTML: a script expression; a pointer button is double-clicked |
| onkeydown | HTML: a script expression; a key is pressed down |
| onkeypress | HTML: a script expression; a key is pressed and released |
| onkeyup | HTML: a script expression; a key is released |
| onmousedown | HTML: script expression; a pointer button is pressed down |
| onmousemove | HTML: a script expression; a pointer is moved within |
| onmouseout | HTML: a script expression; a pointer is moved away |
| onmouseover | HTML: a script expression; a pointer is moved onto |
| onmouseup | HTML: script expression; a pointer button is released |
| rendered | If "false", this component is not rendered |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| title | Advisory title information about markup elements generated for this component |

Table 6.30. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | org.ajax4jsf.OutputPanel |
| component-family | javax.faces.Panel |
| component-type | org.ajax4jsf.ajax.OutputPanel |
| component-class | org.ajax4jsf.component.html.HtmlAjaxOutputPanel |
| renderer-type | org.ajax4jsf.components.AjaxOutputPanelRenderer |

6.15.2. Creating on a page

Here is the simplest way for a component creation on a page.

Example:

```
<a4j:outputPanel>
<!--...Some Content Inside-->
</a4j:outputPanel>
```

6.15.3. Dynamical creation of a component from Java code**Example:**

```
import org.ajax4jsf.component.html.HtmlAjaxOutputPanel;
...
HtmlAjaxOutputPanel myPanel = new HtmlAjaxOutputPanel();
```

6.15.4. Key attributes and ways of usage

<a4j:outPanel> allows marking of a page area, which is updated on Ajax response. Anyway, **<a4j:outputPanel>** usage is optional, as in RichFaces it's possible to indicate any existing component id on a component view in order to define updating areas. To speed up the performance, RichFaces updates only a component tree. **<a4j:outputPanel>** usage is recommended for wrapping components that aren't rendered during the primary non-ajax response, as the components don't present in a component tree.

Example:

```
<a4j:support ... reRender="mypanel"/>
...
<a4j:outputPanel id="mypanel">
  <h:panelGrid rendered="#{not empty foo.bar}">
    ...
  </h:panelGrid>
</a4j:outputPanel>
```

In addition to the areas directly indicated in *"reRender"* attribute of Ajax components, **<a4j:outputPanel>** allows to update a part of a page basing on its own flag. The flag is defined by the *"ajaxRendered"* attribute. The flag is commonly used when a part of a page must be updated or can be updated on any response.

Example:

```
<a4j:outputPanel ajaxRendered="true">
  <h:messages/>
</a4j:outputPanel>
```

On default **<a4j:outputPanel>** is output as a pair of opening and closing html **** tag, but with the help of the layout attribute this output way could be changed. There are three variants for this component value:

- inline (default)
- block
- none

If `layout="block"` is chosen, the component is rendered as a pair of opening and closing `<div>` tag, to which it's possible to apply any available style attributes available for block tags.

`Layout="none"` helps to avoid an unnecessary tag round a context that could or couldn't be rendered according to the defined *"rendered"* attribute conditions. If an inner context isn't rendered, `<a4j:outputPanel>` is rendered as a `` tag with the id equal to an id of a child component and `"display:none"` style. If a child component is rendered, `<a4j:outputPanel>` doesn't present at all in a final code.

Example:

```
<a4j:support .... reRender="mypanel"/>
...
<a4j:outputPanel layout="none">
  <h:panelGrid id="mypanel" rendered="#{not empty foo.bar}">
    ...
  </h:panelGrid>
</a4j:outputPanel>
```

As you see, the code is very similar to the one shown above, but *"reRender"* attribute refers directly to the updating panelGrid and not to the framing outputPanel, and it's more semantically correct.

`<a4j:outPanel>` should be used for non-JSF component part framing, which is to be updated on Ajax response, as RichFaces specifies the list of updating areas as a list of an existing JSF component.

On default non-JSF context isn't saved in a component tree, but is rendered anew every time. To accelerate the processing speed and Ajax response input speed, RichFaces saves non-JSF context in a component tree on default. This option could be canceled by *"keepTransient"* attribute that cancels transient flag forced setting for child components. This flag setting keeps the current value set by child components.

Note: In JSF 1.1 implementation and lower, where non-JSF context should be framed with the *"f:verbatim"* attribute, `<a4j:outputPanel>` doesn't improve this JSF implementation option in any way, so you still have to use this tag where it's necessary without RichFaces usage.

RichFaces allows setting Ajax responses rendering directly basing on component tree nodes without referring to the JSP (XHTML) page code. It could be defined by `selfRendered` attribute setting to *"true"* on `<a4j:region>` and could help considerably speed up a response output. However, if a transient flag is kept as it is, this rapid processing could cause missing of transient components that present on view and don't come into a component tree. Hence, for any particular case you could choose a way for you application optimization: speed up processing or redundant memory for keeping tree part earlier defined a transient.

6.15.5. Relevant resources links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/outputPanel.jsf?c=outputPanel>] you can see the example of `<a4j:outputPanel>` usage and sources for the given example.

Some additional information about usage of component can be found here [<http://www.jboss.com/index.html?module=bb&op=viewtopic&p=4052203#4052203>].

6.16. < a4j:page >

6.16.1. Description

<a4j:page> is a deprecated component used for solving of incompatibility problems in early Ajax4jsf and MyFaces versions. The component encodes the full html page structure.

Table 6.31. a4j : page attributes

| Attribute Name | Description |
|----------------|--|
| ajaxListener | MethodBinding representing an action listener method that will be notified when this component is activated by the ajax Request and handle it. The expression must evaluate to a public method that takes an AjaxEvent parameter, with a return type of void |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| contentType | Set custom mime content type to response |
| dir | Direction indication for text that does not inherit directionality. Valid values are "LTR" (left-to-right) and "RTL" (right-to-left) |
| format | Page layout format (html, xhtml, html-transitional, html-3.2) for encoding DOCTYPE, namespace and Content-Type definitions |
| id | Every component may have a unique id that is automatically created if omitted |
| immediate | Flag indicating that, if this component is activated by ajaxrequest, notifications should be delivered to interested listeners and actions immediately (that is, during Apply Request Values phase) rather than waiting until Invoke Application phase |
| lang | Code describing the language used in the generated markup for this component |
| namespace | Set html element default namespace |
| onload | JavaScript code to execute on a page load. |
| onunload | JavaScript code to execute on a page unload. |
| pageTitle | String for output as a page title. |
| rendered | If "false", this component is not rendered |
| selfRendered | |

| Attribute Name | Description |
|----------------|--|
| | if "true", self-render subtree at InvokeApplication (or Decode, if immediate property set to true) phase |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| title | Advisory title information about markup elements generated for this component |

Table 6.32. Component identification parameters

| Name | Value |
|------------------|--|
| component-type | org.ajax4jsf.components.Page |
| component-family | org.ajax4jsf.components.AjaxRegion |
| component-class | org.ajax4jsf.component.html.HtmlPage |
| renderer-type | org.ajax4jsf.components.AjaxPageRenderer |

6.16.2. Creating on a page

This component should be defined as a child component for **<f:view>**:

```
<f:view>
  <a4j:page>
    <f:facet name="head">
      <!--...Head Content here-->
    </f:facet>
    <!--...Page Content here-->
  </a4j:page>
</f:view>
```

This structure is rendered as:

Example:

```
<HTML>
  <HEAD>
    <!--...Head Content here-->
  </HEAD>
  <body >
    <!--...Page Content Here-->
  </body>
</HTML>
```

6.16.3. Dynamical creation of a component from Java code

Example:

```
import org.ajax4jsf.component.html.HtmlPage;
...
HtmlPage myPage = new HtmlPage();
...
```

6.16.4. Key attributes and ways of usage

The component is mostly used to solve the following problem with MyFaces for earlier Ajax4jsf versions: in MyFaces `<f:view>` doesn't get control over the `" RENDER_RESPONSE "` phase, thus Ajax can't get control and make a response also. To avoid this problem it was necessary to use `<a4j:page>` on a page around the Ajax updatable area. In the last versions of both frameworks the problem is successfully fixed and no `<a4j:page>` usage is required.

The component is rendered as a full HTML page template (it was shown in the example). The `" head "` section is defined with the help of the corresponding facet with the name="head" and also there is an attribute with the same name for `"contentType"` definition.

6.16.5. Relevant resources links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/page.jsf?c=page>] you can see the example of `<a4j:page>` usage and sources for the given example.

6.17. < a4j:poll >

6.17.1. Description

The `<a4j:poll>` component allows periodical sending of Ajax requests to a server and is used for a page updating according to a specified time interval.

Table 6.33. a4j : poll attributes

| Attribute Name | Description |
|------------------|---|
| action | MethodBinding pointing at the application action to be invoked, if this UIComponent is activated by the user, during the Apply Request Values or Invoke Application phase of the request processing lifecycle, depending on the value of the immediate property |
| actionExpression | The action method binding expression. |
| actionListener | MethodBinding pointing at method accepting an ActionEvent with return type void |
| ajaxSingle | if "true", submits ONLY one field/link, instead of all form controls |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |

| Attribute Name | Description |
|--------------------|--|
| bypassUpdates | If "true", after process validations phase it skips updates of model beans on a force render response. It can be used for validating components input |
| data | Serialized (on default with JSON) data passed on the client by a developer on AJAX request. It's accessible via "data.foo" syntax |
| enabled | Enable/disable polling |
| eventsQueue | Name of requests queue to avoid send next request before complete other from same event. Can be used to reduce number of requests of frequently events (key press, mouse move etc.) |
| focus | id of element to set focus after request completed on client side |
| id | Every component may have a unique id that is automatically created if omitted |
| ignoreDupResponses | Attribute allows to ignore an Ajax Response produced by a request if the newest 'similar' request is in a queue already. ignoreDupResponses="true" does not cancel the request while it is processed on the server, but just allows to avoid unnecessary updates on the client side if the response isn't actual now |
| immediate | True means, that the default ActionListener should be executed immediately (i.e. during Apply Request Values phase of the request processing lifecycle), rather than waiting until the Invoke Application phase |
| interval | Interval (in ms) for call poll requests. Default value 1000 (1 sec) |
| limitToList | If "true", updates on client side ONLY elements from this 'reRender' property. If "false" (default) updates all rendered by ajax region components |
| onbeforedomupdate | JavaScript code for call before DOM has been updated on client side |
| oncomplete | JavaScript code for call after request completed on client side |
| onsubmit | JavaScript code for call before submission of ajax event |
| rendered | If "false", this component is not rendered |

| Attribute Name | Description |
|----------------|---|
| reRender | Id[s] (in format of call <code>UIComponent.findComponent()</code> of components, rendered in case of <code>AjaxRequest</code> caused by this component. Can be single id, comma-separated list of Id's, or EL Expression with array or Collection |
| status | ID (in format of call <code>UIComponent.findComponent()</code> of Request status component |
| timeout | Timeout (in ms) for request |

Table 6.34. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | <code>org.ajax4jsf.Poll</code> |
| component-family | <code>org.ajax4jsf.components.AjaxPoll</code> |
| component-class | <code>org.ajax4jsf.component.html.AjaxPoll</code> |
| renderer-type | <code>org.ajax4jsf.components.AjaxPollRenderer</code> |

6.17.2. Creating on a page

To create the simplest variant on a page use the following syntax:

Example:

```
<a4j:poll interval="500" reRender="grid"/>
```

6.17.3. Dynamical creation of a component from Java code

Example:

```
import org.ajax4jsf.component.html.AjaxPoll;
...
AjaxPoll myPoll = new AjaxPoll();
...
```

6.17.4. Key attributes and ways of usage

The `<a4j:poll>` componet is used for periodical polling of server data. In order to use the component it's necessary to set an update interval. The `"interval"` attribute defines an interval in milliseconds between the previous response and the next request. The total period beetween two requests generated by the `<a4j:poll>` component is a sum of an `"interval"` attribute value and server response time. Default value for `"interval"` attribute is set to `"1000"` milliseconds (1 second). See an example of definition in the "Creating on a page" section [88].

The *"timeout"* attribute defines response waiting time in milliseconds. If a response isn't received during this period a connection is aborted and the next request is sent. Default value for *"timeout"* attribute isn't set.

The *"enabled"* attribute defines should the **<a4j:poll>** send request or not. It's necessary to render the **<a4j:poll>** to apply the current value of *"enabled"* attribute. You can use an EL-expression for *"enabled"* attribute to point to a bean property. An example of usage of mentioned above attributes [88] is placed below:

Example:

```
...
    <a4j:region>
        <h:form>
            <a4j:poll id="poll" interval="1000" enabled="#{userBean.pollEnabled}"
reRender="poll,grid"/>
        </h:form>
    </a4j:region>
    <h:form>
        <h:panelGrid columns="2" width="80%" id="grid">
            <h:panelGrid columns="1">
                <h:outputText value="Polling Inactive" rendered="#{not
userBean.pollEnabled}"></h:outputText>
                <h:outputText value="Polling Active"
rendered="#{userBean.pollEnabled}"></h:outputText>
                <a4j:commandButton style="width:120px" id="control"
                    value="#{userBean.pollEnabled?'Stop':'Start'}"
Polling"
                    reRender="poll, grid">
                    <a4j:actionParam name="polling" value="#{!userBean.pollEnabled}"
                        assignTo="#{userBean.pollEnabled}" />
                </a4j:commandButton>
            </h:panelGrid>
            <h:outputText id="serverDate" style="font-size:16px" value="Server
Date: #{userBean.date}" />
        </h:panelGrid>
    </h:form>
...
```

The example shows how date and time are updated on a page in compliance with data taken from a server. The **<a4j:poll>** componet sends requests to the server every second. *"reRender"* attribute for **<a4j:poll>** contains value of its own Id. Hence, it renders itself for applying the current value of *"enabled"* attribute.

Note:

The form around the **<a4j:poll>** component is required.

6.17.5. Relevant resources links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/poll.jsf?c=poll>] you can see the example of **<a4j:poll>** usage and sources for the given example.

The additional information about component usage you can find here : RichFaces Users Forum [<http://jboss.com/index.html?module=bb&op=viewtopic&t=103909>].

6.18. < a4j:portlet >

6.18.1. Description

The **<a4j:portlet>** can be used in portals. The main component purpose is realization of possibility to create several instances the same portlet on one page.

Table 6.35. a4j : portlet attributes

| Attribute Name | Description |
|----------------|---|
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| id | Every component may have a unique id that is automatically created if omitted |
| rendered | If "false", this component is not rendered |

Table 6.36. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | org.ajax4jsf.Portlet |
| component-family | org.ajax4jsf.component.Portlet |
| component-class | org.ajax4jsf.component.html.HtmlPortlet |

6.18.2. Creating on a page

```
<f:view>
  <a4j:portlet>
    ...
  </a4j:portlet>
</f:view>
```

6.18.3. Dynamical creation of a component from Java code

```
import org.ajax4jsf.component.html.HtmlPortlet;
...
HtmlPortlet myPortlet = new HtmlPortlet();
...
```

6.18.4. Key attributes and ways of usage

Portal page can include some instances of the same portlet but clientId of elements should be different for each window. In that case 'namespace' is used for each portlet. The **<a4j:portlet>** implements NamingContainer interface and adds namespace to all componets on a page. All portlet content should be wrapped by **<a4j:portlet>** for resolving problems mentioned before.

6.18.5. Relevant resources links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/portlet.jsf?c=portlet>] you can see the example of `<a4j:portlet>` usage and sources for the given example.

The additional information about component usage you can find here: Ajax4Jsf Users Forum. [<http://www.jboss.com/index.html?module=bb&op=viewtopic&t=107325>]

Portlet Sample could be checked out from JBoss SVN: portal-echo application. [<http://anonsvn.jboss.org/repos/ajax4jsf/trunk/samples/portal-echo/>]

Usage instructions for this demo could be found at the corresponding: portal-echo application. [<http://www.jboss.com/index.html?module=bb&op=viewtopic&t=107325>]

6.19. < a4j:push >

6.19.1. Description

The `<a4j:push>` periodically perform Ajax request to server, to simulate 'push' data.

Table 6.37. a4j : push attributes

| Attribute Name | Description |
|------------------|---|
| action | MethodBinding pointing at the application action to be invoked, if this UIComponent is activated by the user, during the Apply Request Values or Invoke Application phase of the request processing lifecycle, depending on the value of the immediate property |
| actionExpression | The action method binding expression. |
| actionListener | MethodBinding pointing at method accepting an ActionEvent with return type void |
| ajaxSingle | if "true", submits ONLY one field/link, instead of all form controls |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| bypassUpdates | If "true", after process validations phase it skips updates of model beans on a force render response. It can be used for validating components input |
| data | Serialized (on default with JSON) data passed on the client by a developer on AJAX request. It's accessible via "data.foo" syntax |
| enabled | Enable/disable pushing |
| eventProducer | |

| Attribute Name | Description |
|--------------------|--|
| | MethodBinding pointing at method accepting an PushEventListener with return type void. User bean must register this listener and send EventObject to this listener on ready. |
| eventsQueue | Name of requests queue to avoid send next request before complete other from same event. Can be used to reduce number of requests of frequently events (key press, mouse move etc.) |
| focus | id of element to set focus after request completed on client side |
| id | Every component may have a unique id that is automatically created if omitted |
| ignoreDupResponses | Attribute allows to ignore an Ajax Response produced by a request if the newest 'similar' request is in a queue already. ignoreDupResponses="true" does not cancel the request while it is processed on the server, but just allows to avoid unnecessary updates on the client side if the response isn't actual now |
| immediate | True means, that the default ActionListener should be executed immediately (i.e. during Apply Request Values phase of the request processing lifecycle), rather than waiting until the Invoke Application phase |
| interval | Interval (in ms) for call push requests. Default value 1000 (1 sec) |
| limitToList | If "true", updates on client side ONLY elements from this 'reRender' property. If "false" (default) updates all rendered by ajax region components |
| onbeforedomupdate | JavaScript code for call before DOM has been updated on client side |
| oncomplete | JavaScript code for call after request completed on client side |
| rendered | If "false", this component is not rendered |
| reRender | Id[s] (in format of call UIComponent.findComponent()) of components, rendered in case of AjaxRequest caused by this component. Can be single id, comma-separated list of Id's, or EL Expression with array or Collection |

| Attribute Name | Description |
|----------------|--|
| status | ID (in format of call <code>UIComponent.findComponent()</code>) of Request status component |
| timeout | Timeout (in ms) for request |

Table 6.38. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | <code>org.ajax4jsf.Push</code> |
| component-family | <code>org.ajax4jsf.components.AjaxPush</code> |
| component-class | <code>org.ajax4jsf.component.html.AjaxPush</code> |
| renderer-type | <code>org.ajax4jsf.components.AjaxPushRenderer</code> |

6.19.2. Creating on a page

```
<a4j:push reRender="msg" eventProducer="#{messageBean.addListener}" interval="3000"/>
```

6.19.3. Dynamical creation of a component from Java code

```
import org.ajax4jsf.component.html.AjaxPush;
...
AjaxPush myPush = new AjaxPush();
...
```

6.19.4. Key attributes and ways of usage

The **<a4j:push>** implements reverse Ajax technique. The component makes requests to minimal code (not to JSF tree) in order to check presence of messages in a queue. If a message exists, a complete request is performed. **<a4j:push>** registers `EventListener` in the bean, which receives messages about an event presence.

The bean, for example, could be subscribed to Java Messaging Service (JMS [<http://java.sun.com/products/jms/>]) topic or it could be implemented as Message Driven Bean (MDB) in order to send a message to the **<a4j:push>** component about an event presence. In the presence of the event some action occurs.

Thus, a work paradigm with the **<a4j:push>** component corresponds to an anisochronous model, but not to pools as for **<a4j:poll>** component. See the simplest example below:

Example:

```
...
class MyPushEventListener implements PushEventListener {
    public void onEvent(EventObject evt) {
        System.out.println(evt.getSource());
        //Some action
    }
}
```

...

Code for `EventListener` registration in the bean is placed below:

Example:

```
...
public void addListener(EventListener listener) {
    synchronized (listener) {
        if (this.listener != listener) {
            this.listener = (PushEventListener) listener;
        }
    }
    ...
}
```

A page code for this example is placed below.

Example:

```
...
<a4j:status startText="in progress" stopText="done"/>
<a4j:form>
    <a4j:region>
        <a4j:push reRender="msg" eventProducer="#{pushBean.addListener}"
interval="2000"/>
    </a4j:region>
    <a4j:outputPanel id="msg" >
        <h:outputText value="#{pushBean.date}">
            <f:convertDateTime type="time"/>
        </h:outputText>
    </a4j:outputPanel>
    <a4j:commandButton value="Push!!" action="#{pushBean.push}"
ajaxSingle="true"/>
    </a4j:form>
...
}
```

The example shows how date is updated on a page in compliance with data taken from a server. In the example `"interval"` attribute has value `"2000"`. This attribute defines an interval in milliseconds between the previous response and the next request. Default value is set to `"1000"` milliseconds (1 second). It's possible to set value equal to `"0"`. In this case connection is permanent.

The `"timeout"` attribute defines response waiting time in milliseconds. If a response isn't received during this period a connection is aborted and the next request is sent. Default value for `"timeout"` attribute isn't set. Usage of `"interval"` and `"timeout"` attributes gives an opportunity to set short polls of queue state or long connections, or permanent connection.

Note:

The form around the `<a4j:push>` component is required.

6.19.5. Relevant resources links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/push.jsf?c=push>] you can see the example of `<a4j:push>` usage and sources for the given example.

6.20. < a4j:region >

6.20.1. Description

The <a4j:region> component defines an area that is decoded on the server after Ajax submission.

Table 6.39. a4j : region attributes

| Attribute Name | Description |
|------------------|--|
| ajaxListener | MethodBinding representing an action listener method that will be notified when this component is activated by the ajax Request and handle it. The expression must evaluate to a public method that takes an AjaxEvent parameter, with a return type of void |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| id | Every component may have a unique id that is automatically created if omitted |
| immediate | Flag indicating that, if this component is activated by ajaxrequest, notifications should be delivered to interested listeners and actions immediately (that is, during Apply Request Values phase) rather than waiting until Invoke Application phase |
| rendered | If "false", this component is not rendered |
| renderRegionOnly | Flag to disable rendering in AJAX responses content outside of active region. If this attribute set to "true" , no one of the components outside of region will be included to AJAX response. If set to "false", search for components to include in response will be performed on all tree. Default "false" |
| selfRendered | if "true", self-render subtree at InvokeApplication (or Decode, if immediate property set to true) phase |

Table 6.40. Component identification parameters

| Name | Value |
|------------------|--|
| component-type | org.ajax4jsf.AjaxRegion |
| component-family | org.ajax4jsf.AjaxRegion |
| component-class | org.ajax4jsf.component.html.HtmlAjaxRegion |
| renderer-type | org.ajax4jsf.components.AjaxRegionRenderer |

6.20.2. Creating on a page

Here is an example of the region decoding on a page.

```
<a4j:region>
  <!--...Some content that will be decoded on server after Ajax request.-->
</a4j:region>
```

6.20.3. Dynamical creation of a component from Java code

Example:

```
import org.ajax4jsf.component.html.HtmlAjaxRegion;
...
HtmlAjaxRegion newRegion = new HtmlAjaxRegion();
...
```

6.20.4. Key attributes and ways of usage

The region is a component used for manipulation with components sent to the server. It sets particular processing parameters for an area on the server, i.e. the region deals with data input on the server and has no direct impact on output. To read more on the components responsible for out, see *"reference"*

The region marks an area page that is decoded on the server. In most cases it is not necessary to use the region, as ViewRoot is a default region. This component helps to reduce data quantity processed by the server, but the region doesn't influence on the standard submission rules. It means that:

- The area that is to be submitted onto the server should be embedded in **<h:form/a4j:form>** component.
- The whole form is submitted on Ajax response and not a region that request is performed from.

Example:

```
<h:form id="form1">
  <a4j:region>
    <a4j:commandLink reRender="someID" value="Link" id="link1"/>
    <!--...Some content that will be decoded on server after Ajax request.-->
  </a4j:region>
</h:form>
```

Hence, the **<a4j:commandLink>** request generation causes full "form1" form submission onto the server, the only difference is that a component tree part decoded on the serve is the part included into the region.

The regions could be nested in any order, the server picks out and decodes only the region, which contains a particular component that sends a request.

Example:

```
<a4j:region>
  <a4j:commandLink reRender="someID" value="Link" id="link1"/>
</a4j:region>
  <a4j:commandLink reRender="someID" value="Link" id="link2"/>
```

```

<!--..Some content that will be decoded on server after Ajax request.-->
</a4j:region >
<!--..Some content that will be decoded on server after Ajax request.-->
</a4j:region >

```

Therefore, the external region is decoded for the "link1" and the internal one is decoded for the "link2".

RichFaces allows setting Ajax responses rendering directly basing on component tree nodes without referring to the JSP (XHTML) page code. It could be defined by *"selfRendered"* attribute setting to *"true"* on **<a4j:region>** and could help considerably speed up a response output. However, this rapid processing could cause missing of transient components that present on view and don't come into a component tree as well as omitting of **<a4j:outputPanel>** usage described below.

Example:

```

<a4j:region selfRendered ="true">
  <a4j:commandLink reRender="someID" value="Link" id="link1"/>
  <!--..Some content with HTML used ("br" , "h1" and other tags used)-->
</a4j:region >

```

In this case, the processing is quicker and going on without referring to a page code, but the HTML code that isn't saved in a component tree could be lost. Thus, this optimization should be very carefully performed and a usage of the additional components RichFaces (**<a4j:outputPanel>**) is required.

The processing could be also accelerated if a region decoded for the processing passes straight away into Encode. But to update some data out of the region or on another region, use the *"renderRegionOnly"* attribute set to *"false"* ("true on default") to change this behaviour.

Example:

```

<a4j:region renderRegionOnly="true">
  <a4j:commandLink reRender="someID2" value="Link1" id="link1"/>
  <h:panelGroup id="someId1">
    </h:panelGroup>
</a4j:region>
<a4j:region renderRegionOnly="false">
  <a4j:commandLink reRender="someID1" value="Link2" id="link2"/>
  <h:panelGroup id="someId1">
    </h:panelGroup>
</a4j:region>

```

This example shows that one of the regions is decoded when a link is used inside. Nevertheless, if after processing the "link1" is clicked, the first region passes into Encode as a root region and encode performance time is reduced. This optimization doesn't allow data update out of the region and should be implemented very carefully. The data out of the region described with *"renderRegionOnly"="false"* is updated successfully.

6.20.5. Relevant resources links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/region.jsf?c=region>] you can see the example of **<a4j:region>** usage and sources for the given example.

6.21. <a4j:repeat >

6.21.1. Description

The <a4j:repeat> component implements a basic iteration component allowing to update a set of its children with AJAX.

Table 6.41. a4j : repeat attributes

| Attribute Name | Description |
|----------------|---|
| ajaxKeys | This attribute defines strings that are updated after an AJAX request. |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| componentState | It defines EL-binding for a component state for saving or redefinition. |
| first | A zero-relative row number of the first row to display |
| id | Every component may have a unique id that is automatically created if omitted |
| rendered | If "false", this component is not rendered |
| rowKeyVar | The attribute provides access to a row key in a Request scope. |
| rows | A number of rows to display, or zero for all remaining rows in the table |
| stateVar | The attribute provides access to a component state on the client side. |
| value | The current value for this component. |
| var | A request-scope attribute via which the data object for the current row will be used when iterating |

Table 6.42. Component identification parameters

| Name | Value |
|------------------|--|
| component-type | org.ajax4jsf.Repeat |
| component-family | javax.faces.Data |
| component-class | org.ajax4jsf.component.html.HtmlAjaxRepeat |
| renderer-type | org.ajax4jsf.components.RepeatRenderer |

6.21.2. Creating on a page

The component definition on a page is the same as for the *"facelets"* component:

```
<a4j:repeat id="detail" value="#{bean.props}" var="detail">
  <h:outputText value="#{detail.someProperty}"/>
</a4j:repeat>
```

The output is generated according to a collection contained in *"bean.props"* with the *"detail"* key passed to child components.

6.21.3. Dynamical creation of a component from Java code

Example:

```
import org.ajax4jsf.component.html.HtmlAjaxRepeat;
...
HtmlAjaxRepeat repeater = new HtmlAjaxRepeat ();
...
```

6.21.4. Key attributes and ways of usage

The main difference of this component from iterative components of other libraries is a special *"ajaxKeys"* attribute. This attribute defines strings that are updated after an Ajax request. As a result it becomes easier to update several child components separately without updating the whole page.

```
...
<a4j:poll interval="1000" action="#{repeater.action}" reRender="text">
  <table>
    <tbody>
      <a4j:repeat value="#{bean.props}" var="detail"
ajaxKeys="#{repeater.ajaxedRowsSet}">
        <tr>
          <td>
            <h:outputText value="detail.someProperty" id="text"/>
          </td>
        </tr>
      </a4j:repeat>
    </tbody>
  </table>
</a4j:poll>
...
```

Thus, a list with a table structure from *"bean.props"* is output.

In the above-mentioned example the component **<a4j:poll>** sends Ajax requests every second, calling the *"action"* method of the *"repeater"* bean.

Note:

The **<a4j:repeater>** component is defined as fully updated, but really updated there are only the strings which *rowKeys* includes into the set *"ajaxRowSet"* defined in the *"ajaxKeys"* attribute

The set could be defined during the action method processing using data on a model from the property *"repeater.myRepeat"*

One more benefit of this component is absence of strictly defined markup as JSF HTML DataTable and TOMAHAWK DataTable has, hence the components could be used more flexibly anywhere where it's necessary to output the results of selection from some collection.

The next example shows collection output as a plain HTML list

```
<ul>
    <a4j:repeat ...>
    <li>...</li>
    ...
    <li>...</li>
    </a4j:repeat>
</ul>
```

All other general attributes are defined according to the similar attributes of iterative components (**<h:dataTable>** or **<ui:repeat>**) and are used in the same way.

6.21.5. Relevant resources links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/repeat.jsf?c=repeat>] you can see the example of **<a4j:repeat>** usage and sources for the given example.

6.22. < a4j:status >

6.22.1. Description

The **<a4j:status>** component generates elements for displaying of the current Ajax requests status. There are two status modes: Ajax request is in process or finished.

Table 6.43. a4j : status attributes

| Attribute Name | Description |
|----------------|--|
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| dir | Direction indication for text that does not inherit directionality. Valid values are "LTR" (left-to-right) and "RTL" (right-to-left) |
| for | ID of the AjaxContainer component whose status is indicated (in the format of a javax.faces.UIComponent.findComponent() call). |
| forceId | If true, render the ID of the component in HTML code without JSF modifications. |
| id | Every component may have a unique id that is automatically created if omitted |

| Attribute Name | Description |
|-----------------|--|
| lang | Code describing the language used in the generated markup for this component |
| layout | Define visual layout of panel, can be "block" or "inline". |
| onclick | HTML: a script expression; a pointer button is clicked |
| ondblclick | HTML: a script expression; a pointer button is double-clicked |
| onkeydown | HTML: a script expression; a key is pressed down |
| onkeypress | HTML: a script expression; a key is pressed and released |
| onkeyup | HTML: a script expression; a key is released |
| onmousedown | HTML: script expression; a pointer button is pressed down |
| onmousemove | HTML: a script expression; a pointer is moved within |
| onmouseout | HTML: a script expression; a pointer is moved away |
| onmouseover | HTML: a script expression; a pointer is moved onto |
| onmouseup | HTML: script expression; a pointer button is released |
| onstart | JavaScript code, called on the start of a request. |
| onstop | JavaScript code, called on the stop of a request. |
| rendered | If "false", this component is not rendered |
| startStyle | CSS style class for the element displayed on the start of a request. |
| startStyleClass | CSS style class for the element displayed on the start of a request. |
| startText | Text for display on starting request. |
| stopStyle | CSS style for element displayed on request completion. |
| stopStyleClass | CSS style class for element displayed on request |
| stopText | Text for display on request complete. |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |

| Attribute Name | Description |
|----------------|---|
| title | Advisory title information about markup elements generated for this component |

Table 6.44. Component identification parameters

| Name | Value |
|------------------|--|
| component-type | org.ajax4jsf.Status |
| component-family | javax.faces.Panel |
| component-class | org.ajax4jsf.component.html.HtmlAjaxStatus |
| renderer-type | org.ajax4jsf.components.AjaxStatusRenderer |

6.22.2. Creating on a page

There are two ways to define elements indicating a request status :

- With *"StartText"/"StopText"* attributes:

```
<a4j:status startText="Progress" stopText="Done" for="stat1">
```

In this case, text elements for the corresponding status are generated.

- With "Start"/"Stop" facets definition:

```
<a4j:status for="stat2">
  <f:facet name="start">
    <h:graphicImage value="ajax_process.png" />
  </f:facet>
  <f:facet name="stop">
    <h:graphicImage value="ajax_stoped.png" />
  </f:facet>
</a4j:status>
```

In this case, the elements are generated for each status and correspond the facets content.

6.22.3. Dynamical creation of a component from Java code

Example:

```
import org.ajax4jsf.component.html.HtmlAjaxStatus;
...
HtmlAjaxStatus myStatus = new HtmlAjaxStatus();
...
```

6.22.4. Key attributes and ways of usage

There are two ways for the components or containers definition, which Ajax requests status is tracked by a component.

- Definition with the *"for"* attribute on the `<a4j:status>` component. Here *"for"* attribute should point at an Ajax container (`<a4j:region>`) "id", which requests are tracked by a component.
- Definition with the *"status"* attribute obtained by any RichFaces library action component. The attribute should point at the `<a4j:status>` component "id". Then this `<a4j:status>` component shows the status for the request fired from this action component.

The component creates two `` or `<div>` elements depending on attribute "layout" with content defined for each status, one of the elements (start) is initially hidden. At the beginning of an Ajax request, elements state is inversed, hence the second element is shown and the first is hidden. At the end of a response processing, elements display states return to its initial values.

Example:

```
<a4j:status startText="Started" stopText="stopped" />
```

is decoded on a page as:

```
<span id="j_id20:status.start" style="display: none">
    Started
</span>
<span id="j_id20:status.stop">
    Stopped
</span>
```

And after the generation of an Ajax response is changed to:

```
<span id="j_id20:status.start">
    Started
</span>
<span id="j_id20:status.stop" style="display: none">
    Stopped
</span>
```

There is a possibility to group a `<a4j:status>` elements content into `<div>` elements, instead of ``. To use it, just redefine the *"layout"* attribute from "inline"(default) to "block".

6.22.5. Relevant resources links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/status.jsf?c=status>] you can see the example of `<a4j:status>` usage and sources for the given example.

6.23. < a4j:support >

6.23.1. Description

The `<a4j:support>` component adds an Ajax support to any existing JSF component. It allows a component to generate asynchronous requests on the necessary event demand and with partial update of page content after a response incoming from the server.

Table 6.45. a4j : support attributes

| Attribute Name | Description |
|--------------------|---|
| action | MethodBinding pointing at the application action to be invoked, if this UIComponent is activated by the user, during the Apply Request Values or Invoke Application phase of the request processing lifecycle, depending on the value of the immediate property |
| actionExpression | The action method binding expression. |
| actionListener | MethodBinding pointing at method accepting an ActionEvent with return type void |
| ajaxSingle | if "true", submits ONLY one field/link, instead of all form controls |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| bypassUpdates | If "true", after process validations phase it skips updates of model beans on a force render response. It can be used for validating components input |
| data | Serialized (on default with JSON) data passed on the client by a developer on AJAX request. It's accessible via "data.foo" syntax |
| disableDefault | Disable default action for target event (append "return false;" to javascript) |
| event | Name of JavaScript event property (onclick, onchange, etc.) of parent component, for which we will build AJAX submission code |
| eventsQueue | Name of requests queue to avoid send next request before complete other from same event. Can be used to reduce number of requests of frequently events (key press, mouse move etc.) |
| focus | id of element to set focus after request completed on client side |
| id | Every component may have a unique id that is automatically created if omitted |
| ignoreDupResponses | Attribute allows to ignore an Ajax Response produced by a request if the newest 'similar' request is in a queue already. ignoreDupResponses="true" does not cancel the request while it is processed on the server, but just |

| Attribute Name | Description |
|-------------------|---|
| | allows to avoid unnecessary updates on the client side if the response isn't actual now |
| immediate | True means, that the default ActionListener should be executed immediately (i.e. during Apply Request Values phase of the request processing lifecycle), rather than waiting until the Invoke Application phase |
| limitToList | If "true", updates on client side ONLY elements from this 'reRender' property. If "false" (default) updates all rendered by ajax region components |
| onbeforedomupdate | JavaScript code for call before DOM has been updated on client side |
| oncomplete | JavaScript code for call after request completed on client side |
| onsubmit | JavaScript code for call before submission of ajax event |
| rendered | If "false", this component is not rendered |
| requestDelay | Attribute defines the time (in ms.) that the request will be wait in the queue before it is ready to send. When the delay time is over, the request will be sent to the server or removed if the newest 'similar' request is in a queue already |
| reRender | Id[s] (in format of call <code>UIComponent.findComponent()</code>) of components, rendered in case of AjaxRequest caused by this component. Can be single id, comma-separated list of Id's, or EL Expression with array or Collection |
| status | ID (in format of call <code>UIComponent.findComponent()</code>) of Request status component |
| timeout | Timeout (in ms) for request |

Table 6.46. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | org.ajax4jsf.Support |
| component-family | org.ajax4jsf.AjaxSupport |
| component-class | org.ajax4jsf.component.html.HtmlAjaxSupport |
| renderer-type | org.ajax4jsf.components.AjaxSupportRenderer |

6.23.2. Creating on a page

To use a component, place `<a4j:support>` as nested to the component requesting Ajax functionality and specify an event of a parent component that generates Ajax request and the components to be rerendered after a response from the server.

Example:

```
<h:inputText value="#{bean.text}">
  <a4j:support event="onkeyup" reRender="repeater" />
</h:inputText>
<h:outputText id="repeater" value="#{bean.text}" />
```

On every keyup event generated by an input field, a form is submitted on the server with the help of Ajax and on a response coming from the server, element with *"repeater"* id, founded in a DOM tree is redrawn according to a new data from the response.

6.23.3. Dynamical creation of a component from Java code

Example:

```
import org.ajax4jsf.component.html.HtmlAjaxSupport;
...
HtmlAjaxSupport mySupport = new HtmlAjaxSupport();>
```

6.23.4. Key attributes and ways of usage

A4j support addition is very similar to correspondent event redefinition of a component, i.e.

Example:

```
<h:inputText value="#{bean.text}">
  <a4j:support event="onkeyup" reRender="repeater" />
</h:inputText>
```

Is decoded on a page as:

Example:

```
<input onkeyup="A4J.AJAX.Submit( Some request parameters )" />
```

As you see from the code, the "onkeyup" event calls a utility RichFaces method that submit a form creating a special marks for a filter informing that it is an Ajax request. Thus, any supports quantity could be added to every component, the supports define component behavior on these events.

Note

The components: `<a4j:commandLink>` , `<a4j:commandButton>` , `<a4j:poll>` and others from RichFaces library are already supplied with `<a4j:support>` functionality and there is no necessity to add the support to them.

With the help of *"onsubmit"* and *"oncomplete"* attributes the component allows using JavaScript before (for request sending conditions checking) and after an Ajax response processing termination (for performance of user-defined activities on the client)

Example:

```
<h:selectOneMenu value="#{bean.text}">
  <f:selectItem itemValue="First Item " itemLabel="First Item"/>
  <f:selectItem itemValue=" Second Item " itemLabel="Second Item"/>
  <f:selectItem itemValue=" Third Item " itemLabel="Third Item"/>
  <a4j:support event="onblur" reRender="panel" onsubmit="if(!confirm('Are you sure to
change the option ?'))
    {form.reset(); return false;}" oncomplete="alert('Value succesfully stored')"/>
</h:selectOneMenu>
```

In example there is the condition checking (confirm) is used before request sending and message printing after the request processing is over.

The components allows different Ajax request managing ways for its various optimization in particular conditions such as:

- **Limitation of the submit area and updating area for the request.**

"ajaxSingle" is an attribute that allows submission on the server only component sending a request, as if the component presented on a separate form.

"limitToList" is an attribute that allows to limit areas, which are updated after the responses. Only these components defined in the *"reRender"* attribute are updated.

Example 1:

```
<h:form>
  <h:inputText value="#{person.name}">
    <a4j:support event="onkeyup" reRender="test" ajaxSingle="true"/>
  </h:inputText>
  <h:inputText value="#{person.middleName}" />
</form>
```

In this example the request contains only the input component causes the request generation, not all the components contained on a form, because of *"ajaxSingle=true"* usage.

Example 2:

```
<h:form>
  <a4j:outputPanel ajaxRendered="true">
    <h:messages/>
  </a4j:outputPanel>
  <h:inputText value="#{person.name}">
    <a4j:support event="onkeyup" reRender="test" limitToList="true"/>
  </h:inputText>
  <h:outputText value="#{person.name}" id="test"/>
</form>
```

In this example the component "h:messages" is always updated (as it capturing all Ajax requests, located in ajaxRendered <a4j:outputPanel>), except the case when a response is sent from the input component from the example. On sending this component marks that updating area is limited to the defined in it components, it means that on its usage with "limitToList"="true" the only component updated is the one with "d"="test".

- **Limitation of requests frequency and updates quantity after the responses.**

"requestDelay" is an attribute that defines a time interval in seconds minimally permissible between responses.

"eventQueue" is an attribute for naming of the queue where the next response is kept in till its processing, but if the next event comes in till this time is over, the waiting event is taken away, replacing with a new one.

"ignoreDupResponses" is an attribute that allows to disable any updates on the client after an Ajax request if another Ajax request is already sent.

"timeout" is an attribute that allows to set a time interval in millisecond to define a maximum time period of response wait time. In case of the interval interaction, a new request is sent and the previous one is canceled. Postprocessing of a response isn't performed.

Example:

```
<h:form>
  <h:inputText value="#{person.name}">
    <a4j:support event="onkeyup" reRender="test"
      requestDelay="1000" ignoreDupResponses="true" eventsQueue="myQueue" />
  </h:inputText>
  <h:outputText value="#{person.name}" id="test" />
</form>
```

This example clearly shows mentioned above attributes. If quick typing in a text field happens, every next requests sending is delayed for a second and requests quantity is reduced. The requests are kept in the queue till its the sending. Moreover, if the next request is already sent, the rerendering after the previous request is banned, and it helps to avoid unnecessary processing on the client.

6.23.5. Relevant resources links

Here [http://livedemo.exadel.com/richfaces-demo/richfaces/support.jsf?c=support] you can see the example of <a4j:support> usage and sources for the given example.

6.24. < rich:calendar >

6.24.1. Description

The <rich:calendar> component is used for creating monthly calendar elements on a page.



Figure 6.1. <rich:calendar> component

6.24.2. Key Features

- Highly customizable look and feel
- Popup representation
- Disablement support
- Smart and user-defined positioning
- Cells customization
- Macro substitution based on tool bars customization

Table 6.47. rich : calendar attributes

| Attribute Name | Description |
|-------------------|--|
| ajaxSingle | if "true", submits ONLY one field/link, instead of all form controls |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| boundaryDatesMode | Used for the dates boundaries in the list. Valid values are "inactive" (Default) dates inactive and gray colored, "scroll" boundaries work as month scrolling controls, and "select" boundaries work in the same way as "scroll" but with the date clicked selection |
| buttonClass | Style Class attribute for the popup button |
| buttonIcon | |

| Attribute Name | Description |
|---------------------------|---|
| | Defines icon for the popup button element. The attribute is ignored if the "buttonLabel" is set |
| buttonIconDisabled | Defines disabled icon for the popup button element. The attribute is ignored if the "buttonLabel" is set |
| buttonLabel | Defines label for the popup button element. If the attribute is set "buttonIcon" and "buttonIconDisabled" are ignored |
| bypassUpdates | If "true", after process validations phase it skips updates of model beans on a force render response. It can be used for validating components input |
| cellHeight | attribute to set fixed cells height |
| cellWidth | attribute to set fixed cells width |
| converter | Id of Converter to be used or reference to a Converter |
| converterMessage | A ValueExpression enabled attribute that, if present, will be used as the text of the converter message, replacing any message that comes from the converter |
| currentDate | Defines current date |
| currentDateChangeListener | MethodBinding representing an action listener method that will be notified after date selection |
| dataModel | Used to provide data for calendar elements. If data is not provided, all Data Model related functions are disabled |
| datePattern | Defines date pattern |
| dayStyleClass | Should be binded to some JS function that will provide style classes for special sets of days highlighting. |
| direction | Defines direction of the calendar popup (top-left, top-right, bottom-left, bottom-right (Default), auto) |
| disabled | If "true", rendered is disabled. In "popup" mode both controls are disabled |
| enableManualInput | If "true" calendar input will be editable and it will be possible to change the date manually. If "false" value for this attribute makes a text field "read-only", so the value can be changed only from a handle |
| eventsQueue | Name of requests queue to avoid send next request before complete other from same event. Can be used |

| Attribute Name | Description |
|--------------------|--|
| | to reduce number of requests of frequently events (key press, mouse move etc.) |
| firstWeekDay | Gets what the first day of the week is; e.g., SUNDAY in the U.S., MONDAY in France. |
| focus | id of element to set focus after request completed on client side |
| horizontalOffset | Sets the horizontal offset between button and calendar element conjunction point |
| id | Every component may have a unique id that is automatically created if omitted |
| ignoreDupResponses | Attribute allows to ignore an Ajax Response produced by a request if the newest 'similar' request is in a queue already. ignoreDupResponses="true" does not cancel the request while it is processed on the server, but just allows to avoid unnecessary updates on the client side if the response isn't actual now |
| immediate | A flag indicating that this component value must be converted and validated immediately (that is, during Apply Request Values phase), rather than waiting until a Process Validations phase |
| inputClass | Style Class attribute for the text field |
| inputSize | \${prop.xmlEncodedDescription} |
| inputStyle | Style attribute for text field |
| isDayEnabled | Should be binded to some JS function that returns day state. |
| jointPoint | Set the corner of the button for the popup to be connected with (top-left, top-right, bottom-left (Default), bottom-right, auto) |
| label | A localized user presentable name for this component. |
| limitToList | If "true", updates on client side ONLY elements from this 'reRender' property. If "false" (default) updates all rendered by ajax region components |
| locale | Used for locale definition |
| minDaysInFirstWeek | Gets what the minimal days required in the first week of the year are; e.g., if the first week is defined as one that contains the first day of the first month of a year, |

| Attribute Name | Description |
|---------------------|---|
| | this method returns 1. If the minimal days required must be a full week, this method returns 7. |
| mode | Valid values = ajax or client |
| monthLabels | Attribute that allows to customize names of the months. Should accept list with the month names |
| monthLabelsShort | Attribute that allows to customize short names of the months. Should accept list with the month names |
| onbeforedomupdate | JavaScript code for call before DOM has been updated on client side |
| onchanged | onChanged event handler |
| oncollapse | onCollapse event handler |
| oncomplete | JavaScript code for call after request completed on client side |
| oncurrentdateselect | onCurrentDateSelect event handler |
| oncurrentdateselect | onCurrentDateSelected event handler |
| ondatemouseout | onDateMouseOut event handler |
| ondatemouseover | onDateMouseOver event handler |
| ondateselect | onDateSelect event handler |
| ondateselected | onDateSelected event handler |
| onexpand | onExpand event handler |
| oninputblur | input onBlur event handler |
| oninputchange | input onChange event handler |
| oninputclick | input onClick event handler |
| oninputfocus | input onFocus event handler |
| oninputkeydown | input onKeyDown event handler |
| oninputkeypress | input onKeyPress event handler |
| oninputkeyup | input onKeyUp event handler |
| oninputselect | input onSelect event handler |
| ontimeselect | onTimeSelect event handler |
| ontimeselect | onTimeSelected event handler |

| Attribute Name | Description |
|-----------------------|---|
| popup | If "true" calendar will be rendered initially as hidden with additional elements for calling as popup |
| preloadDateRangeBegin | Define the initial range of date which will be loaded to client from dataModel under rendering |
| preloadDateRangeEnd | Defines the last range of date which will be loaded to client from dataModel under rendering |
| rendered | If "false", this component is not rendered |
| requestDelay | Attribute defines the time (in ms.) that the request will be wait in the queue before it is ready to send. When the delay time is over, the request will be sent to the server or removed if the newest 'similar' request is in a queue already |
| required | If "true", this component is checked for non-empty input |
| requiredMessage | A ValueExpression enabled attribute that, if present, will be used as the text of the validation message for the "required" facility, if the "required" facility is used |
| reRender | Id[s] (in format of call <code>UIComponent.findComponent()</code>) of components, rendered in case of AjaxRequest caused by this component. Can be single id, comma-separated list of Id's, or EL Expression with array or Collection |
| showApplyButton | If false ApplyButton should not be shown |
| showFooter | If false Calendar's footer should not be shown |
| showHeader | If false Calendar's header should not be shown |
| showInput | "false" value for this attribute makes text field invisible. If "true" - input field will be shown |
| showWeekDaysBar | If false this bar should not be shown |
| showWeeksBar | If false this bar should not be shown |
| status | ID (in format of call <code>UIComponent.findComponent()</code>) of Request status component |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |

| Attribute Name | Description |
|---------------------|---|
| timeout | Response waiting time on a particular request. If a response is not received during this time, the request is aborted |
| timeZone | Used for current date calculations |
| todayControlMode | Possible values are "scroll", "select", "hidden" |
| toolTipMode | Used to specify mode to load tooltips. Valid values are "none", "single" and "batch" |
| validator | MethodBinding pointing at a method that is called during Process Validations phase of the request processing lifecycle, to validate the current value of this component |
| validatorMessage | A ValueExpression enabled attribute that, if present, will be used as the text of the validator message, replacing any message that comes from the validator |
| value | The initial value to set when rendered for the first time |
| valueChangeListener | Listener for value changes |
| verticalOffset | Sets the vertical offset between button and calendar element conjunction point |
| weekDayLabels | List of the day names displays on the days bar in the following way "Sun, Mon, Tue, Wed, " |
| weekDayLabelsShort | Attribute that allows to customize short names of the weeks. Should accept list with the weeks names. |
| zindex | Attribute is similar to the standard HTML attribute and can specify window placement relative to the content |

Table 6.48. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | org.richfaces.Calendar |
| component-class | org.richfaces.component.html.HtmlCalendar |
| component-family | org.richfaces.Calendar |
| renderer-type | org.richfaces.CalendarRenderer |
| tag-class | org.richfaces.taglib.CalendarTag |

6.24.3. Creating the Component with a Page Tag

To create the simplest variant on a page use the following syntax:

Example:

```
...
    <rich:calendar popup="false"/>
...
```

6.24.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlCalendar;
...
HtmlCalendar myCalendar = new HtmlCalendar();
...
```

6.24.5. Details of Usage

The *"popup"* attribute defines calendar representation mode on a page. If it's "true" the calendar is represented on a page as an input field and a button. Clicking on the button calls the calendar popup as it's shown on the picture below.

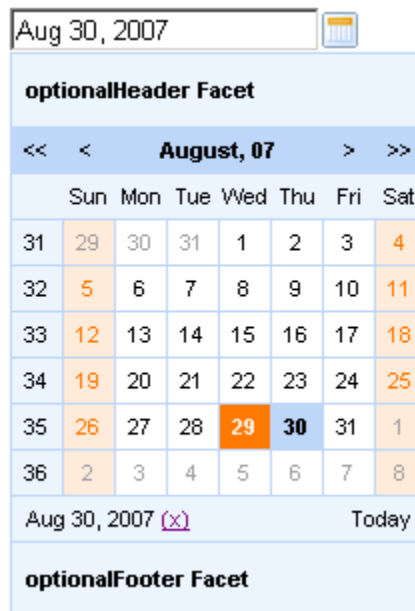


Figure 6.2. Using the *"popup"* attribute

Usage *"currentDate"* attribute isn't available in the popup mode.

The `<rich:calendar>` component ables to render pages of days in two modes. Switching mode could be defined with the *"mode"* attribute with two possible parameters: ajax and client. Default value is set to the "client".

- Ajax

Calendar requests portions of data from Data Model for a page rendering. If *"dataModel"* attribute has *"null"* value, data requests are not sent. In this case the *"ajax"* mode is equal to the *"client"*.

- Client

Calendar loads an initial portion of data in a specified range and use this data to render months. Additional data requests are not sent.

Note:

"preloadDateRangeBegin" and *"preloadDateRangeEnd"* attributes was designed only for the *"client"* mode to load some data initially.

"ondataselect" attribute is used to define an event that is triggered before date selection.

"ondateselected" attribute is used to define an event that is triggered after date selection.

For example, to fire some event after date selection you should use **<a4j:support>** . And it should be bound to *"ondateselected"* event as it's shown in the example below:

```
...
<rich:calendar id="date" value="#{bean.dateTest}">
  <a4j:support event="ondateselected" reRender="mainTable"/>

</rich:calendar>
...
```

"ondataselect" could be used for possibility of date selection canceling. See an example below:

```
...
<rich:calendar id="date" value="#{bean.dateTest}" ondataselect="if (!confirm('Are
you sure to change date?')){return false;}"/>
...
```

How to use these attributes see also on the RichFaces Users Forum [<http://www.jboss.com/index.html?module=bb&op=viewtopic&p=4092275#4092275>].

There are three button-related attributes:

- *"buttonLabel"* defines a label for the button. If the attribute is set *"buttonIcon"* and *"buttonIconDisabled"* are ignored
- *"buttonIcon"* defines an icon for the button
- *"buttonIconDisabled"* defines an icon for the disabled state of the button

The *"direction"* and *"jointPoint"* attributes are used for defining aspects of calendar appearance.

The possible values for the *"direction"* are:

- top-left - a calendar drops to the top and left
- top-right - a calendar drops to the top and right
- bottom-left - a calendar drops to the bottom and left
- bottom-right - a calendar drops to the bottom and right
- auto - smart positioning activation

By default, the *"direction"* attribute is set to "bottom-right".

The possible values for the *"jointPoint"* are:

- top-left - a calendar docked to the top-left point of the button element
- top-right - a calendar docked to the top-right point of the button element
- bottom-left - a calendar docked to the bottom-left point of the button element
- bottom-right - a calendar docked to the bottom-right point of the button element
- auto - smart positioning activation

By default, the *"jointPoint"* attribute is set to "bottom-left".

The **<rich:calendar>** component allows to use *"header"* , *"footer"* , *"optionalHeader"* , *"optionalFooter"* facets. The following elements are available in these facets: {currentMonthControl}, {nextMonthControl}, {nextYearControl}, {previousYearControl}, {previousMonthControl}, {todayControl}, {selectedDateControl}. These elements could be used for labels output.

Also you can use *"weekNumber"* facet with available {weekNumber}, {elementId} elements and *"weekDay"* facet with {weekDayLabel}, {weekDayLabelShort}, {weekDayNumber}, {isWeekend}, {elementId} elements. {weekNumber}, {weekDayLabel}, {weekDayLabelShort}, {weekDayNumber} elements could be used for labels output, {isWeekend}, {elementId} - for additional processing in JavaScript code.

These elements are shown on the picture below.

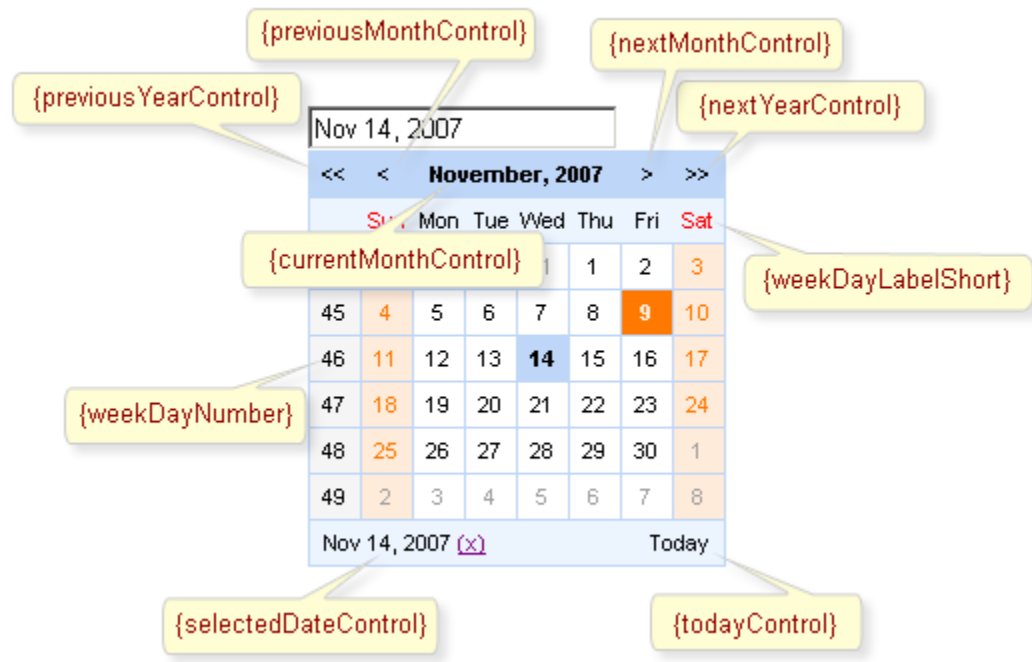


Figure 6.3. Available elements

Simple example of usage is placed below.

Example:

```
...
<!--Styles for cells -->
<style>
    .width100{
        width:100%;
    }
    .talign{
        text-align:center;
    }
</style>
...
```

```
...
<rich:calendar id="myCalendar" popup="true" locale="#{calendarBean.locale}"
    value="#{bean.date}"
    preloadRangeBegin="#{bean.date}"
    preloadRangeEnd="#{bean.date}" selectedDate="#{bean.date}" cellWidth="40px"
    cellHeight="40px">

<!-- Customization with usage of facets and accessible elements -->

    <f:facet name="header">
        <h:panelGrid columns="2" width="100%" columnClasses="width100, fake">
            <h:outputText value="{selectedDateControl}" />
            <h:outputText value="{todayControl}" style="font-weight:bold;
text-align:left"/>
        </h:panelGrid>
    </f:facet>
```



```

<f:facet name="weekDay">
    <h:panelGroup style="width:60px; overflow:hidden;" layout="block">
        <h:outputText value="{weekDayLabelShort}"/>
    </h:panelGroup>
</f:facet>
<f:facet name="weekNumber">
    <h:panelGroup>
        <h:outputText value="{weekNumber}" style="color:red"/>
    </h:panelGroup>
</f:facet>
<f:facet name="footer">
    <h:panelGrid columns="3" width="100%" columnClasses="fake, width100 talign">

        <h:outputText value="{previousMonthControl}" style="font-weight:bold;"/>
        <h:outputText value="{currentMonthControl}" style="font-weight:bold;"/>
        <h:outputText value="{nextMonthControl}" style="font-weight:bold;"/>
    </h:panelGrid>
</f:facet>
<h:outputText value="{day}"></h:outputText>
</rich:calendar>
...

```

This is a result:

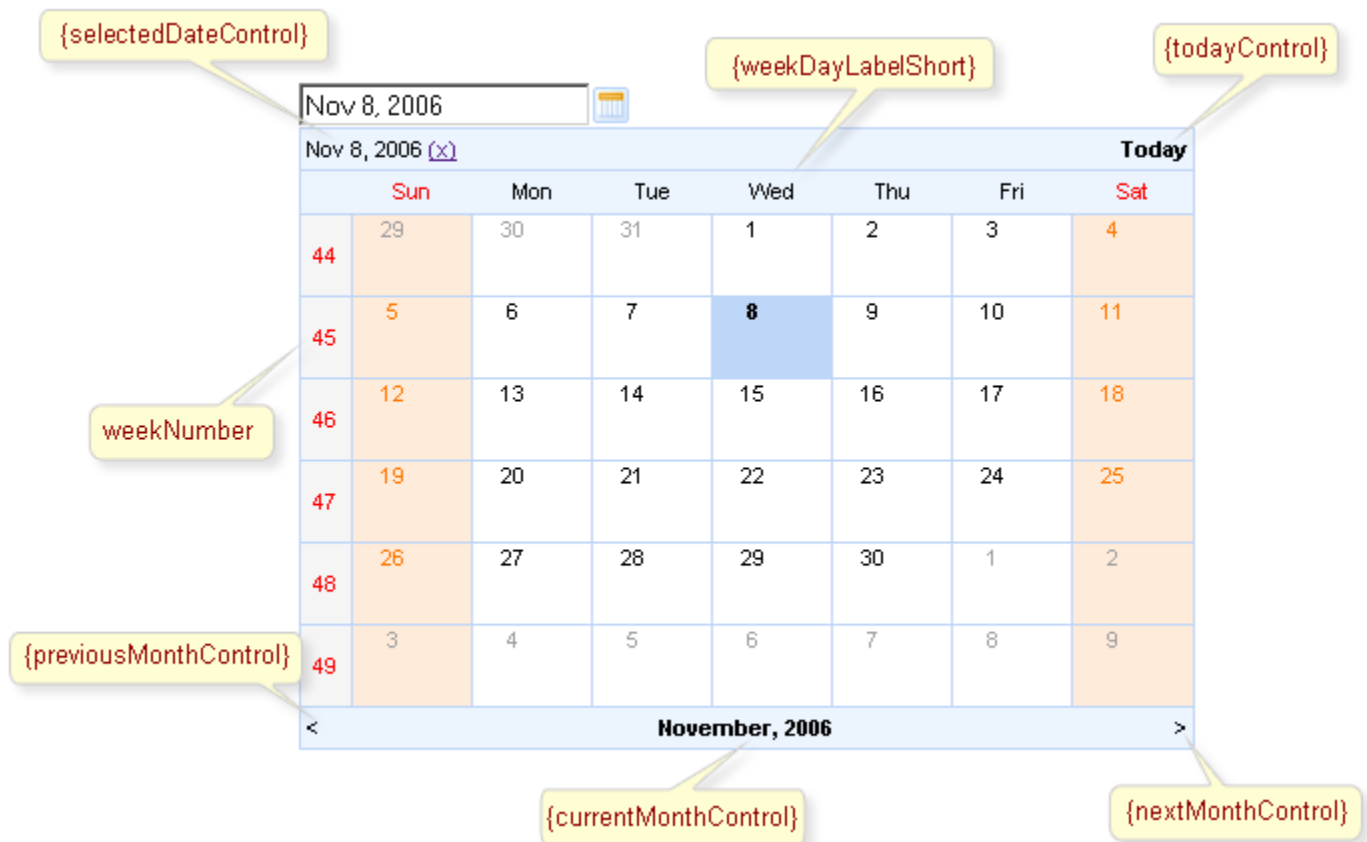


Figure 6.4. Facets usage

As it's shown on the picture above {selectedDateControl}, {todayControl} elements are placed in the "header" facet, {previousMonthControl}, {currentMonthControl}, {nextMonthControl} - in the "footer"

facet, {weekDayLabelShort} - in the "*weekDay*" facet, {nextYearControl}, {previousYearControl} are absent. Numbers of weeks are red colored.

It is possible to show and manage date. Except scrolling controls you can use quick month and year selection feature. It's necessary to click on its field, i.e. current month control, and choose required month and year.

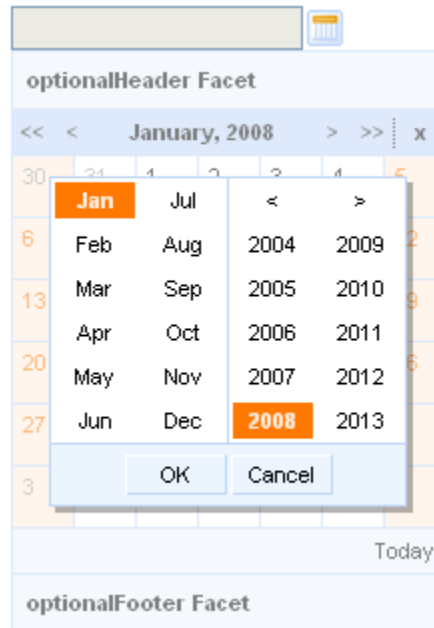


Figure 6.5. Quick month and year selection

Also the `<rich:calendar>` component allows to show and manage time. It's necessary to define time in a pattern (for example, it could be defined as "d/M/yy HH:mm"). Then after you choose some data in the calendar, it becomes possible to manage time for this date. For time editing it's necessary to click on its field (see a picture below). To clean the field click on the "Clean".

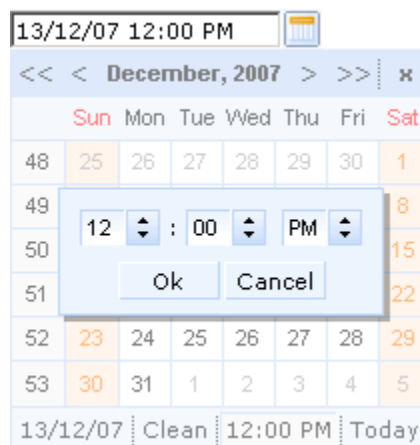


Figure 6.6. Timing

It's possible to program events for calendar from JavaScript code. A simplest example of usage JavaScript API is placed below:

Example:

```

...
    <a4j:form id="form">
        <rich:calendar popup="true" rendered="#{!bean.check}"
value="#{bean.date}" id="c"/>
        <!--The link which is used to call a calendar popup
externally-->
        <a onclick="$('form:c').component.Expand()" href="#">Show</a>
    </a4j:form>
...

```

Also the discussion about this problem can be found on the RichFaces Users Forum [<http://www.jboss.com/index.html?module=bb&op=viewtopic&p=4078301#4078301>].

The **<rich:calendar>** component provides the possibility to use a special Data Model to define data for element rendering. Data Model includes two major interfaces:

- CalendarDataModel
- CalendarDataModelItem

CalendarDataModel provides the following function:

- CalendarDataModelItem[] getData(Date[]);

This method is called when it's necessary to represent the next block of CalendarDataItems. It happens during navigation to the next (previous) month or in any other case when calendar renders. This method is called in *"Ajax"* mode when the calendar renders a new page.

CalendarDataModelItem provides the following function:

- Date getDate() - returns date from the item. Default implementation returns date.
- Boolean isEnabled() - returns *"true"* if date is *"selectable"* on the calendar. Default implementation returns *"true"*.
- String getStyleClass() - returns string appended to the style class for the date span. For example it could be *"relevant holyday"*. It means that the class could be defined like the *"rich-cal-day-relevant-holyday"* one. Default implementation returns empty string.
- Object getData() - returns any additional payload that must be JSON-serializable object. It could be used in the custom date representation on the calendar (inside the custom facet).

6.24.6. JavaScript API

Table 6.49. JavaScript API

| Function | Description |
|------------------|---|
| selectDate(date) | Selects the date specified. If the date isn't in current month - performs request to select |

| Function | Description |
|----------------------|--|
| isDateEnabled(date) | Checks if given date is selectable |
| enableDate(date) | Enables date cell control on the calendar |
| disableDate(date) | Disables date cell control on the calendar |
| enableDates(date[]) | Enables dates cell controls set on the calendar |
| disableDates(date[]) | Disables dates cell controls set on the calendar |
| nextMonth() | Navigates to next month |
| nextYear() | Navigates to next year |
| prevMonth() | Navigates to previous month |
| prevYear() | Navigates to previous year |
| today() | Selects today date |
| getSelectedDate() | Returns currently selected date |
| Object getData() | Returns additional data for the date |
| enable() | Enables calendar |
| disable() | Disables calendar |
| getCurrentMonth() | Returns number of the month currently being viewed |
| getCurrentYear() | Returns number of the year currently being viewed |
| Collapse() | Collapses calendar element |
| Expand() | Expands calendar element |

6.24.7. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

There are two ways to redefine the appearance of all `<rich:calendar>` components at once:

- Redefine the corresponding skin parameters
- Add to your style sheets *style classes* used by a `<rich:calendar>` component

6.24.8. Skin Parameters Redefinition

Table 6.50. Skin parameters redefinition for a popup element

| Skin parameters | CSS properties |
|------------------|----------------|
| panelBorderColor | border-color |

Table 6.51. Skin parameters redefinition for headers (header, optional header)

| Skin parameters | CSS properties |
|---------------------------|---------------------|
| panelBorderColor | border-bottom-color |
| additionalBackgroundColor | background-color |
| generalSizeFont | font-size |
| generalFamilyFont | font-family |

Table 6.52. Skin parameters redefinition for footers (footer, optional footer) and names of working days

| Skin parameters | CSS properties |
|---------------------------|--------------------|
| panelBorderColor | border-top-color |
| panelBorderColor | border-right-color |
| additionalBackgroundColor | background |
| generalSizeFont | font-size |
| generalFamilyFont | font-family |

Table 6.53. Skin parameters redefinition for weeks numbers

| Skin parameters | CSS properties |
|-----------------------------|---------------------|
| panelBorderColor | border-bottom-color |
| panelBorderColor | border-right-color |
| additionalBackgroundColor | background |
| calendarWeekBackgroundColor | background-color |
| generalSizeFont | font-size |
| generalFamilyFont | font-family |

Table 6.54. Skin parameters redefinition for a toolBar and names of months

| Skin parameters | CSS properties |
|-----------------------|------------------|
| headerBackgroundColor | background-color |
| headerSizeFont | font-size |
| headerFamilyFont | font-family |
| headerWeightFont | font-weight |
| headerTextColor | color |

Table 6.55. Skin parameters redefinition for cells with days

| Skin parameters | CSS properties |
|------------------------|---------------------|
| panelBorderColor | border-bottom-color |
| panelBorderColor | border-right-color |
| generalBackgroundColor | background-color |
| generalSizeFont | font-size |
| generalFamilyFont | font-family |

Table 6.56. Skin parameters redefinition for holiday

| Skin parameters | CSS properties |
|---------------------------------|------------------|
| calendarHolidaysBackgroundColor | background-color |
| calendarHolidaysTextColor | color |

Table 6.57. Skin parameters redefinition for cell with a current date

| Skin parameters | CSS properties |
|--------------------------------|------------------|
| calendarCurrentBackgroundColor | background-color |
| calendarCurrentTextColor | color |

Table 6.58. Skin parameters redefinition for a selected day

| Skin parameters | CSS properties |
|-----------------------|------------------|
| headerBackgroundColor | background-color |
| headerTextColor | color |
| headerWeightFont | font-weight |

Table 6.59. Skin parameters redefinition for a popup element during quick month and year selection

| Skin parameters | CSS properties |
|----------------------|----------------|
| tableBackgroundColor | background |
| panelBorderColor | border-color |

Table 6.60. Skin parameters redefinition for a shadow

| Skin parameters | CSS properties |
|-----------------------|------------------|
| shadowBackgroundColor | background-color |

Table 6.61. Skin parameters redefinition for a selected month and year

| Skin parameters | CSS properties |
|--------------------------------|------------------|
| calendarCurrentBackgroundColor | background-color |
| calendarCurrentTextColor | color |

Table 6.62. Skin parameters redefinition for a hovered month and year

| Skin parameters | CSS properties |
|-----------------------------|----------------|
| panelBorderColor | border-color |
| calendarSpecBackgroundColor | background |

Table 6.63. Skin parameters redefinition for a month items near split line

| Skin parameters | CSS properties |
|------------------|--------------------|
| panelBorderColor | border-right-color |

Table 6.64. Skin parameters redefinition for a hovered toolbar items

| Skin parameters | CSS properties |
|-----------------------------|---------------------|
| calendarWeekBackgroundColor | background-color |
| generalTextColor | color |
| tableBackgroundColor | border-color |
| panelBorderColor | border-right-color |
| panelBorderColor | border-bottom-color |

Table 6.65. Skin parameters redefinition for a pressed toolbar items

| Skin parameters | CSS properties |
|----------------------|---------------------|
| panelBorderColor | border-color |
| tableBackgroundColor | border-right-color |
| tableBackgroundColor | border-bottom-color |

Table 6.66. Skin parameters redefinition for "ok" and "cancel" buttons

| Skin parameters | CSS properties |
|---------------------------|------------------|
| additionalBackgroundColor | background |
| panelBorderColor | border-top-color |

Table 6.67. Skin parameters redefinition for a popup element during time selection

| Skin parameters | CSS properties |
|---------------------------|----------------|
| additionalBackgroundColor | background |
| panelBorderColor | border-color |

Table 6.68. Skin parameters redefinition for a wrapper <td> element for an input field

| Skin parameters | CSS properties |
|------------------------|---------------------|
| controlBackgroundColor | background-color |
| panelBorderColor | border-color |
| subBorderColor | border-bottom-color |
| subBorderColor | border-right-color |

Table 6.69. Skin parameters redefinition for an input field

| Skin parameters | CSS properties |
|------------------|----------------|
| buttonSizeFont | font-size |
| buttonFamilyFont | font-family |

Table 6.70. Skin parameters redefinition for a wrapper <td> element for spinner buttons

| Skin parameters | CSS properties |
|-----------------------|------------------|
| headerBackgroundColor | background-color |
| headerBackgroundColor | border-color |

6.24.9. Definition of Custom Style Classes

On the screenshot there are classes names that define styles for component elements.

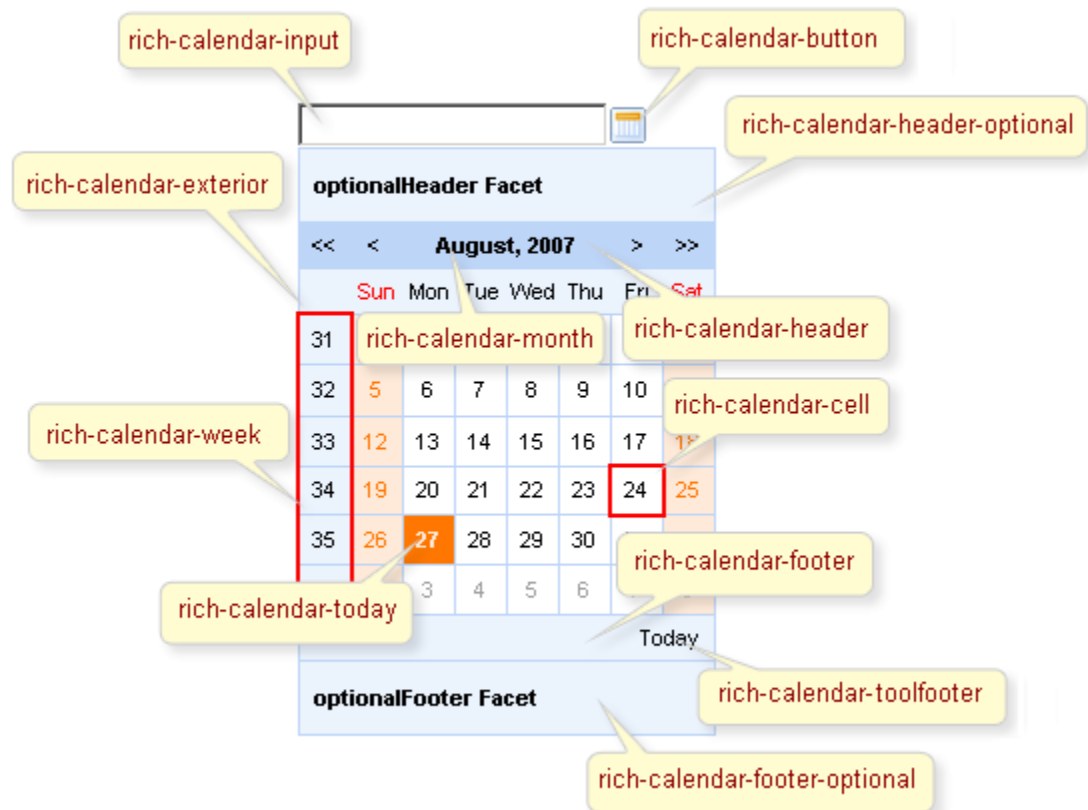


Figure 6.7. Style classes

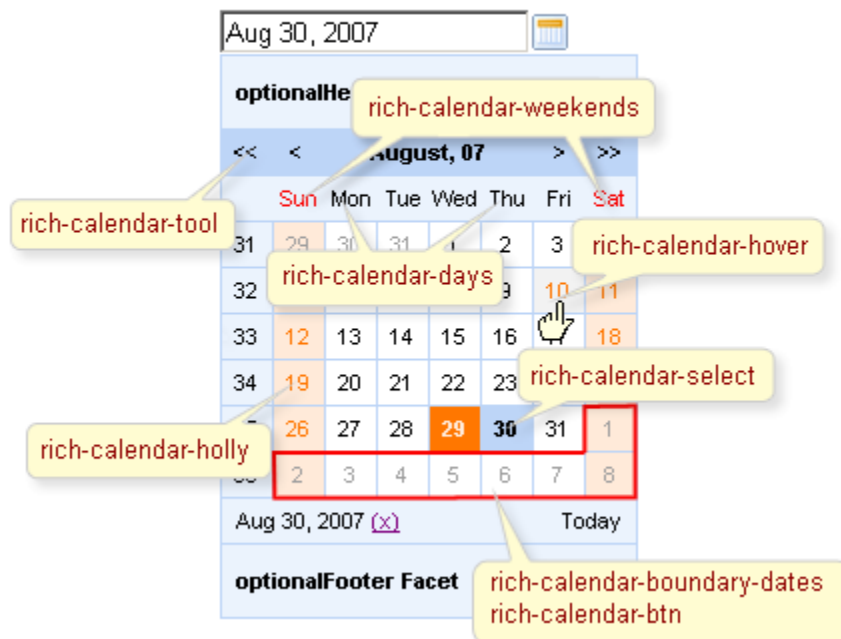


Figure 6.8. Style classes

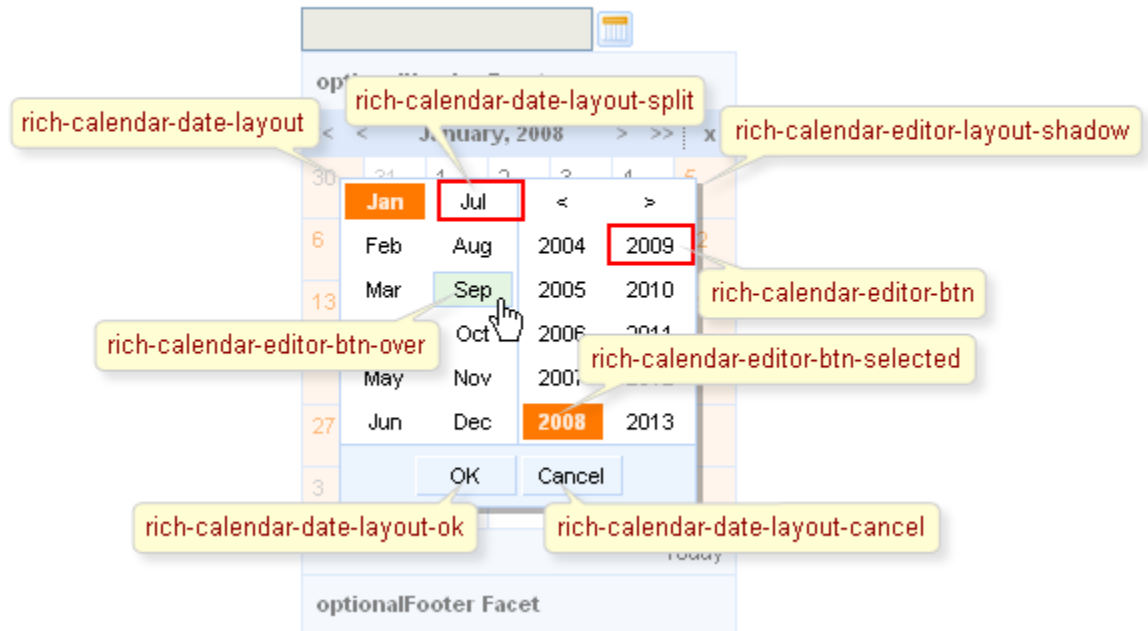


Figure 6.9. Style classes

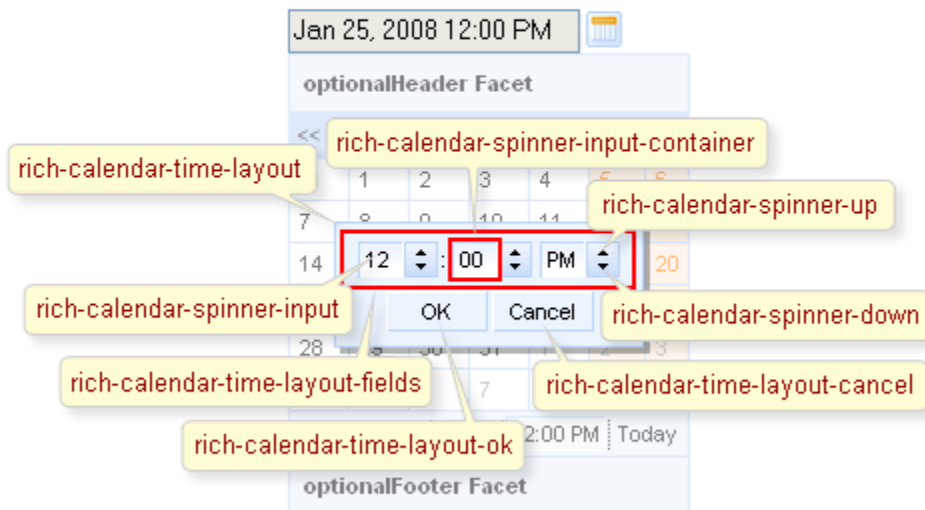


Figure 6.10. Style classes

Table 6.71. Classes names that define an input field and a button appearance

| Class name | Description |
|---------------------|-----------------------------------|
| rich-calendar-input | Defines styles for an input field |

| Class name | Description |
|----------------------|-----------------------------------|
| rich-calendar-button | Defines styles for a popup button |

Table 6.72. Classes names that define a days appearance

| Class name | Description |
|------------------------|--|
| rich-calendar-days | Defines styles for names of working days in a header |
| rich-calendar-weekends | Defines styles for names of weekend in a header |
| rich-calendar-week | Defines styles for weeks numbers |
| rich-calendar-today | Defines styles for cell with a current date |
| rich-calendar-cell | Defines styles for cells with days |
| rich-calendar-holly | Defines styles for holiday |
| rich-calendar-select | Defines styles for a selected day |
| rich-calendar-hover | Defines styles for a hovered day |

Table 6.73. Classes names that define a popup element

| Class name | Description |
|-------------------------------|--|
| rich-calendar-popup | Defines styles for a popup element |
| rich-calendar-exterior | Defines styles for a popup element exterior |
| rich-calendar-tool | Defines styles for toolbars |
| rich-calendar-month | Defines styles for names of months |
| rich-calendar-header-optional | Defines styles for an optional header |
| rich-calendar-footer-optional | Defines styles for an optional footer |
| rich-calendar-header | Defines styles for a header |
| rich-calendar-footer | Defines styles for a footer |
| rich-calendar-boundary-dates | Defines styles for an active boundary button |
| rich-calendar-btn | Defines styles for an inactive boundary date |
| rich-calendar-toolfooter | Defines styles for a today control date |

Table 6.74. Classes names that define a popup element during quick month and year selection

| Class name | Description |
|---------------------------|--|
| rich-calendar-date-layout | Defines styles for a popup element during quick year selection |

| Class name | Description |
|------------------------------------|---|
| rich-calendar-editor-layout-shadow | Defines styles for a shadow |
| rich-calendar-editor-btn | Defines styles for an inactive boundary date |
| rich-calendar-date-layout-split | Defines styles for a wrapper <td> element for month items near split line |
| rich-calendar-editor-btn-selected | Defines styles for an selected boundary date |
| rich-calendar-editor-btn-over | Defines styles for a boundary date when pointer was moved onto |
| rich-calendar-editor-tool-over | Defines styles for a hovered toolbar items |
| rich-calendar-editor-tool-press | Defines styles for a pressed toolbar items |
| rich-calendar-date-layout-ok | Defines styles for a "ok" button |
| rich-calendar-date-layout-cancel | Defines styles for a "cancel" button |

Table 6.75. Classes names that define a popup element during time selection

| Class name | Description |
|---------------------------------------|--|
| rich-calendar-time-layout | Defines styles for a popup element during time selection |
| rich-calendar-editor-layout-shadow | Defines styles for a shadow |
| rich-calendar-time-layout-fields | Defines styles for a wrapper <td> element for input fields and buttons |
| rich-calendar-spinner-input-container | Defines styles for a wrapper <td> element for an input field |
| rich-calendar-spinner-input | Defines styles for an input field |
| rich-calendar-spinner-buttons | Defines styles for a wrapper <td> element for spinner buttons |
| rich-calendar-spinner-up | Defines styles for a "up" button |
| rich-calendar-spinner-down | Defines styles for a "down" button |
| rich-calendar-time-layout-ok | Defines styles for a "ok" button |
| rich-calendar-time-layout-cancel | Defines styles for a "cancel" button |

In order to redefine styles for all **<rich:calendar>** components on a page using CSS, it's enough to create classes with the same names and define necessary properties in them.

To change styles of particular **<rich:calendar>** components, define your own style classes in the corresponding **<rich:calendar>** attributes.

CSS code piece used on a page:

Example:

```
...
.rich-calendar-today {
    background-color: #00FA9A;
}
.myClass {
    font-size: xx-large;
}
...
```

The component is defined in the following way:

Example:

```
<rich:calendar ... inputClass="myClass" .../>
```

6.24.10. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/calendar.jsf?c=calendar>] you can see the example of **<rich:calendar>** usage and sources for the given example.

How to use JavaScript API see on the RichFaces Users Forum [<http://www.jboss.com/index.html?module=bb&op=viewtopic&p=4078301#4078301>].

6.25. < rich:componentControl >

6.25.1. Description

The **<rich:componentControl>** allows to call JS API functions on components after defined events.

Table 6.76. rich : componentControl attributes

| Attribute Name | Description |
|----------------|--|
| attachTiming | Defines an Id of a component or HTML element listened by the control component |
| attachTo | Client identifier of the component or id of the existing DOM element that is a source for given event. If attachTo is defined, the event is attached on the client according to the AttachTiming attribute. If attachTo is not defined, the event is attached on the server to the closest in the component tree parent component. |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| disableDefault | If "true", it is used to avoid a problem with form submit and modalPanel showing |

| Attribute Name | Description |
|----------------|---|
| event | The Event that is used to trigger the operation on the target component |
| for | Client identifier of the target component. |
| id | Every component may have a unique id that is automatically created if omitted |
| name | The optional name of the function that might be used to trigger the operation on the target component |
| operation | The function of Javascript API that will be invoked. The API method is attached to the 'component' property of the root DOM element that represents the target component. The function has two parameters - event and params. See: 'params' attribute for details. |
| params | The set of parameters passed to the function of Javascript API that will be invoked. The JSON syntax is used to define the parameters, but without open and closed curve bracket. As an alternative, the set of f:param can be used to define the parameters passed to the API function. If both way are used to define the parameters, both set are concatenated. if names are equals, the f:param has a priority. |
| rendered | If "false", this component is not rendered |

Table 6.77. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | org.richfaces.ComponentControl |
| component-class | org.richfaces.component.html.HtmlComponentControl |
| component-family | org.richfaces.ComponentControl |
| renderer-type | org.richfaces.ComponentControlRenderer |
| tag-class | org.richfaces.taglib.ComponentControlTag |

6.25.2. Creating the Component with a Page Tag

To create the simplest variant on a page use the following syntax:

Example:

```
...
<rich:componentControl attachTo="doExpandCalendarID" for="ccCalendarID"
event="onclick" operation="Expand" />
```

...

6.25.3. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlComponentControl;
...
HtmlComponentControl myCalendar = new HtmlComponentControl();
...
```

6.25.4. Details of Usage

There are three ways to use the `<rich:componentControl>` to another component.

- Defining a name of a function that is generated (definition is similar to a definition of `<a4j:jsFunction>`). An "event" argument is passed to this function.

An example is placed below:

```
...
<rich:componentControl name="ffunction" for="comp_ID" operation="show" />
...
```

According to this code a function with name "call" is generated. It is used in JavaScript code to trigger an operation on the target component with defined id="comp_ID".

The generated function is shown below:

```
function ffunction (event) {
}
```

- Attaching to a parent component (usage is similar to `<a4j:support>` component).

An example is placed below:

```
...
<rich:modalPanel id="ccModalPanelID" onshow="alert(event.parameters.show)"
onhide="alert(event.parameters.hide)">
...
</rich:modalPanel>
<h:commandButton value="Show Modal Panel">
    <rich:componentControl for="ccModalPanelID" event="onclick"
disableDefault="true" operation="show">
        <f:param name="show" value="componentControl work(show)" />
    </rich:componentControl/>
</h:commandButton>
...
```

In the example the "for" attribute contains value of an Id of `<rich:modalPanel>` component. The "operation" attribute contains a name of JavaScript API function. An "event" attribute is used to trigger an operation defined with the "operation" attribute. A set of parameters is defined with

<f:param> . As an alternative, the *"params"* attribute can be used. Thus, one of main features is that **<rich:componentControl>** allows to transfer parameters. The *"disableDefault"* attribute with *"true"* value is used instead of *onclick="return false;"* attribute for **<h:commandButton>** to avoid a problem with form submit and modalPanel showing.

- Attaching with *"attachTo"* attribute.

An example is placed below:

```
...
<rich:calendar popup="#{componentControl.calendarPopup}" id="ccCalendarID" />
...
<f:verbatim>
    <a href="#" id="doExpandCalendarID">Calendar (nextYear)</a>
</f:verbatim>
<rich:componentControl attachTo="doExpandCalendarID" for="ccCalendarID"
event="onclick" disableDefault="true" operation="nextYear" />
...
```

In the example the *"attachTo"* attribute contains a value of an Id of **<a>** tag. The *"for"* attribute contains value of an Id of **<rich:calendar>** component. The *"operation"* attribute contains a name of JavaScript API function. Thus, clicking on the link represents the next year on the calendar.

6.25.5. Look-and-Feel Customization

<rich:componentControl> has no skin parameters and custom style classes, as the component isn't visual.

6.25.6. Relevant Resources Links

Here [http://livedemo.exadel.com/richfaces-demo/richfaces/componentControl.jsf?c=componentControl] you can see an example of **<rich:componentControl>** usage and sources for the given example.

6.26. < rich:contextMenu >

6.26.1. Description

The **<rich:contextMenu>** component is used for creation multileveled context menus that are activated after a user defines an event (onmouseover, onclick, etc.) on any element on the page.



Figure 6.11. **<rich:contextMenu>** component

6.26.2. Key Features

- Highly customizable look and feel
- "oncontextmenu" event support
- Disablement support
- Pop-up appearance event customization
- Usage of shared instance of a menu on a page

Table 6.78. rich : contextMenu attributes

| Attribute Name | Description |
|--------------------|---|
| attached | If the value of the "attached" attribute is true, component is attached to parent component |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| disableDefaultMenu | Forbids default handling for adjusted event |
| disabledItemClass | Space-separated list of CSS style class(es) that are applied to disabled item of this component |
| disabledItemStyle | CSS style(s) is/are to be applied to disabled item when this component is rendered. |
| event | Defines an event on the parent element that triggers the menu's appearance |
| hideDelay | Delay between losing focus and menu closing. |
| id | Every component may have a unique id that is automatically created if omitted |
| itemClass | Space-separated list of CSS style class(es) that are applied to item of this component |
| itemStyle | CSS style(s) is/are to be applied to item when this component is rendered. |
| oncollapse | Event must occurs on menu closure |
| onexpand | Event must occurs on menu opening |
| ongroupactivate | HTML: script expression; some group was activated |
| onitemselect | HTML: script expression; some item was selected |
| onmousemove | HTML: script expression; a pointer was moved within |
| onmouseout | HTML: script expression; a pointer was moved away |

| Attribute Name | Description |
|-----------------|---|
| onmouseover | HTML: script expression; a pointer was moved onto |
| popupWidth | Set minimal width for the all of the lists that will appear |
| rendered | If "false", this component is not rendered |
| selectItemClass | Space-separated list of CSS style class(es) that are be applied to selected item of this component. |
| selectItemStyle | CSS style(s) is/are to be applied to selected item when this component is rendered. |
| showDelay | Delay between event and menu showing. |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| submitMode | Set the submission mode for all menu items of the menu except ones where this attribute redefined (ajax, server(Default), none) |

Table 6.79. Component identification parameters

| Name | Value |
|------------------|--|
| component-type | org.richfaces.ContextMenu |
| component-class | org.richfaces.component.html.ContextMenu |
| component-family | org.richfaces.ContextMenu |
| renderer-type | org.richfaces.DropDownMenuRenderer |
| tag-class | org.richfaces.taglib.ContextMenuTagHandler |

6.26.3. Creating the Component with a Page Tag

Here is a simple example as it could be used on a page:

Example:

```
...
<rich:contextMenu event="oncontextmenu" attached="true">
...
```

6.26.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlContextMenu;
```

```
...
HtmlContextMenu myContextMenu = new HtmlContextMenu();
...
```

6.26.5. Details of Usage

<rich:contextMenu> is a support-like component. Context menu itself is an invisible panel that appears after a particular client side event (onmouseover, onclick, etc) occurred on a parent component. The event is defined with an *"event"* attribute. The component uses *"oncontextmenu"* event by default to call a context menu by clicking on the right mouse button.

<rich:menuGroup>, **<rich:menuItem>** and **<rich:menuSeparator>** components are used as nested elements for **<rich:contextMenu>** in the same way as for **<rich:dropDownMenu>**.

If a value of the *"attached"* attribute is defined as "true", the component is attached to the parent component. An example is placed below.

Example:

```
...
    <h:panelGrid columns="1" columnClasses="cent">
        <h:panelGroup id="picture">
            <h:graphicImage value="/richfaces/jquery/images/pic1.png"
id="pic"/>
            <rich:contextMenu event="oncontextmenu" attached="true"
submitMode="none">
                <rich:menuItem value="Zoom In" onclick="enlarge();"
id="zin"></rich:menuItem>
                <rich:menuItem value="Zoom Out" onclick="decrease();"
id="zout"></rich:menuItem>
            </rich:contextMenu>
        </h:panelGroup>
    </h:panelGrid>
...
```

The "enlarge()" and "decrease()" functions definition is placed below.

```
...
    <script type="text/javascript">
        function enlarge(){
            document.getElementById('pic').width=document.getElementById('pic').width*1.1;
            document.getElementById('pic').height=document.getElementById('pic').height*1.1;
        }
        function decrease(){
            document.getElementById('pic').width=document.getElementById('pic').width*0.9;
            document.getElementById('pic').height=document.getElementById('pic').height*0.9;
        }
    </script>
...
```

In the example a picture zooming possibility with `<rich:contextMenu>` component usage was shown. The picture is placed on the `<h:panelGroup>` component. The `<rich:contextMenu>` component is defined as nested to `<h:panelGroup>` one and has a value of the `"attached"` attribute defined as `"true"`. Thus, the context menu is attached to the parent component. The context menu has two items to zoom in (zoom out) a picture by `"onclick"` event. For each item corresponding JavaScript function is defined to provide necessary action as a result of the clicking on it. For the menu is defined an `"oncontextmenu"` event to call the context menu on a right click mouse event.

In the example the context menu is defined for the parent `<h:panelGroup>` component with a value of `"id"` attribute equal to `"picture"`. You should be careful with such definition, because a client context menu is looked for a DOM element with a client Id of a parent component on a server. If a parent component doesn't encode an Id on a client, it can't be found by the `<rich:contextMenu>` and it's attached to its closest parent in a DOM tree.

If the `"attached"` attribute has `"false"` value, component activates via JavaScript API with assistance of `<rich:componentControl>`. An example is placed below.

Example:

```
...
    <h:form id="form">
        <rich:contextMenu attached="false" id="menu" submitMode="ajax">
            <rich:menuItem ajaxSingle="true">
                <b>{car} {model}</b> details
                <a4j:actionParam name="det"
assignTo="#{ddmenu.current}" value="{car} {model} details"/>
            </rich:menuItem>
            <rich:menuGroup value="Actions">
                <rich:menuItem ajaxSingle="true">
                    Put <b>{car} {model}</b> To Basket
                    <a4j:actionParam name="bask"
assignTo="#{ddmenu.current}" value="Put {car} {model} To Basket"/>
                </rich:menuItem>
                <rich:menuItem value="Read Comments"
ajaxSingle="true">
                    <a4j:actionParam name="bask"
assignTo="#{ddmenu.current}" value="Read Comments"/>
                </rich:menuItem>
                <rich:menuItem ajaxSingle="true">
                    Go to <b>{car}</b> site
                    <a4j:actionParam name="bask"
assignTo="#{ddmenu.current}" value="Go to {car} site"/>
                </rich:menuItem>
            </rich:menuGroup>
        </rich:contextMenu>

        <h:panelGrid columns="2">
            <rich:dataTable
value="#{dataTableScrollerBean.tenRandomCars}" var="car" id="table"

onRowMouseOver="this.style.backgroundColor='#F8F8F8'"

onRowMouseOut="this.style.backgroundColor='{a4jSkin.tableBackgroundColor}'"
rowClasses="cur">
```

```

<rich:column>
    <f:facet name="header">Make</f:facet>
    <h:outputText value="#{car.make}" />
</rich:column>
<rich:column>
    <f:facet name="header">Model</f:facet>
    <h:outputText value="#{car.model}" />
</rich:column>
<rich:column>
    <f:facet name="header">Price</f:facet>
    <h:outputText value="#{car.price}" />
</rich:column>

<rich:componentControl event="onRowClick" for="menu"
operation="show">

    <f:param value="#{car.model}" name="model" />
    <f:param value="#{car.make}" name="car" />
</rich:componentControl>
</rich:dataTable>

<a4j:outputPanel ajaxRendered="true">
    <rich:panel>
        <f:facet name="header">Last Menu
Action</f:facet>
        <h:outputText
value="#{ddmenu.current}"></h:outputText>
    </rich:panel>
</a4j:outputPanel>
</h:panelGrid>
</h:form>
...

```

This is a result:

| Make | Model | Price |
|-----------|----------|-------|
| GMC | Sierra | 18636 |
| Chevrolet | Malibu | 30412 |
| GMC | Yukon | 39719 |
| Ford | Explorer | 44998 |
| Infiniti | G35 | 47579 |
| GMC | Yukon | 28771 |
| Toy | | 15337 |
| For | | |
| Toyota | Camry | |
| Nissan | Maxima | 54635 |

GMC Yukon details

Actions

Last Menu Action
Read Comments

Put GMC Yukon To Basket
Read Comments
Go to GMC site

Figure 6.12. The "attached" attribute usage

In the example the context menu is activated (by clicking on the left mouse button) on the table via JavaScript API with assistance of `<rich:componentControl>`. The attribute `"for"` contains a value of

the `<rich:contextMenu>` Id. For menu appearance JavaScript API function "Show" is used. It is defined with *"operation"* attribute for the `<rich:componentControl>` component. Context menu is recreated after the every call on a client and new {car} and {model} values are inserted in it. In the example for a menu customization macrosubstitutions were used.

The `<rich:contextMenu>` component can be defined once on a page and can be used as shared for different components (this is the main difference from the `<rich:dropDownMenu>` component). It's necessary to define it once on a page (as it was shown in the example above [138]) and activate it on required components via JavaScript API with assistance of `<rich:componentControl>` .

The `<rich:contextMenu>` *"submitMode"* attribute can be set to three possible parameters:

- Server (default)

Regular form submission request is used

- Ajax

Ajax submission is used for switching

- None

The *"action"* and *"actionListener"* item's attributes are ignored. Menu items don't fire any submits themselves. The behavior is fully defined by the components nested inside items.

Note:

As the `<rich:contextMenu>` component doesn't provide its own form, use it between `<h:form>` and `</h:form>` tags.

6.26.6. JavaScript API

Table 6.80. JavaScript API

| Function | Description | Apply to |
|----------|-------------------------|------------------|
| Hide() | Hide component or group | Component, group |
| Show() | Show component or group | Component, group |

6.26.7. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

There are two ways to redefine the appearance of all `<rich:contextMenu>` components at once:

- Redefine the corresponding skin parameters
- Add to your style sheets style classes used by a `<rich:contextMenu>` component

6.26.8. Skin Parameters Redefinition

Table 6.81. Skin parameters redefinition for a border

| Skin parameters | CSS properties |
|---------------------------|------------------|
| panelBorderColor | border-color |
| additionalBackgroundColor | background-color |

Table 6.82. Skin parameters redefinition for a background

| Skin parameters | CSS properties |
|---------------------------|--------------------|
| additionalBackgroundColor | border-top-color |
| additionalBackgroundColor | border-left-color |
| additionalBackgroundColor | border-right-color |

6.26.9. Definition of Custom Style Classes

On the screenshot there are classes names that define styles for component elements.

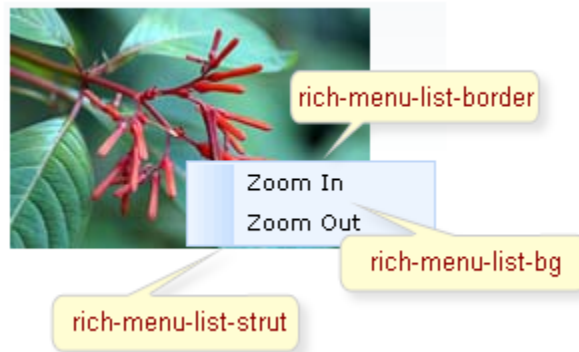


Figure 6.13. Style classes

Table 6.83. Classes names that define the contextMenu element

| Class name | Description |
|-----------------------|--|
| rich-menu-list-border | Defines styles for borders |
| rich-menu-list-bg | Defines styles for a general background list |
| rich-menu-list-strut | Defines styles for a wrapper <div> element for a strut of a popup list |

In order to redefine styles for all `<rich:contextMenu>` components on a page using CSS, it's enough to create classes with the same names and define necessary properties in them.

To change styles of particular `<rich:contextMenu>` components, define your own style classes in the corresponding `<rich:contextMenu>` component attributes.

6.26.10. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/contextMenu.jsf?c=contextMenu>] you can see an example of `<rich:contextMenu>` usage and sources for the given example.

6.27. < rich:dataFilterSlider >

6.27.1. Description

A slider-based action component is used for filtering table data.

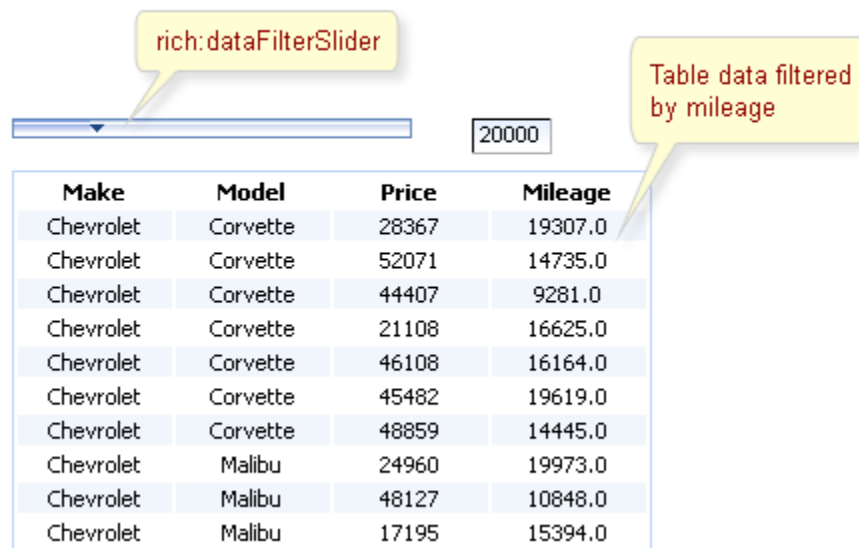


Figure 6.14. `<rich:dataFilterSlider>` component

6.27.2. Key Features

- Filter any UIData based component in dependency on its child's values
- Fully skinnable control and input elements
- Optional value text field with an attribute-managed position
- Optional disablement of the component on a page
- Optional toolTip to display the current value while a handle is dragged
- Dragged state is stable after the mouse moves
- Optional manual input possible if a text input field is present
- Validation of manual input

Table 6.84. rich : dataFilterSlider attributes

| Attribute Name | Description |
|------------------|---|
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| endRange | A slider end point |
| fieldStyleClass | The styleClass for input that displays the value : 'manualInput' must be true |
| filterBy | A getter of an object member required to compare a slider value to. This is a value that is used in results filtering |
| for | The component using UIData (datatable id) |
| forValRef | This is a string which is used in a value attribute of the datatable. It is used for resetting the datatable back to the original list provided by a backing bean |
| handleStyleClass | The handleStyleClass for a handle |
| handleValue | Current handle value |
| id | Every component may have a unique id that is automatically created if omitted |
| increment | Amount to which a handle on each slide/move should be incremented |
| manualInput | False value for this attribute makes text field "read-only" and "hidden". Hence, the value can be changed only from a handle |
| onChange | If the slider value changes must submit a form, onSlide or OnChange can be true |
| onSlide | If the slider value changes must submit a form, onSlide or OnChange can be true |
| rangeStyleClass | The rangeStyleClass for the background div showing a full range |
| rendered | If "false", this component is not rendered |
| sliderListener | MethodBinding representing an action listener method that will be notified after changing of slider control position |
| startRange | A slider begin point |
| storeResults | |

| Attribute Name | Description |
|-------------------|--|
| | Specifies if the component will store a UIData object (your table rows) in session |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | The styleClass for the container div surrounding the component |
| trackStyleClass | The trackStyleClass for a background div |
| trailer | It shows or hides a trailer following a handle |
| trailerStyleClass | The trailerStyleClass for a div following a handle |
| width | Width of the slider control |

Table 6.85. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | org.richfaces.dataFilterSlider |
| component-class | org.richfaces.component.html.HtmlDataFilterSlider |
| component-family | org.richfaces.DataFilterSlider |
| renderer-type | org.richfaces.DataFilterSliderRenderer |
| tag-class | org.richfaces.taglib.dataFilterSliderTag |

6.27.3. Creating the Component with a Page Tag

Here is a simple example as it could be used on a page:

Example:

```
...
<rich:dataFilterSlider sliderListener="#{mybean.doSlide}" startRange="0"
                        endRange="50000" increment="10000"
handleValue="1" />
...
```

6.27.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlDataFilterSlider;
...
HtmlDataFilterSlider mySlider = new HtmlDataFilterSlider();
...
```

6.27.5. Details of Usage

The **dataFilterSlider** component is bound to some UIData component using a *"for"* attribute and filters data in a table.

Example:

```
...
    <rich:dataFilterSlider sliderListener="#{mybean.doSlide}"
        startRange="0"
        endRange="50000"
        increment="10000"
        handleValue="1"
        for="carIndex"
        forValRef="inventoryList.carInventory"
        filterBy="getMileage"
    />
...
    <h:dataTable id="carIndex">
        ...
    </h:dataTable>
...
```

In this example other two attributes are used for filtering:

- *"forValRef"* is a string which is used in a value attribute of the target UIData component. It's designed for resetting the UIData component back to the original list provided by a backing bean.
- *"filterBy"* is a getter of an object member that is to be compared to a slider value. It's a value that is used in results filtering.

"handleValue" is an attribute for keeping the current handler position on the dataFilterSlider component. Based on the current value, appropriate values obtained from a getter method defined in *"filterBy"* are filtered.

One more important attribute is a *"storeResults"* one that allows the dataFilterSlider component to keep UIData target object in session.

If it's necessary the component submits a form on event of a handler state changing, use the *"onSlide"* attribute (*"onChange"* is its alias). When the attribute definition = true, submission on this event is defined.

6.27.6. Look-and-Feel Customization

The **<rich:dataFilterSlider>** component has no skin parameters and special *style classes*, as it consists of one element generated with a your method on the server. To define some style properties such as an indent or a border, it's possible to use *"style"* and *"styleClass"* attributes on the component.

6.27.7. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/dataFilterSlider.jsf?c=dataFilterSlider>] you can see the example of **<rich:dataFilterSlider>** usage and sources for the given example.

6.28. < rich:datascroller >

6.28.1. Description

The component designed for providing the functionality of tables scrolling using Ajax requests.

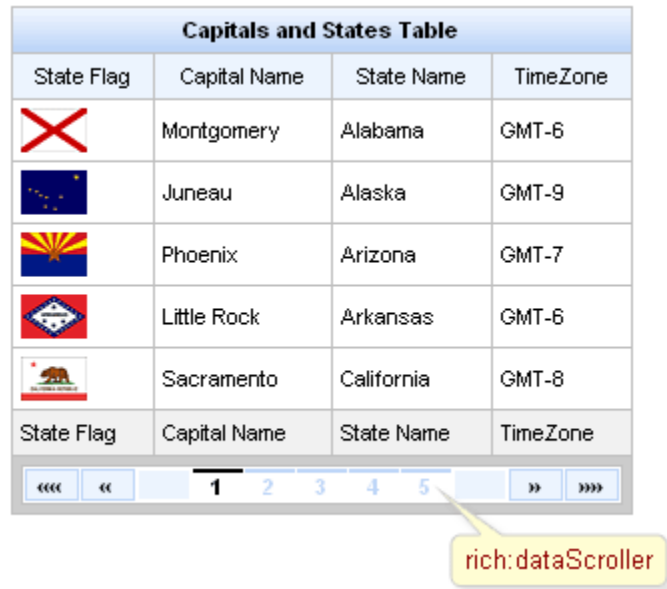


Figure 6.15. <rich:datascroller> component

6.28.2. Key Features

- Provides table scrolling functionality
- Built-in Ajax processing
- Provides fast controls
- Skin support

Table 6.86. rich : datascroller attributes

| Attribute Name | Description |
|------------------|---|
| action | MethodBinding pointing at the application action to be invoked, if this UIComponent is activated by the user, during the Apply Request Values or Invoke Application phase of the request processing lifecycle, depending on the value of the immediate property |
| actionExpression | The action method binding expression |
| actionListener | MethodBinding pointing at method accepting an ActionEvent with return type void |
| ajaxSingle | If "true", submits ONLY one field/link, instead of all form controls |

| Attribute Name | Description |
|--------------------|--|
| align | left center right [CI] Deprecated. This attribute specifies the position of the table with respect to the document. Permitted values: * left: The table is to the left of the document. * center: The table is to the center of the document. * right: The table is to the right of the document |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| boundaryControls | Possible values are: "show" - default mode. Controls are visible always. "hide" - controls are hidden. "auto" - unnecessary controls are hidden |
| bypassUpdates | If "true", after process validations phase it skips updates of model beans on a force render response. It can be used for validating components input |
| data | Serialized (on default with JSON) data passed on the client by a developer on AJAX request. It's accessible via "data.foo" syntax |
| eventsQueue | Name of requests queue to avoid send next request before complete other from same event. Can be used to reduce number of requests of frequently events (key press, mouse move etc.) |
| fastControls | Possible values are: "show" - default mode. Controls are visible always. "hide" - controls are hidden. "auto" - unnecessary controls are hidden |
| fastStep | The attribute indicates pages quantity to switch onto when fast scrolling is used |
| focus | id of element to set focus after request completed on client side |
| for | ID of the table component whose data is scrolled |
| handleValue | Current handle value |
| id | Every component may have a unique id that is automatically created if omitted |
| ignoreDupResponses | Attribute allows to ignore an Ajax Response produced by a request if the newest 'similar' request is in a queue already. ignoreDupResponses="true" does not cancel the request while it is processed on the server, but just |

| Attribute Name | Description |
|--------------------|---|
| | allows to avoid unnecessary updates on the client side if the response isn't actual now |
| immediate | A flag indicating that this component value must be converted and validated immediately (that is, during Apply Request Values phase), rather than waiting until a Process Validations phase |
| inactiveStyle | Corresponds to the HTML style attribute for the inactive cell on scroller |
| inactiveStyleClass | Corresponds to the HTML class attribute for the inactive cell on scroller |
| limitToList | If "true", updates on client side ONLY elements from this 'reRender' property. If "false" (default) updates all rendered by ajax region components |
| maxPages | Maximum quantity of pages |
| onbeforedomupdate | JavaScript code for call before DOM has been updated on client side |
| onclick | HTML: a script expression; a pointer button is clicked |
| oncomplete | JavaScript code for call after request completed on client side |
| ondblclick | HTML: a script expression; a pointer button is double-clicked |
| onkeydown | HTML: a script expression; a key is pressed down |
| onkeypress | HTML: a script expression; a key is pressed and released |
| onkeyup | HTML: a script expression; a key is released |
| onmousedown | HTML: script expression; a pointer button is pressed down |
| onmousemove | HTML: a script expression; a pointer is moved within |
| onmouseout | HTML: a script expression; a pointer is moved away |
| onmouseover | HTML: a script expression; a pointer is moved onto |
| onmouseup | HTML: script expression; a pointer button is released |
| onpagechange | <code>\${prop.xmlEncodedDescription}</code> |
| pageIndexVar | Name of variable in request scope containing index of active page |

| Attribute Name | Description |
|--------------------|---|
| pagesVar | Name of variable in request scope containing number of pages |
| rendered | If "false", this component is not rendered |
| renderIfSinglePage | If renderIfSinglePage=true then datascroller is displayed on condition that the data hold on one page |
| requestDelay | Attribute defines the time (in ms.) that the request will be wait in the queue before it is ready to send. When the delay time is over, the request will be sent to the server or removed if the newest 'similar' request is in a queue already |
| reRender | Id[s] (in format of call <code>UIComponent.findComponent()</code>) of components, rendered in case of <code>AjaxRequest</code> caused by this component. Can be single id, comma-separated list of Id's, or EL Expression with array or Collection |
| scrollerListener | MethodBinding representing an action listener method that will be notified after scrolling |
| selectedStyle | Corresponds to the HTML style attribute for the selected cell on scroller |
| selectedStyleClass | Corresponds to the HTML class attribute for the selected cell on scroller |
| status | ID (in format of call <code>UIComponent.findComponent()</code>) of Request status component |
| stepControls | Possible values are: "show" - default mode. Controls are visible always. "hide" - controls are hidden. "auto" - unnecessary controls are hidden |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| tableStyle | CSS style(s) is/are to be applied to outside table when this component is rendered |
| tableStyleClass | Space-separated list of CSS style class(es) that are be applied to outside table of this component |
| timeout | Response waiting time on a particular request. If a response is not received during this time, the request is aborted |

| Attribute Name | Description |
|----------------|--------------------------------------|
| value | The current value for this component |

Table 6.87. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | org.richfaces.Datascroller |
| component-class | org.richfaces.component.html.HtmlDataScroller |
| component-family | org.richfaces.Datascroller |
| renderer-type | org.richfaces.DataScrollerRenderer |
| tag-class | org.richfaces.taglib.DatascrollerTag |

6.28.3. Creating the Component with a Page Tag

Here is a simple example as it could be used on a page:

Example:

```
...
<h:dataTable id="table">
    ...
</h:dataTable>
...
<rich:datascroller for="table"/>
...
```

6.28.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlDataScroller;
...
HtmlDataScroller myScroll = new HtmlDataScroller();
...
```

6.28.5. Details of Usage

The **<rich:datascroller>** component provides table scrolling functionality the same as TOMAHAWK scroller but with Ajax requests usage.

The component should be placed into footer of the parent table or be bound to it with the *"for"* attribute.

The table should also have the defined *"rows"* attribute limiting the quantity of inputted table rows.

The scroller could limit the maximum quantity of rendered links on the table pages with the help of the *"maxPages"* attribute.

Component provides two controllers groups for switching:

- Page numbers for switching onto a particular page
- The controls of fast switching: *"first"*, *"last"*, *"next"*, *"previous"*, *"fastforward"*, *"fastrewind"*

The controls of fast switching are created adding the facets component with the corresponding name:

Example:

```
...
<rich:datascroller for="table" maxPages="10">
  <f:facet name="first">
    <h:outputText value="First"/>
  </f:facet>
  <f:facet name="last">
    <h:outputText value="Last"/>
  </f:facet>
</rich:datascroller>
...
```



Figure 6.16. <rich:datascroller> controls

The screenshot shows one controller from each group.

There are also facets used to create the disabled states: *"first_disabled"*, *"last_disabled"*, *"next_disabled"*, *"previous_disabled"*, *"fastforward_disabled"*, *"fastrewind_disabled"*.

For the *"fastforward"/"fastrewind"* controls customization the additional *"fastStep"* attribute is used. The attribute indicates pages quantity to switch onto when fast scrolling is used.

The *"pageIndexVar"* and *"pagesVar"* attributes provide an ability to show the current page and the number of pages in the datascroller. These attributes are used for definition the names of variables, that will be used in the facet with name *"pages"*. An example can be found below:

Example:

```

...
<h:form>
    <rich:dataTable value="#{capitalsBean.capitals}" var="cap" rows="5">
        <rich:column>
            <h:outputText value="#{cap.name}"></h:outputText>
        </rich:column>
        <f:facet name="footer">
            <rich:datascroller pageIndexVar="pageIndex" pagesVar="pages">
                <f:facet name="pages">
                    <h:outputText value="#{pageIndex} /
#{pages}"></h:outputText>
                </f:facet>
            </rich:datascroller>
        </f:facet>
    </rich:dataTable>
</h:form>
...

```

It's possible to insert optional separators between controls. For this purpose use a *"controlSeparator"* facet. An example is placed below.

```

...
<f:facet name="controlSeparator">
    <h:graphicImage value="/image/sep.png" />
</f:facet>
...

```

6.28.6. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

There are two ways to redefine the appearance of all **<rich:datascroller>** components at once:

- Redefine the corresponding skin parameters
- Add to your style sheets *style classes* used by a **<rich:datascroller>** component

6.28.7. Skin Parameters Redefinition

Table 6.88. Skin parameters redefinition for a wrapper element

| Skin parameters | CSS properties |
|----------------------|------------------|
| tableBackgroundColor | background-color |
| panelBorderColor | border-color |

Table 6.89. Skin parameters redefinition for a button

| Skin parameters | CSS properties |
|---------------------------|------------------|
| additionalBackgroundColor | background-color |
| panelBorderColor | border-color |

| Skin parameters | CSS properties |
|-------------------|----------------|
| generalFamilyFont | font-family |
| generalSizeFont | font-size |

Table 6.90. Skin parameters redefinition for an active button

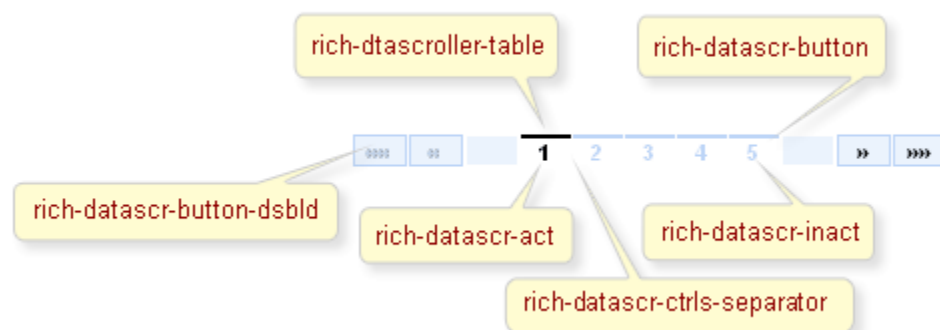
| Skin parameters | CSS properties |
|-------------------|------------------|
| generalTextColor | border-top-color |
| generalTextColor | color |
| generalFamilyFont | font-family |
| generalSizeFont | font-size |

Table 6.91. Skin parameters redefinition for an inactive button

| Skin parameters | CSS properties |
|-----------------------|------------------|
| headerBackgroundColor | border-top-color |
| headerBackgroundColor | color |
| generalFamilyFont | font-family |
| generalSizeFont | font-size |

6.28.8. Definition of Custom Style Classes

On the screenshot there are classes names that define styles for component elements.

**Figure 6.17. Style classes****Table 6.92. Classes names that define a component appearance**

| Class name | Description |
|--------------|--|
| rich-datascr | Defines styles for a wrapper <div> element of a datascroller |

| Class name | Description |
|------------------------------|--|
| rich-dtascroller-table | Defines styles for a wrapper table element of a datascroller |
| rich-datascr-button | Defines styles for a button |
| rich-datascr-ctrls-separator | Defines styles for a separator between buttons |

Table 6.93. Classes names that define a buttons appearance

| Class name | Description |
|--------------------------|---------------------------------------|
| rich-datascr-act | Defines styles for an active button |
| rich-datascr-inact | Defines styles for an inactive button |
| rich-datascr-button-dsbl | Defines styles for a disabled button |

In order to redefine styles for all **<rich:datascroller>** components on a page using CSS, it's enough to create classes with the same names and define necessary properties in them.

To change styles of particular **<rich:datascroller>** components, define your own style classes in the corresponding **<rich:datascroller>** attributes.

CSS code piece used on a page:

Example:

```
...
.rich-datascr-button{
    color: #A52A2A;
}
.myClass{
    font-size: xx-large;
}
...
```

The component is defined in the following way:

Example:

```
<rich:datascroller ... inputClass="myClass" .../>
```

6.28.9. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/dataTableScroller.jsf?c=dataTableScroller>] you can see the example of **<rich:datascroller>** usage and sources for the given example.

The solution about how to do correct pagination using datascroller (load a part of data from database) can be found on the RichFaces Users Forum [<http://www.jboss.com/index.html?module=bb&op=viewtopic&p=4060199#4060199>].

How to use `<rich:dataTable>` and `<rich:datascroller>` in a context of Extended Data Model see here [<http://www.jboss.com/index.html?module=bb&op=viewtopic&t=115636>].

6.29. < rich:columnGroup >

6.29.1. Description

The component combines columns in one row to organize complex subparts of a table.

| State Flag | | |
|---|-------------|-------|
|  | | |
| Alabama | Montgomery | GMT-6 |
|  | | |
| Alaska | Juneau | GMT-9 |
|  | | |
| Arizona | Phoenix | GMT-7 |
|  | | |
| Arkansas | Little Rock | GMT-6 |
|  | | |
| California | Sacramento | GMT-8 |

Figure 6.18. `<rich:columnGroup>` component

6.29.2. Key Features

- Completely skinned table columns and child elements
- Possibility to combine columns and rows inside
- Possibility to update a limited set of strings with Ajax

Table 6.94. `rich : columnGroup` attributes

| Attribute Name | Description |
|----------------|---|
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| columnClasses | Comma-delimited list of CSS style classes that are be applied to the columns of this table. A space separated list of classes may also be specified for any individual column. If the number of elements in this list is less than the number of columns specified in the "columns" |

| Attribute Name | Description |
|----------------|--|
| | attribute, no "class" attribute is output for each column greater than the number of elements in the list. If the number of elements in the list is greater than the number of columns specified in the "columns" attribute, the elements at the position in the list after the value of the "columns" attribute are ignored |
| dir | Direction indication for text that does not inherit directionality. Valid values are "LTR" (left-to-right) and "RTL" (right-to-left) |
| id | Every component may have a unique id that is automatically created if omitted |
| lang | Code describing the language used in the generated markup for this component |
| rendered | If "false", this component is not rendered |
| rowClasses | A comma-delimited list of CSS style classes that is applied to popup table rows. A space separated list of classes may also be specified for any individual row. The styles are applied, in turn, to each row in the table. For example, if the list has two elements, the first style class in the list is applied to the first row, the second to the second row, the first to the third row, the second to the fourth row, etc. In other words, we keep iterating through the list until we reach the end, and then we start at the beginning again |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| title | Advisory title information about markup elements generated for this component |

Table 6.95. Component identification parameters

| Name | Value |
|------------------|--|
| component-type | org.richfaces.ColumnGroup |
| component-class | org.richfaces.component.html.HtmlColumnGroup |
| component-family | org.richfaces.ColumnGroup |
| renderer-type | org.richfaces.ColumnGroupRenderer |
| tag-class | org.richfaces.taglib.ColumnGroupTag |

6.29.3. Creating the Component with a Page Tag

To create the simplest variant of `columnGroup` on a page, use the following syntax:

Example:

```
...
    <rich:columnGroup>
        <rich:column>
            <h:outputText value="Column1"/>
        </rich:column>
        <rich:column>
            <h:outputText value="Column2"/>
        </rich:column>
    </rich:columnGroup>
...
```

6.29.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlColumnGroup;
...
HtmlColumnGroup myRow = new HtmlColumnGroup();
...
```

6.29.5. Details of Usage

The `<rich:columnGroup>` component combines columns set wrapping them into the `<tr>` element and outputting them into one row. Columns are combined in a group the same way as when the *"breakBefore"* attribute is used for columns to add a moving to the next rows, but the first variant is clearer from a source code. Hence, the following simple examples are very same.

Example:

```
...
    <rich:dataTable value="#{capitalsBean.capitals}" var="cap" rows="5" id="sublist">

        <rich:column colspan="3">
            <f:facet name="header">State Flag</f:facet>
            <h:graphicImage value="#{cap.stateFlag}"/>
        </rich:column>
        <rich:columnGroup>
            <rich:column>
                <h:outputText value="#{cap.state}"/>
            </rich:column>
            <rich:column >
                <h:outputText value="#{cap.name}"/>
            </rich:column>
            <rich:column >
                <h:outputText value="#{cap.timeZone}"/>
            </rich:column>
        </rich:columnGroup>
    </rich:dataTable>
```

...

And representation without a grouping:

Example:

```
...
<rich:dataTable value="#{capitalsBean.capitals}" var="cap" rows="5" id="sublist">

    <rich:column colspan="3">
        <f:facet name="header">State Flag</f:facet>
        <h:graphicImage value="#{cap.stateFlag}"/>
    </rich:column>
    <rich:column breakBefore="true">
        <h:outputText value="#{cap.state}"/>
    </rich:column>
    <rich:column breakBefore="true">
        <h:outputText value="#{cap.name}"/>
    </rich:column>
    <rich:column >
        <h:outputText value="#{cap.timeZone}"/>
    </rich:column>
</rich:dataTable>
....
```

The result is:






| State Flag | | |
|---|-------------|-------|
|  | | |
| Alabama | Montgomery | GMT-6 |
|  | | |
| Alaska | Juneau | GMT-9 |
|  | | |
| Arizona | Phoenix | GMT-7 |
|  | | |
| Arkansas | Little Rock | GMT-6 |
|  | | |
| California | Sacramento | GMT-8 |

Figure 6.19. Generated <rich:columnGroup> component

It's also possible to use the component for output of complex headers in a table. For example adding of a complex header to a facet for the whole table looks the following way:

Example:

...


```

<f:facet name="header">
  <rich:columnGroup>
    <rich:column rowspan="2">
      <h:outputText value="State Flag"/>
    </rich:column>
    <rich:column colspan="3">
      <h:outputText value="State Info"/>
    </rich:column>
    <rich:column breakBefore="true">
      <h:outputText value="State Name"/>
    </rich:column>
    <rich:column>
      <h:outputText value="State Capital"/>
    </rich:column>
    <rich:column>
      <h:outputText value="Time Zone"/>
    </rich:column>
  </rich:columnGroup>
</f:facet>
...

```

Generated on a page as:

| State Flag | State Info | | |
|---|------------|---------------|-----------|
| | State Name | State Capital | Time Zone |
|  | Alabama | Montgomery | GMT-6 |
|  | Alaska | Juneau | GMT-9 |
|  | Arizona | Phoenix | GMT-7 |
|  | Arkansas | Little Rock | GMT-6 |
|  | California | Sacramento | GMT-8 |

Figure 6.20. `<rich:columnGroup>` with complex headers

6.29.6. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

There are two ways to redefine the appearance of all `<rich:columnGroup>` components at once:

- Redefine the corresponding skin parameters
- Add to your style sheets *style classes* used by a `<rich:columnGroup>` component

6.29.7. Skin Parameters Redefinition

Skin parameters redefinition for `<rich:columnGroup>` are the same as for the `<rich:dataTable>` component.

6.29.8. Definition of Custom Style Classes

Custom style classes for `<rich:columnGroup>` are the same as for the `<rich:dataTable>` component.

In order to redefine styles for all `<rich:columnGroup>` components on a page using CSS, it's enough to create classes with the same names and define necessary properties in them.

To change styles of particular `<rich:columnGroup>` components, define your own style classes in the corresponding `<rich:columnGroup>` attributes.

6.29.9. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/dataTable.jsf?c=columnGroup>] you can see the example of `<rich:columnGroup>` usage and sources for the given example.

6.30. < rich:column >

6.30.1. Description

The component for row rendering for a UIData component.

United States Capitals

| Capitals and States Table | | | |
|---|--------------|------------|----------|
| State Flag | Capital Name | State Name | TimeZone |
|  | Montgomery | Alabama | GMT-6 |
|  | Juneau | Alaska | GMT-9 |
|  | Phoenix | Arizona | GMT-7 |
|  | Little Rock | Arkansas | GMT-6 |
|  | Sacramento | California | GMT-8 |
| State Flag | Capital Name | State Name | TimeZone |
| Capitals and States Table | | | |

Figure 6.21. `<rich:column>` component

6.30.2. Key Features

- Completely skinned table rows and child elements
- Possibility to combine columns with the help of `"colspan"`
- Possibility to combine rows with the help of `"rowspan"` and `"breakBefore"`

Table 6.96. `rich : column` attributes

| Attribute Name | Description |
|----------------|-------------|
| binding | |

| Attribute Name | Description |
|----------------|--|
| | The attribute takes a value-binding expression for a component property of a backing bean |
| breakBefore | if "true" next column begins from the first row |
| colspan | Corresponds to the HTML colspan attribute |
| dir | Direction indication for text that does not inherit directionality. Valid values are "LTR" (left-to-right) and "RTL" (right-to-left) |
| footerClass | Space-separated list of CSS style class(es) that are be applied to any footer generated for this table |
| headerClass | Space-separated list of CSS style class(es) that are be applied to any header generated for this table |
| id | Every component may have a unique id that is automatically created if omitted |
| lang | Code describing the language used in the generated markup for this component |
| rendered | If "false", this component is not rendered |
| rowspan | Corresponds to the HTML rowspan attribute |
| sortable | Boolean attribute. If "true" it's possible to sort the column content after click on the header. Default value is "true" |
| sortExpression | Attribute defines a bean property which is used for sorting of a column |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| title | Advisory title information about markup elements generated for this component |
| width | Attribute defines width of column. Default value is "100px". |

Table 6.97. Component identification parameters

| Name | Value |
|-----------------|---|
| component-type | org.richfaces.Column |
| component-class | org.richfaces.component.html.HtmlColumn |

| Name | Value |
|------------------|--------------------------------|
| component-family | org.richfaces.Column |
| renderer-type | org.richfaces.ColumnRenderer |
| tag-class | org.richfaces.taglib.ColumnTag |

6.30.3. Creating the Component with a Page Tag

To create the simplest variant of column on a page, use the following syntax:

Example:

```
...
<rich:dataTable var="set">
  <rich:column>
    <h:outputText value="#{set.property1}" />
  </rich:column>
  <!--Set of another columns and header/footer facets-->
</rich:dataTable>
...
```

6.30.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlColumn;
...
HtmlColumn myColumn = new HtmlColumn();
...
```

6.30.5. Details of Usage

To output a simple table, the `<rich:column>` component is used the same way as the standard `<h:column>`, i.e. the following code on a page is used:

Example:

```
...
<rich:dataTable value="#{capitalsBean.capitals}" var="cap" rows="5">
  <rich:column>
    <f:facet name="header">State Flag</f:facet>
    <h:graphicImage value="#{cap.stateFlag}" />
  </rich:column>
  <rich:column>
    <f:facet name="header">State Name</f:facet>
    <h:outputText value="#{cap.state}" />
  </rich:column>
  <rich:column>
    <f:facet name="header">State Capital</f:facet>
    <h:outputText value="#{cap.name}" />
  </rich:column>
</rich:dataTable>
```

```

<rich:column>
  <f:facet name="header">Time Zone</f:facet>
  <h:outputText value="#{cap.timeZone}" />
</rich:column>
</rich:dataTable>
...

```

The result is:

| State Flag | State Name | State Capital | Time Zone |
|---|------------|---------------|-----------|
|  | Alabama | Montgomery | GMT-6 |
|  | Alaska | Juneau | GMT-9 |
|  | Arizona | Phoenix | GMT-7 |
|  | Arkansas | Little Rock | GMT-6 |
|  | California | Sacramento | GMT-8 |

Figure 6.22. Generated <rich:column> component

Now, in order to group columns with text information into one row in one column with a flag, use the *"colspan"* attribute, which is similar to an HTML one, specifying that the first column contains 3 columns. In addition, it's necessary to specify that the next column begins from the first row with the help of the *"breakBefore"* attribute = true.

Example:

```

...
<rich:dataTable value="#{capitalsBean.capitals}" var="cap" rows="5">
  <rich:column colspan="3">
    <h:graphicImage value="#{cap.stateFlag}" />
  </rich:column>
  <rich:column breakBefore="true">
    <h:outputText value="#{cap.state}" />
  </rich:column>
  <rich:column >
    <h:outputText value="#{cap.name}" />
  </rich:column>
  <rich:column>
    <h:outputText value="#{cap.timeZone}" />
  </rich:column>
</rich:dataTable>
...

```

As a result the following structure is rendered:






| | | |
|---|-------------|-------|
|  | | |
| Alabama | Montgomery | GMT-6 |
|  | | |
| Alaska | Juneau | GMT-9 |
|  | | |
| Arizona | Phoenix | GMT-7 |
|  | | |
| Arkansas | Little Rock | GMT-6 |
|  | | |
| California | Sacramento | GMT-8 |

Figure 6.23. `<rich:column>` modified with `colspan` and `breakbefore` attributes

The same way is used for columns grouping with the `"rowspan"` attribute that is similar to an HTML one responsible for rows quantity definition occupied with the current one. The only thing to add in the example is an instruction to move onto the next row for each next after the second column.

Example:

```
...
<rich:dataTable value="#{capitalsBean.capitals}" var="cap" rows="5">
  <rich:column rowspan="3">
    <f:facet name="header">State Flag</f:facet>
    <h:graphicImage value="#{cap.stateFlag}"/>
  </rich:column>
  <rich:column>
    <f:facet name="header">State Info</f:facet>
    <h:outputText value="#{cap.state}"/>
  </rich:column>
  <rich:column breakBefore="true">
    <h:outputText value="#{cap.name}"/>
  </rich:column>
  <rich:column breakBefore="true">
    <h:outputText value="#{cap.timeZone}"/>
  </rich:column>
</rich:dataTable>
...
```

As a result:

| State Flag | State Info |
|---|-------------|
|  | Alabama |
| | Montgomery |
| | GMT-6 |
|  | Alaska |
| | Juneau |
| | GMT-9 |
|  | Arizona |
| | Phoenix |
| | GMT-7 |
|  | Arkansas |
| | Little Rock |
| | GMT-6 |
|  | California |
| | Sacramento |
| | GMT-8 |

Figure 6.24. `<rich:column>` generated with `rowspan` attribute

Hence, additionally to a standard output of a particular row provided with the `<h:column>` component, it becomes possible to group easily the rows with special HTML attribute.

The columns also could be grouped in a particular way with the help of the `<h:columnGroup>` component that is described in the following chapter.

6.30.6. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

There are two ways to redefine the appearance of all `<rich:column>` components at once:

- Redefine the corresponding skin parameters
- Add to your style sheets *style classes* used by a `<rich:column>` component

6.30.7. Skin Parameters Redefinition

Skin parameters redefinition for `<rich:column>` are the same as for the `<rich:dataTable>` component.

6.30.8. Definition of Custom Style Classes

Custom style classes for `<rich:column>` are the same as for the `<rich:dataTable>` component.

In order to redefine styles for all `<rich:column>` components on a page using CSS, it's enough to create classes with the same names and define necessary properties in them.

To change styles of particular `<rich:column>` components, define your own style classes in the corresponding `<rich:column>` attributes.

6.30.9. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/dataTable.jsf?c=column>] you can see the example of `<rich:column>` usage and sources for the given example.

6.31. < rich:dataGrid >

6.31.1. Description

The component to render data as a grid that allows choosing data from a model and obtains built-in support of Ajax updates.

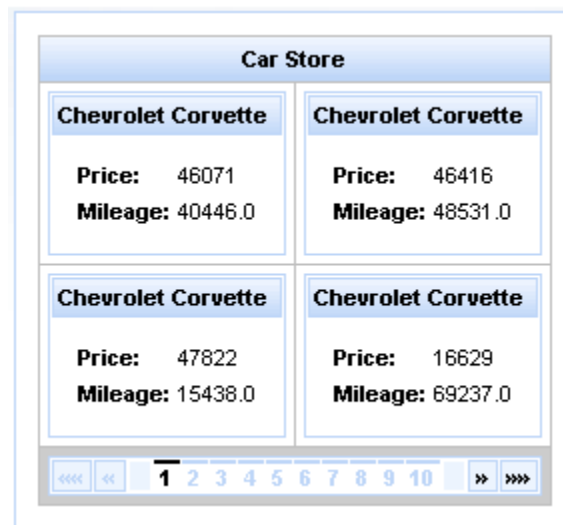


Figure 6.25. `<rich:dataGrid>` component

6.31.2. Key Features

- A completely skinned table and child elements
- Possibility to update a limited set of rows with AJAX
- Possibility to receive values dynamically from a model

Table 6.98. `rich : dataGrid` attributes

| Attribute Name | Description |
|-----------------------|--|
| <code>ajaxKeys</code> | This attribute defines rows that are updated after an AJAX request |
| <code>align</code> | left center right [CI] Deprecated. This attribute specifies the position of the table with respect to the document. Permitted values: * left: The table is to the left of the document. * center: The table is to the center |

| Attribute Name | Description |
|----------------|--|
| | of the document. * right: The table is to the right of the document |
| bgcolor | Deprecated. This attribute sets the background color for the document body or table cells. This attribute sets the background color of the canvas for the document body (the BODY element) or for tables (the TABLE, TR, TH, and TD elements). Additional attributes for specifying text color can be used with the BODY element. This attribute has been deprecated in favor of style sheets for specifying background color information |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| border | This attributes specifies the width of the frame around a component |
| captionClass | Space-separated list of CSS style class(es) that are be applied to caption for this component |
| captionStyle | CSS style(s) is/are to be applied to caption when this component is rendered |
| cellpadding | This attribute specifies the amount of space between the border of the cell and its contents |
| cellspacing | This attribute specifies the amount of space between the border of the cell and its contents. The attribute also specifies the amount of space to leave between cells |
| columnClasses | Comma-delimited list of CSS style classes that are be applied to the columns of this table. A space separated list of classes may also be specified for any individual column. If the number of elements in this list is less than the number of columns specified in the "columns" attribute, no "class" attribute is output for each column greater than the number of elements in the list. If the number of elements in the list is greater than the number of columns specified in the "columns" attribute, the elements at the position in the list after the value of the "columns" attribute are ignored |
| columns | Quantity of columns |
| componentState | It defines EL-binding for a component state for saving or redefinition |

| Attribute Name | Description |
|----------------|--|
| dir | Direction indication for text that does not inherit directionality. Valid values are "LTR" (left-to-right) and "RTL" (right-to-left) |
| elements | Number of elements in grid |
| first | A zero-relative row number of the first row to display |
| footerClass | Space-separated list of CSS style class(es) that are be applied to footer for this component |
| frame | void above below hsides lhs rhs vsides box border [CI] This attribute specifies which sides of the frame surrounding a table will be visible. Possible values: * void: No sides. This is the default value. * above: The top side only. * below: The bottom side only. * hsides: The top and bottom sides only. * vsides: The right and left sides only. * lhs: The left-hand side only. * rhs: The right-hand side only. * box: All four sides. * border: All four sides |
| headerClass | Space-separated list of CSS style class(es) that are be applied to header for this component |
| id | Every component may have a unique id that is automatically created if omitted |
| lang | Code describing the language used in the generated markup for this component |
| onclick | HTML: a script expression; a pointer button is clicked |
| ondblclick | HTML: a script expression; a pointer button is double-clicked |
| onkeydown | HTML: a script expression; a key is pressed down |
| onkeypress | HTML: a script expression; a key is pressed and released |
| onkeyup | HTML: a script expression; a key is released |
| onmousedown | HTML: script expression; a pointer button is pressed down |
| onmousemove | HTML: a script expression; a pointer is moved within |
| onmouseout | HTML: a script expression; a pointer is moved away |
| onmouseover | HTML: a script expression; a pointer is moved onto |

| Attribute Name | Description |
|----------------|--|
| onmouseup | HTML: script expression; a pointer button is released |
| onRowClick | HTML: a script expression; a pointer button is clicked on row |
| onRowDbClick | HTML: a script expression; a pointer button is double-clicked on row |
| onRowMouseDown | HTML: script expression; a pointer button is pressed down on row |
| onRowMouseMove | HTML: a script expression; a pointer is moved within of row |
| onRowMouseOut | HTML: a script expression; a pointer is moved away of row |
| onRowMouseOver | HTML: a script expression; a pointer is moved onto of row |
| onRowMouseUp | HTML: script expression; a pointer button is released on row |
| rendered | If "false", this component is not rendered |
| rowClasses | A comma-delimited list of CSS style classes that is applied to popup table rows. A space separated list of classes may also be specified for any individual row. The styles are applied, in turn, to each row in the table. For example, if the list has two elements, the first style class in the list is applied to the first row, the second to the second row, the first to the third row, the second to the fourth row, etc. In other words, we keep iterating through the list until we reach the end, and then we start at the beginning again |
| rowKey | RowKey is a representation of an identifier for a specific data row |
| rowKeyVar | The attribute provides access to a row key in a Request scope |
| rules | This attribute specifies which rules will appear between cells within a table. The rendering of rules is user agent dependent. Possible values: * none: No rules. This is the default value. * groups: Rules will appear between row groups (see THEAD, TFOOT, and TBODY) and column groups (see COLGROUP and COL) only. * rows: Rules will appear between rows |

| Attribute Name | Description |
|----------------|---|
| | only. * cols: Rules will appear between columns only. * all: Rules will appear between all rows and columns |
| stateVar | The attribute provides access to a component state on the client side |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| summary | This attribute provides a summary of the table's purpose and structure for user agents rendering to non-visual media such as speech and Braille |
| title | Advisory title information about markup elements generated for this component |
| value | The current value for this component |
| var | A request-scope attribute via which the data object for the current row will be used when iterating |
| width | This attribute specifies the desired width of the entire table and is intended for visual user agents. When the value is percentage value, the value is relative to the user agent's available horizontal space. In the absence of any width specification, table width is determined by the user agent |

Table 6.99. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | org.richfaces.DataGrid |
| component-class | org.richfaces.component.html.HtmlDataGrid |
| component-family | org.richfaces.DataGrid |
| renderer-type | org.richfaces.DataGridRenderer |
| tag-class | org.richfaces.taglib.DataGridTag |

6.31.3. Creating the Component with a Page Tag

To create the simplest variant of dataGrid on a page, use the following syntax:

Example:

```
...
<rich:dataGrid value="#{dataTableScrollerBean.allCars}" var="car">
```

```

        <h:outputText value="#{car.model}" />
    </rich:dataGrid>
    ...

```

6.31.4. Creating the Component Dynamically Using Java

Example:

```

import org.richfaces.component.html.HtmlDataGrid;
...
HtmlDataGrid myList = new HtmlDataGrid();
...

```

6.31.5. Details of Usage

The component takes a list from a model and outputs it the same way as with **<h:panelGrid>** for inline data. To define grid properties and styles, use the same definitions as for **<h:panelGrid>**.

The component allows to:

- Use *"header"* and *"footer"* facets for output
- Limit number of output elements (*"elements"* attribute) and define first element for output (*"first"* attribute)
- Bind pages with **<rich:datascroller>** component

Here is an example:

Example:

```

...
<rich:panel style="width:150px;height:200px;">
    <h:form>
        <rich:dataGrid value="#{dataTableScrollerBean.allCars}" var="car"
columns="2" elements="4" first="1">
            <f:facet name="header">
                <h:outputText value="Car Store"></h:outputText>
            </f:facet>
            <rich:panel>
                <f:facet name="header">
                    <h:outputText value="#{car.make}
#{car.model}"></h:outputText>
                </f:facet>
                <h:panelGrid columns="2">
                    <h:outputText value="Price:"
styleClass="label"></h:outputText>
                    <h:outputText value="#{car.price}" />
                    <h:outputText value="Mileage:"
styleClass="label"></h:outputText>
                    <h:outputText value="#{car.mileage}" />
                </h:panelGrid>
            </rich:panel>
        </rich:dataGrid>
    </h:form>
</rich:panel>

```

```

        <f:facet name="footer">
            <rich:datascroller></rich:datascroller>
        </f:facet>
    </rich:dataGrid>
</h:form>
</rich:panel>
...

```

This is a result:

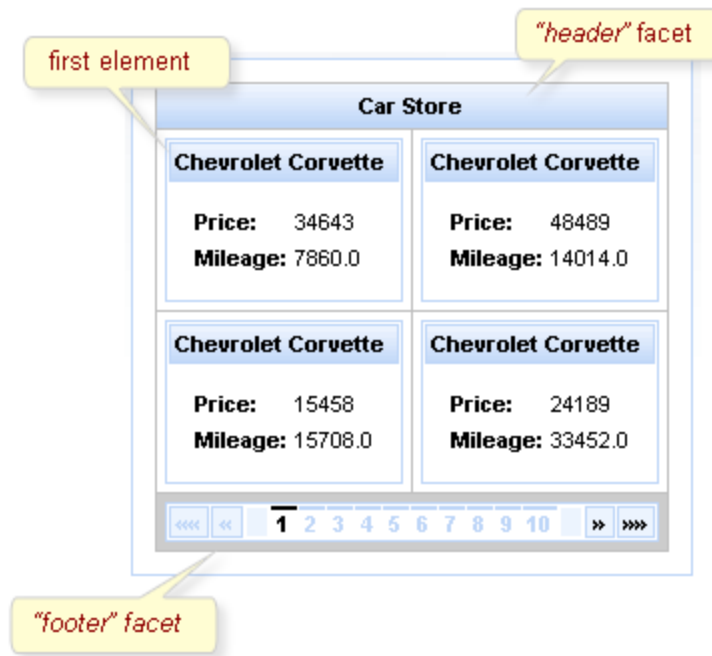


Figure 6.26. Component usage

The component was created basing on the **<a4j:repeat>** component and as a result it could be partially updated with Ajax. *"ajaxKeys"* attribute allows to define rows that are updated after an Ajax request.

Here is an example:

Example:

```

...
<rich:dataGrid value="#{dataTableScrollerBean.allCars}" var="car"
ajaxKeys="#{listBean.list}"
                binding="#{listBean.dataGrid}" id="grid" elements="4"
columns="2">
    ...
</rich:dataGrid>
...
<a4j:commandButton action="#{listBean.action}" reRender="grid" value="Submit"/>
...

```

In the example *"reRender"* attribute contains value of *"id"* attribute for **<rich:dataGrid>** component. As a result the component is updated after an Ajax request.

6.31.6. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

There are two ways to redefine the appearance of all `<rich:dataGrid>` components at once:

- Redefine the corresponding skin parameters
- Add to your style sheets *style classes* used by a `<rich:dataGrid>` component

6.31.7. Skin Parameters Redefinition

Skin parameters redefinition for `<rich:dataGrid>` are the same as for the `<rich:dataTable>` component.

6.31.8. Definition of Custom Style Classes

Custom style classes for `<rich:dataGrid>` are the same as for the `<rich:dataTable>` component.

In order to redefine styles for all `<rich:dataGrid>` components on a page using CSS, it's enough to create classes with the same names and define necessary properties in them.

To change styles of particular `<rich:dataGrid>` components, define your own style classes in the corresponding `<rich:dataGrid>` attributes.

6.31.9. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/dataGrid.jsf?c=dataGrid>] you can see the example of `<rich:dataGrid>` usage and sources for the given example.

6.32. < rich:dataList >

6.32.1. Description

The component for unordered lists rendering that allows choosing data from a model and obtains built-in support of Ajax updates.

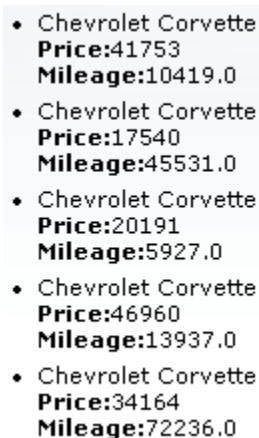
- 
- Chevrolet Corvette
Price:41753
Mileage:10419.0
 - Chevrolet Corvette
Price:17540
Mileage:45531.0
 - Chevrolet Corvette
Price:20191
Mileage:5927.0
 - Chevrolet Corvette
Price:46960
Mileage:13937.0
 - Chevrolet Corvette
Price:34164
Mileage:72236.0

Figure 6.27. `<rich:dataList>` component

6.32.2. Key Features

- A completely skinned list and child elements
- Possibility to update a limited set of rows with AJAX
- Possibility to receive values dynamically from a model

Table 6.100. rich : dataList attributes

| Attribute Name | Description |
|----------------|--|
| ajaxKeys | This attribute defines rows that are updated after an AJAX request |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| dir | Direction indication for text that does not inherit directionality. Valid values are "LTR" (left-to-right) and "RTL" (right-to-left) |
| first | A zero-relative row number of the first row to display |
| id | Every component may have a unique id that is automatically created if omitted |
| lang | Code describing the language used in the generated markup for this component |
| rendered | If "false", this component is not rendered |
| rowClasses | A comma-delimited list of CSS style classes that is applied to popup table rows. A space separated list of classes may also be specified for any individual row. The styles are applied, in turn, to each row in the table. For example, if the list has two elements, the first style class in the list is applied to the first row, the second to the second row, the first to the third row, the second to the fourth row, etc. In other words, we keep iterating through the list until we reach the end, and then we start at the beginning again |
| rowKeyVar | The attribute provides access to a row key in a Request scope |
| rows | A number of rows to display, or zero for all remaining rows in the table |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |

| Attribute Name | Description |
|----------------|---|
| title | Advisory title information about markup elements generated for this component |
| type | Corresponds to the HTML DL type attribute |
| value | The current value for this component |
| var | A request-scope attribute via which the data object for the current row will be used when iterating |

Table 6.101. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | org.richfaces.DataList |
| component-class | org.richfaces.component.html.HtmlDataList |
| component-family | org.richfaces.DataList |
| renderer-type | org.richfaces.DataListRenderer |
| tag-class | org.richfaces.taglib.DataListTag |

6.32.3. Creating the Component with a Page Tag

To create the simplest variant of dataList on a page, use the following syntax:

Example:

```
...
<rich:dataList var="car" value="#{dataTableScrollerBean.allCars}" >
  <h:outputText value="#{car.model}" />
</rich:dataList>
...
```

6.32.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlDataList;
...
HtmlDataList myList = new HtmlDataList();
...
```

6.32.5. Details of Usage

The **<rich:dataList>** component allows to generate a list from a model.

The component has the *"type"* attribute, which corresponds to the *"type"* parameter for the *"UL"* HTML element and defines a marker type. Possible values for *"type"* attribute are: "disc", "circle", "square".

Here is an example:

```
...
<h:form>
  <rich:dataList var="car" value="#{dataTableScrollerBean.allCars}" rows="5"
  type="disc" title="Car Store">
    <h:outputText value="#{car.make} #{car.model}" /><br/>
    <h:outputText value="Price:" styleClass="label"></h:outputText>
    <h:outputText value="#{car.price}" /><br/>
    <h:outputText value="Mileage:" styleClass="label"></h:outputText>
    <h:outputText value="#{car.mileage}" /><br/>
  </rich:dataList>
</h:form>
...
```

This is a result:

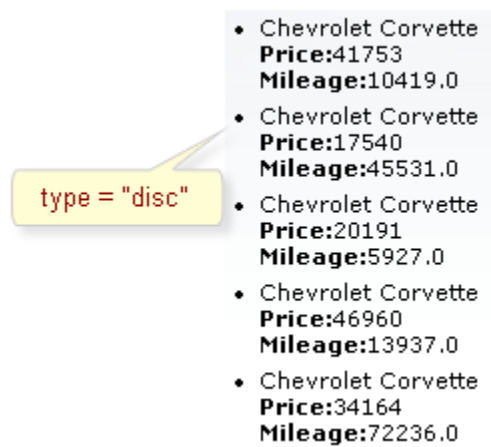


Figure 6.28. Component usage

In the example the *"rows"* attribute limits number of output elements of the list.

"first" attribute defines first element for output. *"title"* are used for popup title. See picture below:



Figure 6.29. Component usage

The component was created basing on the `<a4j:repeat>` component and as a result it could be partially updated with Ajax. `"ajaxKeys"` attribute allows to define rows that are updated after an Ajax request.

Here is an example:

Example:

```
...
<rich:dataList value="#{dataTableScrollerBean.allCars}" var="car"
ajaxKeys="#{listBean.list}"
binding="#{listBean.dataList}" id="list" rows="5"
type="disc">
...
</rich:dataList>
...
<a4j:commandButton action="#{listBean.action}" reRender="list" value="Submit"/>
...
```

In the example `"reRender"` attribute contains value of `"id"` attribute for `<rich:dataList>` component. As a result the component is updated after an Ajax request.

6.32.6. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

There are two ways to redefine the appearance of all `<rich:dataList>` components at once:

- Redefine the corresponding skin parameters
- Add to your style sheets *style classes* used by a `<rich:dataList>` component

6.32.7. Definition of Custom Style Classes

On the screenshot there are classes names that define styles for component elements.

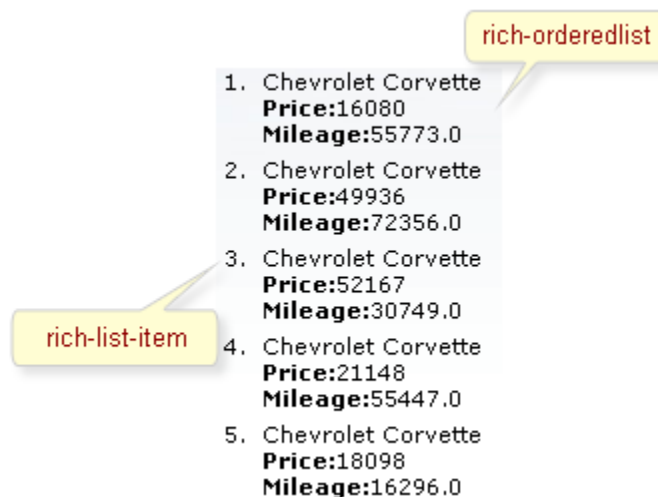


Figure 6.30. Style classes

Table 6.102. Classes names that define a list appearance

| Class name | Description |
|----------------|---|
| rich-datalist | Defines styles for an html element |
| rich-list-item | Defines styles for an html element |

In order to redefine styles for all `<rich:dataList>` components on a page using CSS, it's enough to create classes with the same names and define necessary properties in them.

To change styles of particular `<rich:dataList>` components, define your own style classes in the corresponding `<rich:dataList>` attributes.

6.32.8. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/dataLists.jsf?c=dataList>] you can see the example of `<rich:dataList>` usage and sources for the given example.

6.33. < rich:dataOrderedList >

6.33.1. Description

The component for ordered lists rendering that allows choosing data from a model and obtains built-in support of Ajax updates.

```

1. Chevrolet Corvette
   Price:16080
   Mileage:55773.0
2. Chevrolet Corvette
   Price:49936
   Mileage:72356.0
3. Chevrolet Corvette
   Price:52167
   Mileage:30749.0
4. Chevrolet Corvette
   Price:21148
   Mileage:55447.0
5. Chevrolet Corvette
   Price:18098
   Mileage:16296.0

```

Figure 6.31. <rich:dataOrderedList> component

6.33.2. Key Features

- A completely skinned list and child elements
- Possibility to update a limited set of rows with AJAX
- Possibility to receive values dynamically from a model

Table 6.103. rich : dataOrderedList attributes

| Attribute Name | Description |
|----------------|--|
| ajaxKeys | ajaxKeys |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| dir | Direction indication for text that does not inherit directionality. Valid values are "LTR" (left-to-right) and "RTL" (right-to-left) |
| first | A zero-relative row number of the first row to display |
| id | Every component may have a unique id that is automatically created if omitted |
| lang | Code describing the language used in the generated markup for this component |
| rendered | If "false", this component is not rendered |
| rowClasses | A comma-delimited list of CSS style classes that is applied to popup table rows. A space separated list of classes may also be specified for any individual row. The styles are applied, in turn, to each row in the table. For example, if the list has two elements, the first style class in the list is applied to the first row, the second to the second row, the first to the third row, the second to the fourth row, etc. In other words, we keep iterating through the list until we reach the end, and then we start at the beginning again |
| rowKeyVar | rowKeyVar |
| rows | A number of rows to display, or zero for all remaining rows in the table |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| title | Advisory title information about markup elements generated for this component |
| type | Corresponds to the HTML OL type attribute |
| value | value |
| var | A request-scope attribute via which the data object for the current row will be used when iterating |

Table 6.104. Component identification parameters

| Name | Value |
|------------------|--|
| component-type | org.richfaces.DataOrderedList |
| component-class | org.richfaces.component.html.HtmlDataOrderedList |
| component-family | org.richfaces.DataOrderedList |
| renderer-type | org.richfaces.DataOrderedListRenderer |
| tag-class | org.richfaces.taglib.DataOrderedListTag |

6.33.3. Creating the Component with a Page Tag

To create the simplest variant of `dataOrderedList` on a page, use the following syntax:

Example:

```
...
<rich:dataOrderedList var="car" value="#{dataTableScrollerBean.allCars}" >
  <h:outputText value="#{car.model}" />
</rich:dataOrderedList>
...
```

6.33.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlDataOrderedList;
...
HtmlDataOrderedList myList = new HtmlDataOrderedList();
...
```

6.33.5. Details of Usage

The `<rich:dataOrderedList>` component allows to generate an ordered list from a model.

The component has the *"type"* attribute, which corresponds to the *"type"* parameter for the *"OL"* HTML element and defines a marker type. Possible values for *"type"* attribute are: "A", "a", "I", "i", "1".

Here is an example:

```
...
<h:form>
  <rich:dataOrderedList var="car" value="#{dataTableScrollerBean.allCars}"
    rows="5" type="1" title="Car Store">
    <h:outputText value="#{car.make} #{car.model}" /><br/>
    <h:outputText value="Price:" styleClass="label"></h:outputText>
    <h:outputText value="#{car.price}" /><br/>
    <h:outputText value="Mileage:" styleClass="label"></h:outputText>
    <h:outputText value="#{car.mileage}" /><br/>
  </rich:dataOrderedList>
</h:form>
...
```

```

    </rich:dataOrderedList>
  </h:form>
  ...

```

This is a result:

```

1. Chevrolet Corvette
   Price:16080
   Mileage:55773.0
2. Chevrolet Corvette
   Price:49936
   Mileage:72356.0
3. Chevrolet Corvette
   Price:52167
   Mileage:30749.0
4. Chevrolet Corvette
   Price:21148
   Mileage:55447.0
5. Chevrolet Corvette
   Price:18098
   Mileage:16296.0

```

Figure 6.32. Component usage

In the example the *"rows"* attribute limits number of output elements of the list.

"first" attribute defines first element for output. *"title"* are used for popup title.

The component was created basing on the **<a4j:repeat>** component and as a result it could be partially updated with Ajax. *"ajaxKeys"* attribute allows to define rows that are updated after an Ajax request.

Here is an example:

Example:

```

...
<rich:dataOrderedList value="#{dataTableScrollerBean.allCars}" var="car"
  ajaxKeys="#{listBean.list}"
                    binding="#{listBean.dataList}" id="list">
  ...
</rich:dataOrderedList>
...
<a4j:commandButton action="#{listBean.action}" reRender="list" value="Submit"/>
...

```

In the example *"reRender"* attribute contains value of *"id"* attribute for **<rich:dataOrderedList>** component. As a result the component is updated after an Ajax request.

6.33.6. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

There are two ways to redefine the appearance of all **<rich:dataOrderedList>** components at once:

- Redefine the corresponding skin parameters
- Add to your style sheets *style classes* used by a `<rich:dataOrderedList>` component

6.33.7. Definition of Custom Style Classes

On the screenshot there are classes names that define styles for component elements.

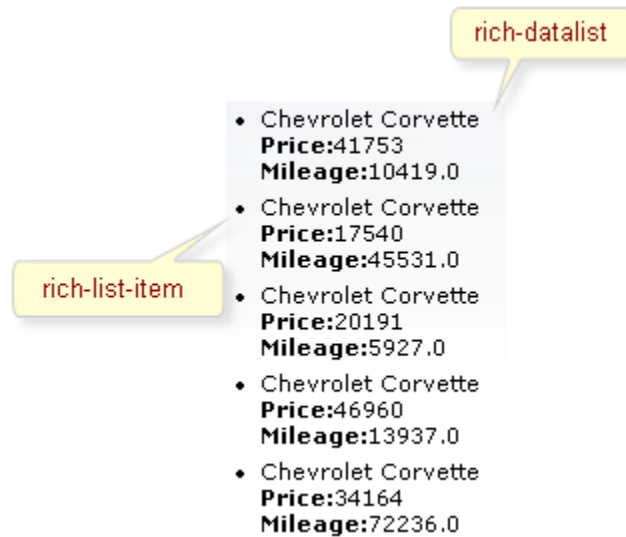


Figure 6.33. Style classes

Table 6.105. Classes names that define a list appearance

| Class name | Description |
|------------------|--|
| rich-orderedlist | Defines styles for an html <code></code> element |
| rich-list-item | Defines styles for an html <code></code> element |

In order to redefine styles for all `<rich:dataOrderedList>` components on a page using CSS, it's enough to create classes with the same names and define necessary properties in them.

To change styles of particular `<rich:dataOrderedList>` components, define your own style classes in the corresponding `<rich:dataOrderedList>` attributes.

6.33.8. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/dataLists.jsf?c=dataOrderedList>] you can see the example of `<rich:dataOrderedList>` usage and sources for the given example.

6.34. < rich:dataDefinitionList >

6.34.1. Description

The component for definition lists rendering that allows choosing data from a model and obtains built-in support of Ajax updates.


```

Chevrolet Corvette
  Price:18098
  Mileage:16296.0
Chevrolet Malibu
  Price:36523
  Mileage:46112.0
Chevrolet Malibu
  Price:33307
  Mileage:57709.0
Chevrolet Malibu
  Price:34248
  Mileage:62821.0
Chevrolet Malibu
  Price:51555
  Mileage:51549.0

```

Figure 6.34. <rich:dataDefinitionList> component

6.34.2. Key Features

- Completely skinned table rows and child elements
- Possibility to update a limited set of rows with AJAX
- Possibility to receive values dynamically from a model

Table 6.106. rich : dataDefinitionList attributes

| Attribute Name | Description |
|----------------|--|
| ajaxKeys | ajaxKeys |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| columnClasses | Comma-delimited list of CSS style classes that are be applied to the columns of this table. A space separated list of classes may also be specified for any individual column. If the number of elements in this list is less than the number of columns specified in the "columns" attribute, no "class" attribute is output for each column greater than the number of elements in the list. If the number of elements in the list is greater than the number of columns specified in the "columns" attribute, the elements at the position in the list after the value of the "columns" attribute are ignored |
| componentState | componentState |
| dir | Direction indication for text that does not inherit directionality. Valid values are "LTR" (left-to-right) and "RTL" (right-to-left) |
| first | A zero-relative row number of the first row to display |
| footerClass | |

| Attribute Name | Description |
|----------------|--|
| | Space-separated list of CSS style class(es) that are be applied to any footer generated for this table |
| headerClass | Space-separated list of CSS style class(es) that are be applied to any header generated for this table |
| id | Every component may have a unique id that is automatically created if omitted |
| lang | Code describing the language used in the generated markup for this component |
| rendered | If "false", this component is not rendered |
| rowClasses | A comma-delimited list of CSS style classes that is applied to popup table rows. A space separated list of classes may also be specified for any individual row. The styles are applied, in turn, to each row in the table. For example, if the list has two elements, the first style class in the list is applied to the first row, the second to the second row, the first to the third row, the second to the fourth row, etc. In other words, we keep iterating through the list until we reach the end, and then we start at the beginning again |
| rowKey | rowKey |
| rowKeyVar | rowKeyVar |
| rows | A number of rows to display, or zero for all remaining rows in the table |
| stateVar | The attribute provides access to a component state on the client side |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| title | Advisory title information about markup elements generated for this component |
| value | value |
| var | A request-scope attribute via which the data object for the current row will be used when iterating |

Table 6.107. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | org.richfaces.DataDefinitionList |
| component-class | org.richfaces.component.html.HtmlDataDefinitionList |
| component-family | org.richfaces.DataDefinitionList |
| renderer-type | org.richfaces.DataDefinitionListRenderer |
| tag-class | org.richfaces.taglib.DataDefinitionListTag |

6.34.3. Creating the Component with a Page Tag

To create the simplest variant of `dataDefinitionList` on a page, use the following syntax:

Example:

```
...
<rich:dataDefinitionList value="#{bean.capitals}" var="caps">
  <f:facet name="term">Cars</f:facet>
  <h:outputText value="#{car.model}" />
</rich:dataDefinitionList>
...
```

6.34.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlDataDefinitionList;
...
HtmlDataDefinitionList myList = new HtmlDataDefinitionList();
...
```

6.34.5. Details of Usage

The `<rich:dataDefinitionList>` component allows to generate an definition list from a model.

The component has the *"term"* facet, which corresponds to the *"type"* parameter for the *"DT"* HTML element.

Here is an example:

```
...
<h:form>
  <rich:dataDefinitionList var="car" value="#{dataTableScrollerBean.allCars}"
    rows="5" first="4" title="Cars">
    <f:facet name="term">
      <h:outputText value="#{car.make} #{car.model}"></h:outputText>
    </f:facet>
    <h:outputText value="Price:" styleClass="label"></h:outputText>
    <h:outputText value="#{car.price}" /><br/>
    <h:outputText value="Mileage:" styleClass="label"></h:outputText>
  </rich:dataDefinitionList>
</h:form>
```

```

        <h:outputText value="#{car.mileage}" /><br/>
    </rich:dataDefinitionList>
</h:form>
...

```

This is a result:

```

Chevrolet Corvette
  Price:18098
  Mileage:16296.0
Chevrolet Malibu
  Price:36523
  Mileage:46112.0
Chevrolet Malibu
  Price:33307
  Mileage:57709.0
Chevrolet Malibu
  Price:34248
  Mileage:62821.0
Chevrolet Malibu
  Price:51555
  Mileage:51549.0

```

Figure 6.35. Component usage

In the example the *"rows"* attribute limits number of output elements of the list.

"first" attribute defines first element for output. *"title"* are used for popup title.

The component was created basing on the **<a4j:repeat>** component and as a result it could be partially updated with Ajax. *"ajaxKeys"* attribute allows to define rows that are updated after an Ajax request.

Here is an example:

Example:

```

...
<rich:dataDefinitionList value="#{dataTableScrollerBean.allCars}" var="car"
  ajaxKeys="#{listBean.list}"
    binding="#{listBean.dataList}" id="list">
    ...
</rich:dataDefinitionList>
...
<a4j:commandButton action="#{listBean.action}" reRender="list" value="Submit"/>
...

```

In the example *"reRender"* attribute contains value of *"id"* attribute for **<rich:dataDefinitionList>** component. As a result the component is updated after an Ajax request.

6.34.6. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

There are two ways to redefine the appearance of all **<rich:dataDefinitionList>** components at once:

- Redefine the corresponding skin parameters

- Add to your style sheets *style classes* used by a `<rich:dataDefinitionList>` component

6.34.7. Definition of Custom Style Classes

On the screenshot there are classes names that define styles for component elements.

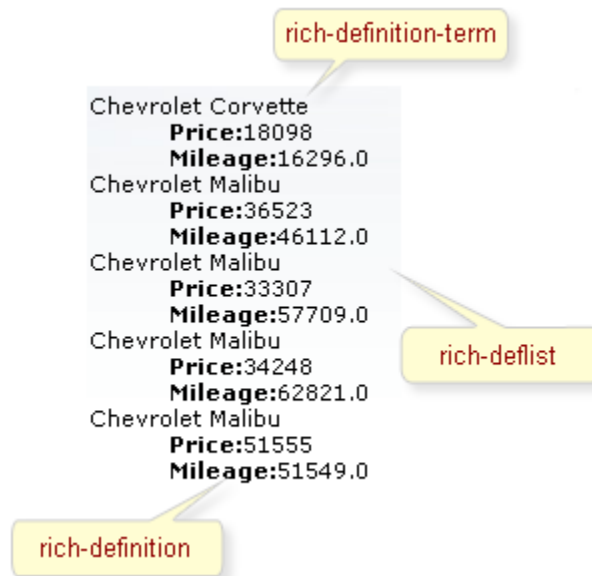


Figure 6.36. Style classes

Table 6.108. Classes names that define a list appearance

| Class name | Description |
|----------------------|--|
| rich-deflist | Defines styles for an html <code><dl></code> element |
| rich-definition | Defines styles for an html <code><dd></code> element |
| rich-definition-term | Defines styles for an html <code><dt></code> element |

In order to redefine styles for all `<rich:dataDefinitionList>` components on a page using CSS, it's enough to create classes with the same names and define necessary properties in them.

To change styles of particular `<rich:dataDefinitionList>` components, define your own style classes in the corresponding `<rich:dataDefinitionList>` attributes.

6.34.8. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/dataLists.jsf?c=dataDefinitionList>] you can see the example of `<rich:dataDefinitionList>` usage and sources for the given example.

6.35. < rich:dataTable >

6.35.1. Description

The component for tables rendering that allows choosing data from a model and obtains built-in support of Ajax updates.

United States Capitals

| Capitals and States Table | | | |
|---|--------------|------------|----------|
| State Flag | Capital Name | State Name | TimeZone |
|  | Montgomery | Alabama | GMT-6 |
|  | Juneau | Alaska | GMT-9 |
|  | Phoenix | Arizona | GMT-7 |
|  | Little Rock | Arkansas | GMT-6 |
|  | Sacramento | California | GMT-8 |
| State Flag | Capital Name | State Name | TimeZone |
| Capitals and States Table | | | |

Figure 6.37. <rich:dataTable> component

6.35.2. Key Features

- A completely skinned table and child elements
- Possibility to insert the complex subcomponents "colGroup" and "subTable"
- Possibility to update a limited set of strings with AJAX

Table 6.109. rich : dataTable attributes

| Attribute Name | Description |
|----------------|---|
| ajaxKeys | This attribute defines rows that are updated after an AJAX request |
| align | left center right [CI] Deprecated. This attribute specifies the position of the table with respect to the document. Permitted values: * left: The table is to the left of the document. * center: The table is to the center of the document. * right: The table is to the right of the document |
| bgcolor | Deprecated. This attribute sets the background color for the document body or table cells. This attribute sets the background color of the canvas for the document body (the BODY element) or for tables (the TABLE, TR, TH, and TD elements). Additional attributes for specifying text color can be used with the BODY element. This attribute has been deprecated in favor of style sheets for specifying background color information |
| binding | |

| Attribute Name | Description |
|----------------|---|
| | The attribute takes a value-binding expression for a component property of a backing bean |
| border | This attribute specifies the width of the frame around a component |
| captionClass | Space-separated list of CSS style class(es) that are applied to caption for this component |
| captionStyle | CSS style(s) is/are to be applied to caption when this component is rendered |
| cellpadding | This attribute specifies the amount of space between the border of the cell and its contents |
| cellspacing | This attribute specifies the amount of space between the border of the cell and its contents. The attribute also specifies the amount of space to leave between cells |
| columnClasses | Comma-delimited list of CSS style classes that are applied to the columns of this table. A space separated list of classes may also be specified for any individual column. If the number of elements in this list is less than the number of columns specified in the "columns" attribute, no "class" attribute is output for each column greater than the number of elements in the list. If the number of elements in the list is greater than the number of columns specified in the "columns" attribute, the elements at the position in the list after the value of the "columns" attribute are ignored |
| columns | Quantity of columns |
| columnsWidth | Comma-separated list of width attribute for every column. Specifies a default width for each column in the table. In addition to the standard pixel, percentage, and relative values, this attribute allows the special form "0*" (zero asterisk) which means that the width of the each column in the group should be the minimum width necessary to hold the column's contents. This implies that a column's entire contents must be known before its width may be correctly computed. Authors should be aware that specifying "0*" will prevent visual user agents from rendering a table incrementally |
| componentState | It defines EL-binding for a component state for saving or redefinition |

| Attribute Name | Description |
|----------------|--|
| dir | Direction indication for text that does not inherit directionality. Valid values are "LTR" (left-to-right) and "RTL" (right-to-left) |
| first | A zero-relative row number of the first row to display |
| footerClass | Space-separated list of CSS style class(es) that are be applied to footer for this component |
| frame | void above below hsides lhs rhs vsides box border [CI] This attribute specifies which sides of the frame surrounding a table will be visible. Possible values: * void: No sides. This is the default value. * above: The top side only. * below: The bottom side only. * hsides: The top and bottom sides only. * vsides: The right and left sides only. * lhs: The left-hand side only. * rhs: The right-hand side only. * box: All four sides. * border: All four sides |
| headerClass | Space-separated list of CSS style class(es) that are be applied to header for this component |
| id | Every component may have a unique id that is automatically created if omitted |
| lang | Code describing the language used in the generated markup for this component |
| onclick | HTML: a script expression; a pointer button is clicked |
| ondblclick | HTML: a script expression; a pointer button is double-clicked |
| onkeydown | HTML: a script expression; a key is pressed down |
| onkeypress | HTML: a script expression; a key is pressed and released |
| onkeyup | HTML: a script expression; a key is released |
| onmousedown | HTML: script expression; a pointer button is pressed down |
| onmousemove | HTML: a script expression; a pointer is moved within |
| onmouseout | HTML: a script expression; a pointer is moved away |
| onmouseover | HTML: a script expression; a pointer is moved onto |
| onmouseup | HTML: script expression; a pointer button is released |

| Attribute Name | Description |
|----------------|--|
| onRowClick | HTML: a script expression; a pointer button is clicked on row |
| onRowDbClick | HTML: a script expression; a pointer button is double-clicked on row |
| onRowMouseDown | HTML: script expression; a pointer button is pressed down on row |
| onRowMouseMove | HTML: a script expression; a pointer is moved within of row |
| onRowMouseOut | HTML: a script expression; a pointer is moved away of row |
| onRowMouseOver | HTML: a script expression; a pointer is moved onto of row |
| onRowMouseUp | HTML: script expression; a pointer button is released on row |
| rendered | If "false", this component is not rendered |
| rowClasses | A comma-delimited list of CSS style classes that is applied to popup table rows. A space separated list of classes may also be specified for any individual row. The styles are applied, in turn, to each row in the table. For example, if the list has two elements, the first style class in the list is applied to the first row, the second to the second row, the first to the third row, the second to the fourth row, etc. In other words, we keep iterating through the list until we reach the end, and then we start at the beginning again |
| rowKey | RowKey is a representation of an identifier for a specific data row |
| rowKeyVar | The attribute provides access to a row key in a Request scope |
| rows | A number of rows to display, or zero for all remaining rows in the table |
| rules | This attribute specifies which rules will appear between cells within a table. The rendering of rules is user agent dependent. Possible values: * none: No rules. This is the default value. * groups: Rules will appear between row groups (see THEAD, TFOOT, and TBODY) and column groups (see COLGROUP and |

| Attribute Name | Description |
|----------------|---|
| | COL) only. * rows: Rules will appear between rows only. * cols: Rules will appear between columns only. * all: Rules will appear between all rows and columns |
| stateVar | The attribute provides access to a component state on the client side |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| summary | This attribute provides a summary of the table's purpose and structure for user agents rendering to non-visual media such as speech and Braille |
| title | Advisory title information about markup elements generated for this component |
| value | The current value for this component |
| var | A request-scope attribute via which the data object for the current row will be used when iterating |
| width | This attribute specifies the desired width of the entire table and is intended for visual user agents. When the value is percentage value, the value is relative to the user agent's available horizontal space. In the absence of any width specification, table width is determined by the user agent |

Table 6.110. Component identification parameters

| Name | Value |
|------------------|--|
| component-type | org.richfaces.DataTable |
| component-class | org.richfaces.component.html.HtmlDataTable |
| component-family | org.richfaces.DataTable |
| renderer-type | org.richfaces.DataTableRenderer |
| tag-class | org.richfaces.taglib.DataTableTag |

6.35.3. Creating the Component with a Page Tag

Here is a simple example as it could be used on a page:

Example:

```
...
```

```

<rich:dataTable value="#{capitalsBean.capitals}" var="capitals">
    <rich:column>
        ...
    </rich:column>
</rich:dataTable>
...

```

6.35.4. Creating the Component Dynamically from Java

Example:

```

import org.richfaces.component.html.HtmlDataTable;
...
HtmlDataTable myTable = new HtmlDataTable();
...

```

6.35.5. Details of Usage

The **<rich:dataTable>** component is similar to the **<h:dataTable>** one, except Ajax support and skinnability. Ajax support is possible, because the component was created basing on the **<a4j:repeat>** component and as a result it could be partially updated with Ajax. *"ajaxKeys"* attribute allows to define rows that is updated after an Ajax request.

Here is an example:

Example:

```

...
<rich:dataTable value="#{capitalsBean.capitals}" var="capitals"
    ajaxKeys="#{bean.ajaxSet}" binding="#{bean.table}" id="table">
    ...
</rich:dataTable>
...
<a4j:commandButton action="#{tableBean.action}" reRender="table" value="Submit"/>
...

```

In the example *"reRender"* attribute contains value of *"id"* attribute for **<rich:dataTable>** component. As a result the component is updated after an Ajax request.

The component allows to use *"header"*, *"footer"* and *"caption"* facets for output. See an example below:

Example:

```

...
<rich:dataTable value="#{capitalsBean.capitals}" var="cap" rows="5">
    <f:facet name="caption"><h:outputText value="United States Capitals"
/></f:facet>
    <f:facet name="header"><h:outputText value="Capitals and States Table"
/></f:facet>
    <rich:column>
        <f:facet name="header">State Flag</f:facet>
        <h:graphicImage value="#{cap.stateFlag}"/>
        <f:facet name="footer">State Flag</f:facet>
    </rich:column>

```

```

</rich:column>
<rich:column>
    <f:facet name="header">State Name</f:facet>
    <h:outputText value="#{cap.state}"/>
    <f:facet name="footer">State Name</f:facet>
</rich:column>
<rich:column>
    <f:facet name="header">State Capital</f:facet>
    <h:outputText value="#{cap.name}"/>
    <f:facet name="footer">State Capital</f:facet>
</rich:column>
<rich:column>
    <f:facet name="header">Time Zone</f:facet>
    <h:outputText value="#{cap.timeZone}"/>
    <f:facet name="footer">Time Zone</f:facet>
</rich:column>
    <f:facet name="footer"><h:outputText value="Capitals and States Table"
/></f:facet>
</rich:dataTable>
...

```

This is a result:






| | | | |
|---|--------------|------------|----------|
| United States Capitals | | | |
| Capitals and States Table | | | |
| | Capital Name | State Name | TimeZone |
|  | Montgomery | Alabama | GMT-6 |
|  | Juneau | Alaska | GMT-8 |
|  | Phoenix | Arizona | GMT-7 |
|  | Little Rock | Arkansas | GMT-6 |
|  | Sacramento | California | GMT-8 |
| State Flag | Capital Name | State Name | TimeZone |
| Capitals and States Table | | | |

Figure 6.38. Component usage

6.35.6. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

There are two ways to redefine the appearance of all `<rich:dataTable>` components at once:

- Redefine the corresponding skin parameters
- Add to your style sheets *style classes* used by a `<rich:dataTable>` component

6.35.7. Skin Parameters Redefinition

Table 6.111. Skin parameters redefinition for a table

| Skin parameters | CSS properties |
|----------------------|------------------|
| tableBackgroundColor | background-color |

Table 6.112. Skin parameters redefinition for a header

| Skin parameters | CSS properties |
|-----------------------|------------------|
| headerBackgroundColor | background-color |

Table 6.113. Skin parameters redefinition for a footer

| Skin parameters | CSS properties |
|----------------------------|------------------|
| tableFooterBackgroundColor | background-color |

Table 6.114. Skin parameters redefinition for a column header

| Skin parameters | CSS properties |
|---------------------------|------------------|
| additionalBackgroundColor | background-color |

Table 6.115. Skin parameters redefinition for a column footer

| Skin parameters | CSS properties |
|-------------------------------|------------------|
| tableSubfooterBackgroundColor | background-color |

Table 6.116. Skin parameters redefinition for cells

| Skin parameters | CSS properties |
|-------------------|----------------|
| generalSizeFont | font-size |
| generalTextColor | color |
| generalFamilyFont | font-family |

6.35.8. Definition of Custom Style Classes

On the screenshot there are classes names that define styles for component elements.

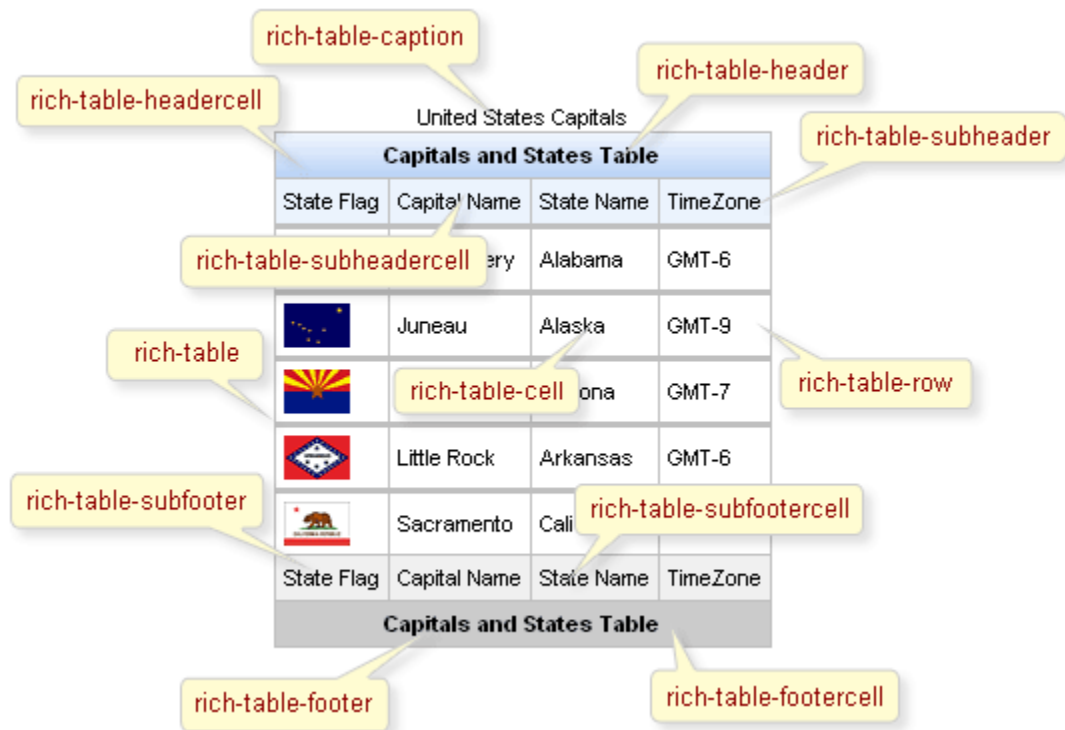
Figure 6.39. `<rich:dataTable>` class names

Table 6.117. Classes names that define a whole component appearance

| Class name | Description |
|---------------------------------|--|
| <code>rich-table</code> | Defines styles for all table |
| <code>rich-table-caption</code> | Defines styles for a "caption" facet element |

Table 6.118. Classes names that define header and footer elements

| Class name | Description |
|---|---|
| <code>rich-table-header</code> | Defines styles for a table header row |
| <code>rich-table-header-continue</code> | Defines styles for all header lines after the first |
| <code>rich-table-subheader</code> | Defines styles for a column header |
| <code>rich-table-footer</code> | Defines styles for a footer row |
| <code>rich-table-footer-continue</code> | Defines styles for all footer lines after the first |
| <code>rich-table-subfooter</code> | Defines styles for a column footer |

Table 6.119. Classes names that define rows and cells of a table

| Class name | Description |
|---------------------------------------|---|
| <code>rich-table-headercell</code> | Defines styles for a header cell |
| <code>rich-table-subheadercell</code> | Defines styles for a column header cell |

| Class name | Description |
|--------------------------|---|
| rich-table-cell | Defines styles for a table cell |
| rich-table-row | Defines styles for a table row |
| rich-table-firstrow | Defines styles for a table start row |
| rich-table-footercell | Defines styles for a footer cell |
| rich-table-subfootercell | Defines styles for a column footer cell |

In order to redefine styles for all **<rich:dataTable>** components on a page using CSS, it's enough to create classes with the same names and define necessary properties in them.

To change styles of particular **<rich:dataTable>** components, define your own style classes in the corresponding **<rich:dataTable>** attributes.

6.35.9. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/dataTable.jsf?c=dataTable>] you can see the example of **<rich:dataTable>** usage and sources for the given example.

The article about **<rich:dataTable>** flexibility can be found here [<http://labs.jboss.com/wiki/RichFacesArticleDataTable>].

More information about using **<rich:dataTable>** and **<rich:subTable>** could be found on the RichFaces Users Forum. [<http://www.jboss.com/index.html?module=bb&op=viewtopic&p=4059044#4059044>]

How to use **<rich:dataTable>** and **<rich:datascroller>** in a context of Extended Data Model see here [<http://www.jboss.com/index.html?module=bb&op=viewtopic&t=115636>].

6.36. < rich:subTable >

6.36.1. Description

The component is used for inserting subtables into tables with opportunity to choose data from a model and built-in Ajax updates support.





| Countries And Capitals | | | |
|---|------------|---------------|----------|
| Country | | | |
| United States | | | |
| State Flag | State Name | State Capital | Timezone |
|  | Alabama | Montgomery | GMT-6 |
|  | Alaska | Juneau | GMT-9 |
|  | Arizona | Phoenix | GMT-7 |
|  | Arkansas | Little Rock | GMT-6 |
|  | California | Sacramento | GMT-8 |

Figure 6.40. <rich:subTable> element

6.36.2. Key Features

- Completely skinned table rows and child elements
- Possibility to insert complex columnGroup subcomponents
- Possibility to combine rows and columns inside
- Possibility to update a limited set of rows with AJAX

Table 6.120. rich : subTable attributes

| Attribute Name | Description |
|----------------|--|
| ajaxKeys | This attribute defines rows that are updated after an AJAX request |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| columnClasses | Comma-delimited list of CSS style classes that are be applied to the columns of this table. A space separated list of classes may also be specified for any individual column. If the number of elements in this list is less than the number of columns specified in the "columns" attribute, no "class" attribute is output for each column greater than the number of elements in the list. If the number of elements in the list is greater than the number of columns specified in the "columns" attribute, the elements at the position in the list after the value of the "columns" attribute are ignored |
| componentState | |

| Attribute Name | Description |
|----------------|--|
| | It defines EL-binding for a component state for saving or redefinition |
| first | A zero-relative row number of the first row to display |
| footerClass | Space-separated list of CSS style class(es) that are be applied to any footer generated for this table |
| headerClass | Space-separated list of CSS style class(es) that are be applied to any header generated for this table |
| id | Every component may have a unique id that is automatically created if omitted |
| onclick | HTML: a script expression; a pointer button is clicked |
| ondblclick | HTML: a script expression; a pointer button is double-clicked |
| onkeydown | HTML: a script expression; a key is pressed down |
| onkeypress | HTML: a script expression; a key is pressed and released |
| onkeyup | HTML: a script expression; a key is released |
| onmousedown | HTML: script expression; a pointer button is pressed down |
| onmousemove | HTML: a script expression; a pointer is moved within |
| onmouseout | HTML: a script expression; a pointer is moved away |
| onmouseover | HTML: a script expression; a pointer is moved onto |
| onmouseup | HTML: script expression; a pointer button is released |
| onRowClick | HTML: a script expression; a pointer button is clicked on row |
| onRowDbClick | HTML: a script expression; a pointer button is double-clicked on row |
| onRowMouseDown | HTML: script expression; a pointer button is pressed down on row |
| onRowMouseMove | HTML: a script expression; a pointer is moved within of row |
| onRowMouseOut | HTML: a script expression; a pointer is moved away of row |

| Attribute Name | Description |
|----------------|--|
| onRowMouseOver | HTML: a script expression; a pointer is moved onto of row |
| onRowMouseUp | HTML: script expression; a pointer button is released on row |
| rendered | If "false", this component is not rendered |
| rowClasses | A comma-delimited list of CSS style classes that is applied to popup table rows. A space separated list of classes may also be specified for any individual row. The styles are applied, in turn, to each row in the table. For example, if the list has two elements, the first style class in the list is applied to the first row, the second to the second row, the first to the third row, the second to the fourth row, etc. In other words, we keep iterating through the list until we reach the end, and then we start at the beginning again |
| rowKey | RowKey is a representation of an identifier for a specific data row |
| rowKeyVar | The attribute provides access to a row key in a Request scope |
| rows | A number of rows to display, or zero for all remaining rows in the table |
| sortExpression | sortExpression |
| stateVar | The attribute provides access to a component state on the client side |
| value | The current value for this component |
| var | A request-scope attribute via which the data object for the current row will be used when iterating |

Table 6.121. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | org.richfaces.SubTable |
| component-class | org.richfaces.component.html.HtmlSubTable |
| component-family | org.richfaces.SubTable |
| renderer-type | org.richfaces.SubTableRenderer |
| tag-class | org.richfaces.taglib.SubTableTag |

6.36.3. Creating the Component with a Page Tag

Here is a simple example as it could be used on a page:

Example:

```
...
    <rich:dataTable value="#{capitalsBean.capitals}" var="capitals">
        <rich:column>
            ...
        </rich:column>
        <rich:subTable value=#{capitals.details} var="detail">
            <rich:column>
                ...
            </rich:column>
        </rich:subTable>
    </rich:dataTable>
...
```

6.36.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlSubTable;
...
HtmlSubTable mySubTable = new HtmlSubTable();
...
```

6.36.5. Details of Usage

The `<rich:subTable>` component is similar to the `<h:dataTable>` one, except Ajax support and skinnability. One more difference is that the component doesn't add the wrapping `<table>` and `<tbody>` tags. Ajax support is possible, because the component was created basing on the `<a4j:repeat>` component and as a result it could be partially updated with Ajax. *"ajaxKeys"* attribute allows to define rows that is updated after an Ajax request.

Here is an example:

Example:

```
...
    <rich:dataTable value="#{capitalsBean.capitals}" var="capitals">
        <rich:column>
            ...
        </rich:column>
        <rich:subTable value="#{capitals.details}" var="detail"
ajaxKeys="#{bean.ajaxSet}" binding="#{bean.subtable}" id="subtable">
            <rich:column>
                ...
            </rich:column>
        </rich:subTable>
    </rich:dataTable>
...
    <a4j:commandButton action="#{tableBean.action}" reRender="subtable"/>
```

...

In the example *"reRender"* attribute contains value of *"id"* attribute for **<rich:subTable>** component. As a result the component is updated after an Ajax request.

The component allows to use *"header"* and *"footer"* facets for output. See an example for **<rich:dataTable>** component [193].

6.36.6. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

There are two ways to redefine the appearance of all **<rich:subTable>** components at once:

- Redefine the corresponding skin parameters
- Add to your style sheets *style classes* used by a **<rich:subTable>** component

6.36.7. Skin Parameters Redefinition

Skin parameters redefinition for **<rich:subTable>** are the same as for the **<rich:dataTable>** component.

6.36.8. Definition of Custom Style Classes

Table 6.122. Classes names that define a component appearance

| Class name | Description |
|-----------------------|--|
| rich-subtable | Defines styles for all subtable |
| rich-subtable-caption | Defines styles for a "caption" facet element |

Table 6.123. Classes names that define header and footer elements

| Class name | Description |
|-------------------------------|--|
| rich-subtable-header | Defines styles for a subtable header row |
| rich-subtable-header-continue | Defines styles for all subtable header lines after the first |
| rich-subtable-subheader | Defines styles for a column header of subtable |
| rich-subtable-subfooter | Defines styles for a column footer of subtable |
| rich-subtable-footer | Defines styles for a subtable footer row |
| rich-subtable-footer-continue | Defines styles for all subtable footer lines after the first |

Table 6.124. Classes names that define rows and cells

| Class name | Description |
|--------------------------|---|
| rich-subtable-headercell | Defines styles for a subtable header cell |

| Class name | Description |
|-----------------------------|---|
| rich-subtable-subheadercell | Defines styles for a column header cell of subtable |
| rich-subtable-cell | Defines styles for a subtable cell |
| rich-subtable-row | Defines styles for a subtable row |
| rich-subtable-firstrow | Defines styles for a subtable start row |
| rich-subtable-subfootercell | Defines styles for a column footer cell of subtable |
| rich-subtable-footercell | Defines styles for a subtable footer cell |

In order to redefine styles for all `<rich:subTable>` components on a page using CSS, it's enough to create classes with the same names and define necessary properties in them.

To change styles of particular `<rich:subTable>` components, define your own style classes in the corresponding `<rich:subTable>` attributes.

6.37. < rich:dndParam >

6.37.1. Description

This component is used for passing parameters during drag-and-drop operations.

Table 6.125. rich : dndParam attributes

| Attribute Name | Description |
|----------------|--|
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| id | Every component may have a unique id that is automatically created if omitted |
| name | A name of this parameter |
| rendered | If "false", this component is not rendered |
| type | This attribute defines parameter functionality. Possible values are "drag", "drop" and "default" |
| value | The current value for this component |

Table 6.126. Component identification parameters

| Name | Value |
|-----------------|---|
| component-type | org.richfaces.DndParam |
| component-class | org.richfaces.component.html.HtmlDndParam |
| tag-class | org.richfaces.taglib.DndParamTag |

6.37.2. Creating the Component with a Page Tag

Here is a simple example as it could be used on a page, nested in one of the drag-and-drop components:

Example:

```
...
<rich:dragSupport dragType="file">
    <rich:dndParam name="testDrag" value="testDragValue"
                    type="drag" />
</rich:dragSupport>
...
```

6.37.3. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlDndParam;
...
HtmlDndParam myDparam = new HtmlDndParam();
...
```

6.37.4. Details of Usage

dndParam is used during drag-and-drop operations to pass parameters to an indicator. At first, a parameter type is defined with the type attribute (to specify parameter functionality), then a parameter name could be defined with the name and value attribute. Although, it's possible to use nested content defined inside dndParam for value definition, instead of the attribute.

Variants of usage:

- Parameters passing for a drag icon when an indicator is in drag.

In this case, dndParam is of a drag type and is defined in the following way:

Example:

```
...
<rich:dragSupport ...>
    <rich:dndParam type="drag" name="dragging">
        <h:graphicImage value="/img/product1_small.png" />
    </rich:dndParam>
    <h:graphicImage value="product1.png" />
</rich:dragSupport>
...
```

Here dndParam defines an icon that is used by an indicator when a drag is on the place of a default icon (e.g. a minimized image of a draggable element)

- Parameters passing for an indicator informational part during a drag.

In this case dndParam is of a drag type and is defined in the following way:

Example:

```
...
<rich:dragSupport ...>
    <rich:dndParam type="drag" name="label" value="#{msg.subj}" />
    ...
</rich:dragSupport>
...
```

The parameter is transmitted into an indicator for usage in an informational part of the dragIndicator component (inside an indicator a call to {label} happens)

- Parameters passing happens when dragged content is brought onto some zone with dropSupport

In this case dndParam is of a drop type and is defined in the following way:

Example:

```
...
<rich:dropSupport ...>
    <rich:dndParam type="drop" name="comp" >
        <h:graphicImage height="16" width="16" value="/images/comp.png" />
    </rich:dndParam>
    ...
</rich:dropSupport >
...
```

Here, dndParam passes icons into an indicator, if dragged content of a comp type is above the given drop zone that processes it on the next drop event.

6.37.5. Look-and-Feel Customization

<rich:dndParam> has no skin parameters and custom style classes, as the component isn't visual.

6.37.6. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/dragSupport.jsf?c=dndParam>] you can see the example of **<rich:dndParam>** usage and sources for the given example.

6.38. < rich:dragIndicator >

6.38.1. Description

This is a component for defining what appears under the mouse cursor during drag-and-drop operations. The displayed drag indicator can show information about the dragged elements.

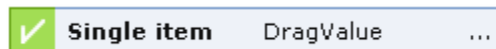


Figure 6.41. <rich:dragIndicator> component

6.38.2. Key Features

- Customizable look and feel
- Customizable marker according to the type of draggable elements

Table 6.127. rich : dragIndicator attributes

| Attribute Name | Description |
|----------------|---|
| acceptClass | Corresponds to the HTML class attribute and added to an indicator when a drop is accepted |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| id | Every component may have a unique id that is automatically created if omitted |
| rejectClass | Corresponds to the HTML class attribute and added to an indicator when a drop is rejected |
| rendered | If "false", this component is not rendered |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |

Table 6.128. Component identification parameters

| Name | Value |
|------------------|--|
| component-type | org.richfaces.Draggable |
| component-class | org.richfaces.component.html.HtmlDragIndicator |
| component-family | org.richfaces.DragIndicator |
| renderer-type | org.richfaces.DragIndicatorRenderer |
| tag-class | org.richfaces.taglib.DragIndicatorTag |

6.38.3. Creating the Component with a Page Tag

Here is a simple example as it could be used on a page:

Example:

```
...
<dnd:dragIndicator id="indicator">
  <f:facet name="single">
    <f:verbatim>
      <b>Single item</b> {DragInfo}
    </f:verbatim>
  </f:facet>
</dnd:dragIndicator>
```



```

        </f:verbatim>
    </f:facet>
</dnd:dragIndicator>
...
<dnd:dragSupport dragType="text" dragIndicator="indicator">
...

```

6.38.4. Creating the Component Dynamically Using Java

Example:

```

import org.richfaces.component.html.HtmlDragIndicator;
...
HtmlDragIndicator myDragIndicator = new HtmlDragIndicator();
...

```

6.38.5. Details of Usage

In the simplest way the component could be defined empty - in that case a default indicator will be shown like this:



Figure 6.42. Simplest <rich:dragIndicator>

For indicator customization you need to define one of the following facets:

- single

Indicator shown when dragging a single element.

- multy

Indicator shown when dragging several components (for future components that will support multiple selection).

Thus for specify a look-and-feel you have to define one of these facets and include into it a content that should be shown in indicator.

6.38.5.1. Macro defenitions

To place some data from drag or drop zones into component you can use macro defenitions. They are being defining in the following way:

- **<rich:dndParam>** component with a specific name and value is being included into a drag/drop support component (an image can be defined as placed inside **<rich:dndParam>** without defining a value).

- in needed place a parameter value is included into the marking of indicator using syntax (name of parameter)

For instance, this:

```
...
<dnd:dropSupport...>
  <dnd:dndParam name="testDrop">
    <h:graphicImage value="/images/file-manager.png" />
  </dnd:dndParam>
</dnd:dropSupport>
...
```

..Is placed into indicator as follows:

```
...
<f:facet name="single">
  {testDrop}
</f:facet>
...
```

6.38.5.2. Predefined macro defenitions

Indicator can accept two default macro defenitions:

- marker
- label

Thus including one of these elements in the marking of indicator, in other words after setting up appropriate parameters in DnD components and defining only default indicator - without specifying facets - a developer gets these parameters values displayed in indicator in the order "marker - label".

6.38.5.3. Marker customization

The macro defenition "*marker*" can be customized depending on what a draggable element is located over. For that you should define one of these three parameters (specify a parameter with one of three names):

- accept

Parameter will be set instead of {marker} into indicator when a draggable element is positioned over drop zone that accept this type of elements

- reject

Parameter will be set instead of {marker} into indicator when a draggable element is positioned over drop zone that doesn't accept this type of elements

- default

Parameter will be set instead of {marker} into indicator when a draggable element is positioned over all the rest of page elements

6.38.6. Look-and-Feel Customization

The `<rich:dragIndicator>` component has no skin parameters and special *style classes*, as it consists of one element generated with a your method on the server. To define some style properties such as an indent or a border, it's possible to use *"style"* and *"styleClass"* attributes on the component.

6.38.7. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/dragSupport.jsf?c=dragIndicator>] you can see the example of `<rich:dragIndicator>` usage and sources for the given example.

6.39. < rich:dragSupport >

6.39.1. Description

This component defines a subtree of the component tree as draggable for drag-and-drop operations. Within such a "drag zone," you can click the mouse button on an item and drag it to any component that supports drop operations (a "drop zone"). It encodes all the necessary JavaScript for supporting drag-and-drop operations.



Figure 6.43. `<rich:dragSupport>` component

6.39.2. Key Features

- Encodes all necessary JavaScript to perform drag actions
- Can be used within any component type that provides the required properties for drag operations
- Supports drag-and-drop between different forms

Table 6.129. rich : dragSupport attributes

| Attribute Name | Description |
|----------------|---|
| action | MethodBinding pointing at the application action to be invoked, if this UIComponent is activated by the user, during the Apply Request Values or Invoke Application phase of the request processing lifecycle, depending on the value of the immediate property |
| actionListener | MethodBinding pointing at method accepting an ActionEvent with return type void |
| ajaxSingle | if "true", submits ONLY one field/link, instead of all form controls |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| bypassUpdates | If "true", after process validations phase it skips updates of model beans on a force render response. It can be used for validating components input |
| data | Serialized (on default with JSON) data passed on the client by a developer on AJAX request. It's accessible via "data.foo" syntax |
| disableDefault | Disable default action for target event (append "return false;" to JavaScript) |
| dragIndicator | Id of the dragIndicator component used as drag operation cursor |
| dragListener | MethodBinding representing an action listener method that will be notified after drag operation |
| dragType | Key of a drag object. It's used to define a necessity of processing the current dragged element on the drop zone side |
| dragValue | Data to be sent to the drop zone after a drop event |
| eventsQueue | Name of requests queue to avoid send next request before complete other from same event. Can be used to reduce number of requests of frequently events (key press, mouse move etc.) |
| focus | id of element to set focus after request completed on client side |
| id | Every component may have a unique id that is automatically created if omitted |

| Attribute Name | Description |
|--------------------|--|
| ignoreDupResponses | Attribute allows to ignore an Ajax Response produced by a request if the newest 'similar' request is in a queue already. ignoreDupResponses="true" does not cancel the request while it is processed on the server, but just allows to avoid unnecessary updates on the client side if the response isn't actual now |
| immediate | True means, that the default ActionListener should be executed immediately (i.e. during Apply Request Values phase of the request processing lifecycle), rather than waiting until the Invoke Application phase |
| limitToList | If "true", updates on client side ONLY elements from this 'reRender' property. If "false" (default) updates all rendered by ajax region components |
| onbeforedomupdate | JavaScript code for call before DOM has been updated on client side |
| oncomplete | JavaScript code for call after request completed on client side |
| ondragend | A JavaScript event handler called after a drag operation |
| ondragstart | A JavaScript event handler called before drag object |
| onsubmit | JavaScript code for call before submission of ajax event |
| rendered | If "false", this component is not rendered |
| requestDelay | Attribute defines the time (in ms.) that the request will be wait in the queue before it is ready to send. When the delay time is over, the request will be sent to the server or removed if the newest 'similar' request is in a queue already |
| reRender | Id['s] (in format of call UIComponent.findComponent()) of components, rendered in case of AjaxRequest caused by this component. Can be single id, comma-separated list of Id's, or EL Expression with array or Collection |
| status | ID (in format of call UIComponent.findComponent()) of Request status component |
| timeout | Response waiting time on a particular request. If a response is not received during this time, the request is aborted |

| Attribute Name | Description |
|----------------|--------------------------------------|
| value | The current value for this component |

Table 6.130. Component identification parameters

| Name | Value |
|------------------|--|
| component-type | org.richfaces.DragSupport |
| component-class | org.richfaces.component.html.HtmlDragSupport |
| component-family | org.richfaces.DragSupport |
| renderer-type | org.richfaces.DragSupportRenderer |
| tag-class | org.richfaces.taglib.DragSupportTag |

6.39.3. Creating the Component with a Page Tag

Here is a simple example as it could be used on a page:

Example:

```
...
<h:panelGrid id="drag1">
    <rich:dragSupport dragType="item"/>
        <!--Some content to be dragged-->
</h:panelGrid>
...
```

6.39.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlDragSupport;
...
HtmlDragSupport myDragZone = new HtmlDragSupport();
...
```

6.39.5. Details of Usage

The dragSupport tag inside a component completely specifies the events and JavaScript required to use the component and it's children for dragging as part of a drag-and-drop operation. In order to work, though, dragSupport must be placed inside a wrapper component that outputs child components and that has the right events defined on it. Thus, this example won't work, because the h:column tag doesn't provide the necessary properties for redefining events on the client:

Example:

```
...
<h:column>
```

```

        <rich:dragSupport dragIndicator=":form:iii" dragType="text">
            <a4j:actionParam value="#{caps.name}" name="name"/>
        </rich:dragSupport>
        <h:outputText value="#{caps.name}"/>
    </h:column>
    ...

```

However, using `a4j:outputPanel` as a wrapper inside `h:column`, the following code could be used successfully:

Example:

```

    ...
    <h:column>
        <a4j:outputPanel>
            <rich:dragSupport dragIndicator=":form:iii" dragType="text">
                <a4j:actionParam value="#{caps.name}" name="name"/>
            </rich:dragSupport>
            <h:outputText value="#{caps.name}"/>
        </a4j:outputPanel>
    </h:column>
    ...

```

This code makes all rows of this column draggable.

One of the main attributes for `dragSupport` is *"dragType"*, which associates a name with the drag zone. Only drop zones with this name as an acceptable type can be used in drag-and-drop operations. Here is an example:

Example:

```

    ...
    <h:panelGrid id="drag1">
        <rich:dragSupport dragType="singleItems" .../>
        <!--Some content to be dragged-->
    </h:panelGrid>
    ...
    <h:panelGrid id="drag2">
        <rich:dragSupport dragType="groups" .../>
        <!--Some content to be dragged-->
    </h:panelGrid>
    ...
    <h:panelGrid id="drop1">
        <rich:dropSupport acceptedTypes="singleItems" .../>
        <!--Drop zone content-->
    </h:panelGrid>
    ...

```

In this example, the `drop1` panel grid is a drop zone that invokes drag-and-drop for drops of items from the first `drag1` panel grid, but not the second `drag2` panel grid. In the section about `dropSupport`, you will find an example that shows more detailed information about moving data between tables with drag and drop.

The `dragSupport` component also has a *"value"* attribute for passing data into the processing after a drop event.

One more important attribute for `<rich:dragSupport>` is the *"dragIndicator"* attribute that point to the component id of the `<rich:dragIndicator>` component to be used for dragged items from this drag zone. If it isn't defined, a default indicator for drag operations is used.

Finally, the component has the following extra attributes for event processing on the client:

- `ondragenter`
- `ondragexit`

You can use your own custom JavaScript functions to handle these events.

6.39.6. Look-and-Feel Customization

`<rich:dragSupport>` has no skin parameters and custom style classes, as the component isn't visual.

6.39.7. Relevant Resources Links

Here [[http://livedemo.exadel.com/richfaces-demo/richfaces/dragSupport.jsf?c=dragSource](http://livedemo.exadel.com/richfaces-demo/richfaces/dragSupport.jsf?c=dragSupport)] you can see the example of `<rich:dragSupport>` usage and sources for the given example.

6.40. `< rich:dropSupport >`

6.40.1. Description

This component transforms a parent component into a target zone for drag-and-drop operations. When a draggable element is moved and dropped onto the area of the parent component, Ajax request processing for this event is started.

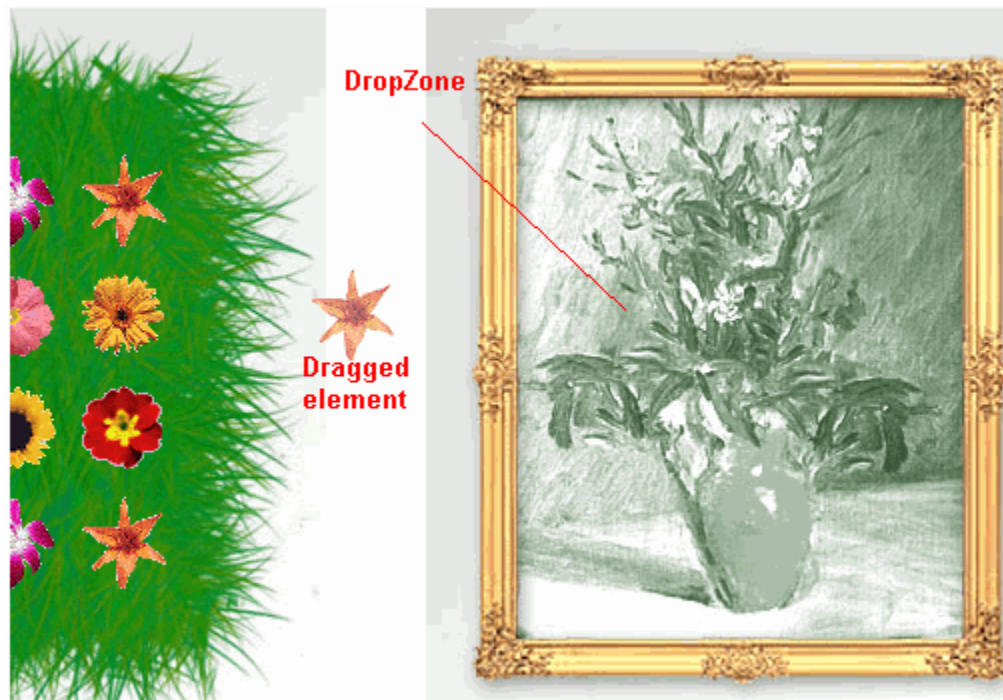


Figure 6.44. `<rich:dropSupport>` component

6.40.2. Key Features

- Encodes all necessary JavaScript to perform drop actions
- Can be used within any component type that provides the required properties for drop operations
- Built-in Ajax processing
- Supports drag-and-drop between different forms

Table 6.131. rich : dropSupport attributes

| Attribute Name | Description |
|----------------|---|
| acceptedTypes | List of drag types to be processed by the current drop zone |
| action | MethodBinding pointing at the application action to be invoked, if this UIComponent is activated by the user, during the Apply Request Values or Invoke Application phase of the request processing lifecycle, depending on the value of the immediate property |
| actionListener | MethodBinding pointing at method accepting an ActionEvent with return type void |
| ajaxSingle | if "true", submits ONLY one field/link, instead of all form controls |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| bypassUpdates | If "true", after process validations phase it skips updates of model beans on a force render response. It can be used for validating components input |
| data | Serialized (on default with JSON) data passed on the client by a developer on AJAX request. It's accessible via "data.foo" syntax |
| disableDefault | Disable default action for target event (append "return false;" to JavaScript) |
| dropListener | MethodBinding representing an action listener method that will be notified after drop operation. |
| dropValue | Data to be processed after a drop event |
| eventsQueue | Name of requests queue to avoid send next request before complete other from same event. Can be used to reduce number of requests of frequently events (key press, mouse move etc.) |

| Attribute Name | Description |
|--------------------|--|
| focus | id of element to set focus after request completed on client side |
| id | Every component may have a unique id that is automatically created if omitted |
| ignoreDupResponses | Attribute allows to ignore an Ajax Response produced by a request if the newest 'similar' request is in a queue already. ignoreDupResponses="true" does not cancel the request while it is processed on the server, but just allows to avoid unnecessary updates on the client side if the response isn't actual now |
| immediate | True means, that the default ActionListener should be executed immediately (i.e. during Apply Request Values phase of the request processing lifecycle), rather than waiting until the Invoke Application phase |
| limitToList | If "true", updates on client side ONLY elements from this 'reRender' property. If "false" (default) updates all rendered by ajax region components |
| onbeforedomupdate | JavaScript code for call before DOM has been updated on client side |
| oncomplete | JavaScript code for call after request completed on client side |
| ondragenter | A JavaScript event handler called on enter draggable object to zone |
| ondragexit | A JavaScript event handler called after a drag object leaves zone |
| ondrop | A JavaScript event handler called after a drag object is dropped to zone |
| ondropend | A JavaScript handler for event fired on a drop even the drop for a given type is not available |
| onsubmit | JavaScript code for call before submission of ajax event |
| rendered | If "false", this component is not rendered |
| requestDelay | Attribute defines the time (in ms.) that the request will be wait in the queue before it is ready to send. When the delay time is over, the request will be sent to the server or removed if the newest 'similar' request is in a queue already |

| Attribute Name | Description |
|----------------|---|
| reRender | Id[s] (in format of call <code>UIComponent.findComponent()</code> of components, rendered in case of <code>AjaxRequest</code> caused by this component. Can be single id, comma-separated list of Id's, or EL Expression with array or Collection |
| status | ID (in format of call <code>UIComponent.findComponent()</code> of Request status component |
| timeout | Response waiting time on a particular request. If a response is not received during this time, the request is aborted |
| typeMapping | Map between a draggable type and an indicator name on zone. it's defined with the pair (drag type:indicator name)) |
| value | The current value for this component |

Table 6.132. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | <code>org.richfaces.DropSupport</code> |
| component-class | <code>org.richfaces.component.html.HtmlDropSupport</code> |
| component-family | <code>org.richfaces.DropSupport</code> |
| renderer-type | <code>org.richfaces.DropSupportRenderer</code> |
| tag-class | <code>org.richfaces.taglib.DropSupportTag</code> |

6.40.4. Creating the Component with a Page Tag

This simple example shows how to make a panel component a potential drop target for drag-and-drop operations using "text" elements as the dragged items.

Example:

```
...
<rich:panel>
  <rich:dropSupport acceptedTypes="text"/>
</rich:panel>
...
```

6.40.5. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlDropSupport;
```

```
...
HtmlDropSupport myDragZone = new HtmlDropSupport();
...
```

6.40.6. Details of Usage

As shown in the example, the key attribute for `<rich:dropSupport>` is *"acceptedTypes"*. This attribute defines the types of draggable items that can be dropped onto the designated drop zone.

The second most important attribute for `<rich:dropSupport>` is *"typeMapping"*. This attribute maps a specific type among the acceptable types for draggable items to a specific `<rich:dndParam>` child element under `<rich:dropSupport>`.

Example:

```
...
<rich:dropSupport acceptedTypes="[iconsDragged, textDragged]"
typeMapping="{iconsDragged: DropIcon}">
    <rich:dndParam name="DropIcon">
        <h:graphicImage value="/images/drop-icon.png"/>
    </rich:dndParam>
</rich:dropSupport>
...
```

In this example, dropping a draggable item of an *"iconsDragged"* type will trigger the use a parameter named *"DropIcon"* in the event processing after a drop event. (Also, an Ajax request is sent, and the action and dropListener defined for the component are called.)

Here is an example of moving records between tables. The example describes all the pieces for drag-and-drop. (To get extra information on these components, read the sections for these components.)

As draggable items, this table contains a list of such items designated as being of type "text":

Example:

```
...
<rich:dataTable value="#{capitalsBean.capitals}" var="caps">
    <f:facet name="caption">Capitals List</f:facet>
    <h:column>
        <a4j:outputPanel>
            <rich:dragSupport dragIndicator=":form:ind" dragType="text">
                <a4j:actionParam value="#{caps.name}" name="name"/>
            </rich:dragSupport>
            <h:outputText value="#{caps.name}"/>
        </a4j:outputPanel>
    </h:column>
</rich:dataTable>
...
```

As a drop zone, this panel will accept draggable items of type "text" and then rerender an element with the ID of "box":

Example:

```
...
```

```

<rich:panel style="width:100px;height:100px;">
    <f:facet name="header">Drop Zone</f:facet>
    <rich:dropSupport acceptedTypes="text" reRender="box"
                    dropListener="#{capitalsBean.addCapital2}" />
</rich:panel>
...

```

As a part of the page that can be updated in a partial page update, this table has an ID of "box":

Example:

```

...
<rich:dataTable value="#{capitalsBean.capitals2}" var="cap2" id="box">
    <f:facet name="caption">Capitals chosen</f:facet>
    <h:column>
        <h:outputText value="#{cap2.name}" />
    </h:column>
</rich:dataTable>
...

```

And finally, as a listener, this listener will implement the dropped element:

Example:

```

...
public void addCapital2(DropEvent event) {
    FacesContext context = FacesContext.getCurrentInstance();
    Capital cap = new Capital();

    cap.setName(context.getExternalContext().getRequestParameterMap().get("name").toString());
    capitals2.add(cap);
}
...

```

Here is the result after a few drops of items from the first table:

| Capitals List | Drop Zone | Capitals chosen |
|---------------|-----------|-----------------|
| Montgomery | | Little Rock |
| Juneau | | Denver |
| Phoenix | | |
| Little Rock | | |
| Sacramento | | |
| Denver | | |
| Hartford | | |
| Dover | | |
| Tallahassee | | |
| Atlanta | | |
| Honolulu | | |

Figure 6.45. Results of drop actions

In this example, items are dragged element-by-element from the rendered list in the first table and dropped on a panel in the middle. After each drop, a drop event is generated and a common Ajax request is performed that renders results in the third table.

As with every Ajax action component, `<rich:dropSupport>` has all the common attributes (*"timeout"*, *"limitToList"*, *"reRender"*, etc.) for Ajax request customization.

Finally, the component has the following extra attributes for event processing on the client:

- `ondragenter`
- `ondragexit`
- `ondrop`
- `ondropend`

Developers can use their own custom JavaScript functions to handle these events.

6.40.7. Look-and-Feel Customization

`<rich:dropSupport>` has no skin parameters and custom style classes, as the component isn't visual.

6.40.8. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/dragSupport.jsf?c=dropSupport>] you can see the example of `<rich:dropSupport>` usage and sources for the given example.

6.41. < rich:dragListener >

6.41.1. Description

The `<rich:dragListener>` represents an action listener method that will be notified after a drag operation.

6.41.2. Key Features

- Allows to define some drag listeners for the components with "Drag and Drop" support

Table 6.133. rich : dragListener attributes

| Attribute Name | Description |
|----------------|---|
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| type | Attribute defines the fully qualified Java class name for listener |

Table 6.134. Component identification parameters

| Name | Value |
|----------------|----------------------------------|
| listener-class | org.richfaces.event.DragListener |

| Name | Value |
|-------------|--------------------------------------|
| event-class | org.richfaces.event.DragEvent |
| tag-class | org.richfaces.taglib.DragListenerTag |

6.41.3. Creating the Component with a Page Tag

Simple Component definition on a page:

Example:

```
...
<rich:dragListener type="demo.Bean"/>
...
```

6.41.4. Creating the Component Dynamically Using Java

Example:

```
package demo;

public class ImplBean implements org.richfaces.event.DragListener{
    ...
}
```

```
import demo.ImplBean;
...
ImplBean myDragListener = new ImplBean();
...
```

6.41.5. Details of Usage

The `<rich:dragListener>` is used as a nested tag with components like `<rich:dragSupport>`, `<rich:tree>` and `<rich:treeNode>`.

Attribute *"type"* defines the fully qualified Java class name for a listener. This class should implement `org.richfaces.event.DragListener` interface.

The typical variant of using:

```
...
<h:panelGrid id="dragPanel">
  <rich:dragSupport dragType="item">
    <rich:dragListener type="demo.ListenerBean"/>
  </rich:dragSupport>
  <!--Some content to be dragged-->
</h:panelGrid>
...
```

Java bean source:

```

package demo;

import org.richfaces.event.DragEvent;

public class ListenerBean implements org.richfaces.event.DragListener{
    ...
    public void processDrag(DragEvent arg0){
        //Custom Developer Code
    }
    ...
}

```

6.41.6. Look-and-Feel Customization

`<rich:dragListener>` has no skin parameters and custom style classes, as the component isn't visual.

6.42. `< rich:dropListener >`

6.42.1. Description

The `<rich:dropListener>` represents an action listener method that will be notified after a drop operation.

6.42.2. Key Features

- Allows to define some drop listeners for the components with "Drag and Drop" support

Table 6.135. rich : dropListener attributes

| Attribute Name | Description |
|----------------|---|
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| type | Attribute defines the fully qualified Java class name for listener |

Table 6.136. Component identification parameters

| Name | Value |
|----------------|--------------------------------------|
| listener-class | org.richfaces.event.DropListener |
| event-class | org.richfaces.event.DropEvent |
| tag-class | org.richfaces.taglib.DropListenerTag |

6.42.3. Creating the Component with a Page Tag

Simple Component definition on a page:

Example:

```

...
<rich:dropListener type="demo.Bean"/>

```


...

6.42.4. Creating the Component Dynamically Using Java

Example:

```
package demo;

public class ImplBean implements org.richfaces.event.DropListener{
    ...
}
```

```
import demo.ImplBean;
...
ImplBean myListener = new ImplBean();
...
```

6.42.5. Details of Usage

The `<rich:dropListener>` is used as a nested tag with components like `<rich:dropSupport>`, `<rich:tree>` and `<rich:treeNode>`.

Attribute *"type"* defines the fully qualified Java class name for the listener. This class should implement `org.richfaces.event.DropListener` interface.

The typical variant of using:

```
...
<rich:panel style="width:100px;height:100px;">
  <f:facet name="header">Drop Zone</f:facet>
  <rich:dropSupport acceptedTypes="text">
    <rich:dropListener type="demo.ListenerBean"/>
  </rich:dropSupport>
</rich:panel>
...
```

Java bean source:

```
package demo;

import org.richfaces.event.DropEvent;

public class ListenerBean implements org.richfaces.event.DropListener{
    ...
    public void processDrop(DropEvent arg0){
        //Custom Developer Code
    }
    ...
}
```

6.42.6. Look-and-Feel Customization

`<rich:dropListener>` has no skin parameters and custom style classes, as the component isn't visual.

6.43. < rich:dropDownMenu >

6.43.1. Description

The <rich:dropDownMenu> component is used for creating multilevel drop-down menus.

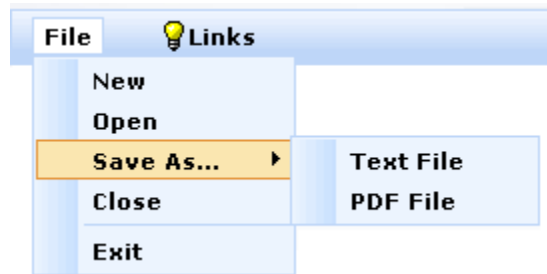


Figure 6.46. <rich:dropDownMenu> component

6.43.2. Key Features

- Highly customizable look-and-feel
- Pop-up appearance event customization
- Different submission modes
- Ability to define a complex representation for elements
- Support for disabling
- Smart user-defined positioning

Table 6.137. rich : dropDownMenu attributes

| Attribute Name | Description |
|-------------------|--|
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| direction | Defines direction of the popup list to appear (top-right, top-left bottom-right, bottom-left, auto(default)) |
| disabled | Attribute 'disabled' provides possibility to make the whole menu disabled if its value equals to "true". |
| disabledItemClass | Space-separated list of CSS style class(es) that are be applied to disabled item of this component |
| disabledItemStyle | CSS style(s) is/are to be applied to disabled item when this component is rendered. |
| event | Defines the event on the representation element that triggers the menu's appearance. |
| hideDelay | Delay between losing focus and menu closing. |

| Attribute Name | Description |
|------------------|--|
| horizontalOffset | Sets the horizontal offset between popup list and label element conjunction point |
| id | Every component may have a unique id that is automatically created if omitted |
| itemClass | Space-separated list of CSS style class(es) that are be applied to item of this component |
| itemStyle | CSS style(s) is/are to be applied to item when this component is rendered. |
| jointPoint | Set the corner of the label for the popup to be connected with. (auto(default), tr, tl, bl, br, where tr top-right) |
| oncollapse | Event must occurs on menu closure |
| onexpand | Event must occurs on menu opening |
| ongroupactivate | HTML: script expression; some group was activated. |
| onitemselect | HTML: script expression; some item was selected. |
| onmousemove | HTML: script expression; a pointer was moved within. |
| onmouseout | HTML: script expression; a pointer was moved away. |
| onmouseover | HTML: script expression; a pointer was moved onto. |
| popupWidth | Set minimal width for the all of the lists that will appear. |
| rendered | If "false", this component is not rendered |
| selectItemClass | Space-separated list of CSS style class(es) that are be applied to selected item of this component. |
| selectItemStyle | CSS style(s) is/are to be applied to selected item when this component is rendered. |
| showDelay | Delay between event and menu showing. |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| submitMode | Set the submission mode for all menu items of the menu except ones where this attribute redefined. (ajax,server(Default),none) |
| value | Defines representation text for Label used for menu calls. |

| Attribute Name | Description |
|----------------|---|
| verticalOffset | Sets the vertical offset between popup list and label element conjunction point |

Table 6.138. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | org.richfaces.DropDownMenu |
| component-class | org.richfaces.component.html.HtmlDropDownMenu |
| component-family | org.richfaces.DropDownMenu |
| renderer-type | org.richfaces.DropDownMenuRenderer |
| tag-class | org.richfaces.taglib.DropDownMenuTag |

6.43.3. Creating the Component with a Page Tag

Here is a simple example as it could be used on a page:

Example:

```
...
<rich:dropDownMenu value="Item1">
  <!--Nested menu components-->
</rich:dropDownMenu>
...
```

6.43.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlDropDownMenu;
...
HtmlDropDownMenu myDropDownMenu = new HtmlDropDownMenu();
...
```

6.43.5. Details of Usage

All attributes except *"value"* are optional. The *"value"* attribute defines text to be represented. If you can use the *"label"* facet, you can even not use the *"value"* attribute.

Here is an example:

Example:

```
...
<f:facet name="label">
```

```

    <h:graphicImage value="/images/img1.png"/>
  </f:facet>
  ...

```

Use the *"event"* attribute to define an event for the represented element that triggers a menu appearance. An example of a menu appearance on a click can be seen below.

Example:

```

...
<rich:dropDownMenu event="onclick" value="Item1">
  <!--Nested menu components-->
</rich:dropDownMenu>
...

```

The **<rich:dropDownMenu>** *"submitMode"* attribute can be set to three possible parameters:

- Server (default)

Regular form submission request is used.

- Ajax

Ajax submission is used for switching.

- None

The *"action"* and *"actionListener"* item's attributes are ignored. Menu items don't fire any submits themselves. The behavior is fully defined by the components nested into items.

Note:

As the **<rich:dropDownMenu>** component doesn't provide its own form, use it between **<h:form>** and **</h:form>** tags.

The *"direction"* and *"jointPoint"* attributes are used for defining aspects of menu appearance.

Possible values for the *"direction"* attribute are:

- top-left - a menu drops to the top and left
- top-right - a menu drops to the top and right
- bottom-left - a menu drops to the bottom and left
- bottom-right - a menu drops to the bottom and right
- auto - smart positioning activation

Possible values for the *"jointPoint"* attribute are:

- tr - a menu is attached to the top-right point of the button element
- tl - a menu is attached to the top-left point of the button element
- br - a menu is attached to the bottom-right point of the button element
- bl - a menu is attached to the bottom-left point of the button element
- auto - smart positioning activation

By default, the *"direction"* and *"jointPoint"* attributes are set to *"auto"*.

Here is an example:

Example:

```
...
<rich:dropDownMenu value="File" direction="bottom-right" jointPoint="tr">
  <rich:menuItem submitMode="ajax" value="New" action="#{ddmenu.doNew}"/>
  <rich:menuItem submitMode="ajax" value="Open" action="#{ddmenu.doOpen}"/>
  <rich:menuGroup value="Save As...">
    <rich:menuItem submitMode="ajax" value="Text File"
action="#{ddmenu.doSaveText}"/>
    <rich:menuItem submitMode="ajax" value="PDF File"
action="#{ddmenu.doSavePDF}"/>
  </rich:menuGroup>
  <rich:menuItem submitMode="ajax" value="Close" action="#{ddmenu.doClose}"/>
  <rich:menuSeparator id="menuSeparator11"/>
  <rich:menuItem submitMode="ajax" value="Exit" action="#{ddmenu.doExit}"/>
</rich:dropDownMenu>
...
```

This is the result:

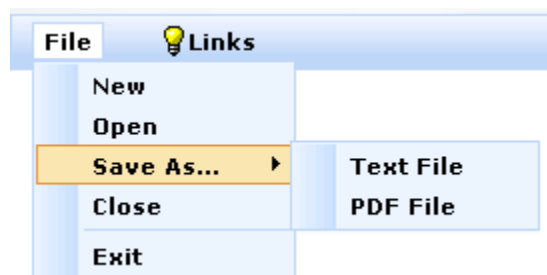


Figure 6.47. Using the *"direction"* and *"jointPoint"* attributes

You can correct an offset of the pop-up list relative to the label using the following attributes: *"horizontalOffset"* and *"verticalOffset"*.

Here is an example:

Example:

```
...
```

```

<rich:dropDownMenu value="File" direction="bottom-right" jointPoint="tr"
horizontalOffset="-15" verticalOffset="0">
    <rich:menuItem submitMode="ajax" value="New" action="#{ddmenu.doNew}"/>
    <rich:menuItem submitMode="ajax" value="Open" action="#{ddmenu.doOpen}"/>
    <rich:menuGroup value="Save As...">
        <rich:menuItem submitMode="ajax" value="Text File"
action="#{ddmenu.doSaveText}"/>
        <rich:menuItem submitMode="ajax" value="PDF File"
action="#{ddmenu.doSavePDF}"/>
    </rich:menuGroup>
    <rich:menuItem submitMode="ajax" value="Close" action="#{ddmenu.doClose}"/>
    <rich:menuSeparator id="menuSeparator11"/>
    <rich:menuItem submitMode="ajax" value="Exit" action="#{ddmenu.doExit}"/>
</rich:dropDownMenu>
...

```

This is the result:

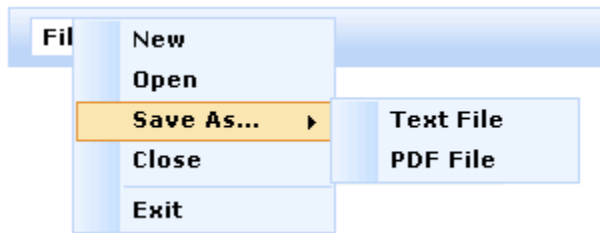


Figure 6.48. Using the "horizontalOffset" and "verticalOffset" attributes

The `<rich:dropDownMenu>` component allows to use `"labelDisabled"`. It's necessary to define `"disabled"` attribute as `"true"` for `dropDownMenu`. An example is placed below.

Example:

```

...
<rich:dropDownMenu disabled="true">
    <f:facet name="labelDisabled">
        <h:graphicImage value="/images/icol.png"/>
    </f:facet>
    <!--Nested menu components-->
</rich:dropDownMenu>
...

```

6.43.6. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

There are two ways to redefine the appearance of all `<rich:dropDownMenu>` components at once:

- Redefine the corresponding skin parameters
- Add to your style sheets *style classes* used by a `<rich:dropDownMenu>` component

6.43.7. Skin Parameters Redefinition

Table 6.139. Skin parameters redefinition for a label <div> element

| Skin parameters | CSS properties |
|-------------------|----------------|
| generalFamilyFont | font-family |
| generalSizeFont | font-size |

Table 6.140. Skin parameters redefinition for a selected label

| Skin parameters | CSS properties |
|------------------------|-----------------------|
| panelBorderColor | border-color |
| controlBackgroundColor | background-color |
| generalTextColor | background-colorcolor |

Table 6.141. Skin parameters redefinition for a border

| Skin parameters | CSS properties |
|---------------------------|------------------|
| panelBorderColor | border-color |
| additionalBackgroundColor | background-color |

Table 6.142. Skin parameters redefinition for a background

| Skin parameters | CSS properties |
|---------------------------|--------------------|
| additionalBackgroundColor | border-top-color |
| additionalBackgroundColor | border-left-color |
| additionalBackgroundColor | border-right-color |

6.43.8. Definition of Custom Style Classes

On the screenshot there are classes names that define styles for component elements.

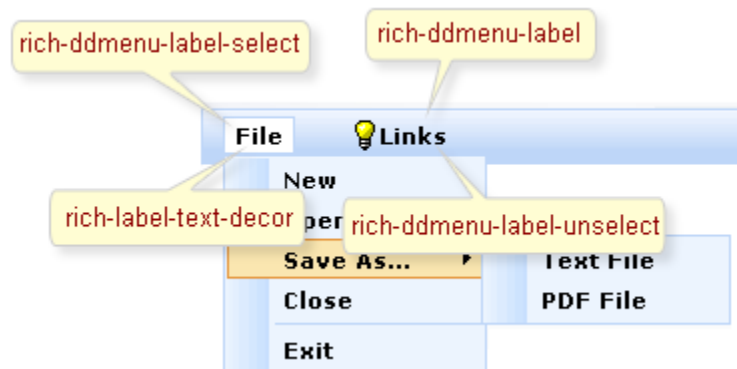
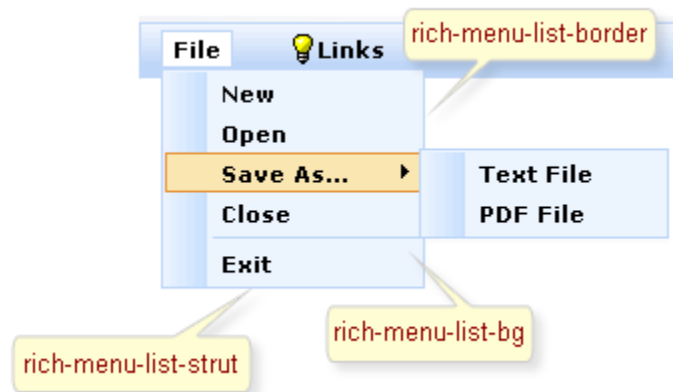


Figure 6.49. Classes names

Table 6.143. Classes names that define a label

| Class name | Description |
|----------------------------|--|
| rich-label-text-decor | Defines text style for a representation element |
| rich-ddmenu-label | Defines styles for a wrapper <div> element of a representation element |
| rich-ddmenu-label-select | Defines styles for a wrapper <div> element of a selected representation element |
| rich-ddmenu-label-unselect | Defines styles for a wrapper <div> element of an unselected representation element |
| rich-ddmenu-label-disabled | Defines styles for a wrapper <div> element of a disabled representation element |

On the screenshot there are classes names that define styles for component elements.

**Figure 6.50. Classes names****Table 6.144. Classes names that define a popup element**

| Class name | Description |
|-----------------------|--|
| rich-menu-list-border | Defines styles for borders |
| rich-menu-list-bg | Defines styles for a general background list |
| rich-menu-list-strut | Defines styles for a wrapper <div> element for a strut of a popup list |

In order to redefine styles for all **<rich:dropDownMenu>** components on a page using CSS, it's enough to create classes with the same names and define necessary properties in them.

To change styles of particular **<rich:dropDownMenu>** components, define your own style classes in the corresponding **<rich:dropDownMenu>** attributes.

CSS code piece used on a page:

Example:

```
...
.rich-ddmenu-label-select{
    font-size: 15px;
    color: #FFF0F5;
}
.myClass{
    background-color: #2F4F4F;
}
...
```

The component is defined in the following way:

Example:

```
<rich:dropDownMenu ... inputClass="myClass" .../>
```

6.43.9. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/dropDownMenu.jsf?c=dropDownMenu>] you can see the example of **<rich:dropDownMenu>** usage and sources for the given example.

6.44. < rich:menuGroup >

6.44.1. Description

The **<rich:menuGroup>** component is used to define an expandable group of items inside a pop-up list or another group.

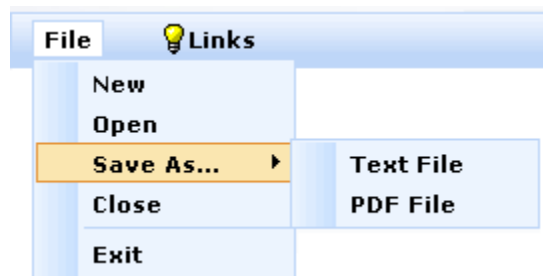


Figure 6.51. **<rich:menuGroup>** component

6.44.2. Key Features

- Highly customizable look-and-feel
- Grouping of any menu's items
- Pop-up appearance event customization

- Support for disabling
- Smart user-defined positioning

Table 6.145. rich : menuGroup attributes

| Attribute Name | Description |
|--------------------|---|
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| converter | Id of Converter to be used or reference to a Converter |
| direction | Defines direction of the popup sublist to appear (right, left, auto(Default), left-down, left-up, right-down, right-up) |
| disabled | If "true" sets state of the item to disabled state. "false" is default |
| event | Defines the event on the representation element that triggers the menu's appearance |
| icon | Path to the icon to be displayed for the enabled item state |
| iconClass | Class to be applied to icon element |
| iconDisabled | Path to the icon to be displayed for the disabled item state |
| iconFolder | Path to the folder icon to be displayed for the enabled item state |
| iconFolderDisabled | Path to the folder icon to be displayed for the disabled item state |
| iconStyle | CSS style rules to be applied to icon element |
| id | Every component may have a unique id that is automatically created if omitted |
| onclose | HTML: script expression; group was closed |
| onmousemove | HTML: a script expression; a pointer is moved within |
| onmouseout | HTML: a script expression; a pointer is moved away |
| onmouseover | HTML: a script expression; a pointer is moved onto |
| onopen | HTML: script expression; group was opened |
| rendered | If "false", this component is not rendered |
| selectClass | Class to be applied to selected items |

| Attribute Name | Description |
|----------------|---|
| selectStyle | CSS style rules to be applied to selected items |
| showDelay | Delay between event and menu showing |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| value | Defines representation text for menuItem |

Table 6.146. Component identification parameters

| Name | Value |
|------------------|--|
| component-type | org.richfaces.MenuGroup |
| component-class | org.richfaces.component.html.HtmlMenuGroup |
| component-family | org.richfaces.DropDownMenu |
| renderer-type | org.richfaces.MenuGroupRenderer |
| tag-class | org.richfaces.taglib.MenuGroupTag |

6.44.3. Creating the Component with a Page Tag

Here is a simple example as it could be used on a page:

Example:

```
...
<rich:dropDownMenu value="Active">
    ...
    <rich:menuGroup value="Active">
        <!--Nested menu components-->
    </rich:menuGroup>
    ...
</rich:dropDownMenu >
...
```

6.44.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlMenuGroup;
...
HtmlMenuGroup myMenuGroup = new HtmlMenuGroup();
...
```

6.44.5. Details of Usage

The *"value"* attribute defines the text representation of a group element in the page.

The *"icon"* attribute defines an icon for the component. The *"iconDisabled"* attribute defines an icon for when the group is disabled. Also you can use the *"icon"* and *"iconDisabled"* facets. If the facets are defined, the corresponding *"icon"* and *"iconDisabled"* attributes are ignored and the facets' contents are used as icons. This could be used for an item check box implementation.

Here is an example:

```
...
    <f:facet name="icon">
        <h:selectBooleanCheckbox value="#{bean.property}" />
    </f:facet>
...
```

The *"iconFolder"* and *"iconFolderDisabled"* attributes are defined for using icons as folder icons. The *"iconFolder"* and *"iconFolderDisabled"* facets use their contents as folder icon representations in place of the attribute values.

The *"direction"* attribute is used to define which way to display the menu as shown in the example below:

Possible values are:

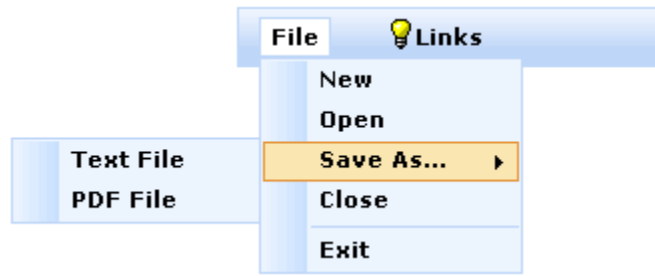
- left - down - a submenu is attached to the left side of the menu and is dropping down
- left - up - a submenu is attached to the left side of the menu and is dropping up
- right - down - a submenu is attached to the right side of the menu and is dropping down
- right - up - a submenu is attached to the right side of the menu and is dropping up
- auto - smart positioning activation

By default, the *"direction"* attribute is set to *"auto"*.

Here is an example:

```
...
    <rich:menuGroup value="Save As..." direction="left-down">
        <rich:menuItem submitMode="ajax" value="Text File"
action="#{ddmenu.doSaveText}" />
        <rich:menuItem submitMode="ajax" value="PDF File"
action="#{ddmenu.doSavePDF}" />
    </rich:menuGroup>
...
```

This would be the result:

Figure 6.52. Using the *"direction"* attribute**Note:**

The `<rich:menuGroup>` component was designed to be used only for pop-up menu list creation.

6.44.6. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

There are two ways to redefine the appearance of all `<rich:menuGroup>` components at once:

- Redefine the corresponding skin parameters
- Add to your style sheets *style classes* used by a `<rich:menuGroup>` component

6.44.7. Skin Parameters Redefinition

Table 6.147. Skin parameters redefinition for a group

| Skin parameters | CSS properties |
|-------------------|----------------|
| generalFamilyFont | font-family |
| generalSizeFont | font-size |

Table 6.148. Skin parameters redefinition for a disabled group

| Skin parameters | CSS properties |
|----------------------|----------------|
| tabDisabledTextColor | color |

Table 6.149. Skin parameters redefinition for a label

| Skin parameters | CSS properties |
|------------------|----------------|
| generalTextColor | color |

6.44.8. Definition of Custom Style Classes

On the screenshot there are classes names that define styles for component elements.



Figure 6.53. Classes names

Table 6.150. Classes names that define an appearance of group elements

| Class name | Description |
|-----------------------|--|
| rich-menu-group | Defines styles for a wrapper <div> element for a group |
| rich-menu-item-label | Defines styles for a label of an item |
| rich-menu-item-icon | Defines styles for the left icon of an item |
| rich-menu-item-folder | Defines styles for the right icon of an item |

Table 6.151. Classes names that define different states

| Class name | Description |
|--------------------------------|---|
| rich-menu-item-label-disabled | Defines styles for a label of a disabled item |
| rich-menu-item-icon-disabled | Defines styles for the left icon of a disabled item |
| rich-menu-item-folder-disabled | Defines styles for the right icon of a disabled item |
| rich-menu-group-hover | Defines styles for a wrapper <div> element of a hover group |
| rich-menu-item-icon-enabled | Defines styles for the left icon of an enabled item |
| rich-menu-item-icon-selected | Defines styles for the left icon of a selected item |

In order to redefine styles for all `<rich:menuGroup>` components on a page using CSS, it's enough to create classes with the same names and define necessary properties in them.

To change styles of particular `<rich:menuGroup>` components, define your own style classes in the corresponding `<rich:menuGroup>` attributes.

6.44.9. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/dropDownMenu.jsf?c=menuGroup>] you can see the example of `<rich:menuGroup>` usage and sources for the given example.

6.45. < rich:menuItem >

6.45.1. Description

The <rich:menuItem> component is used for the definition of a single item inside a pop-up list.

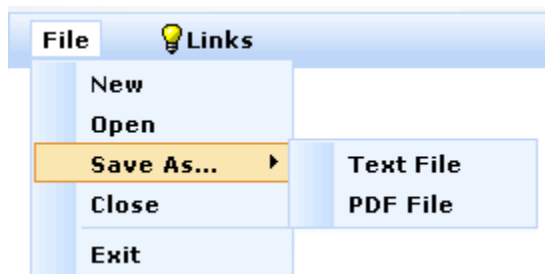


Figure 6.54. <rich:menuItem> component

6.45.2. Key Features

- Highly customizable look-and-feel
- Different submission modes
- Support for disabling
- Custom content support

Table 6.152. rich : menuItem attributes

| Attribute Name | Description |
|------------------|---|
| action | MethodBinding pointing at the application action to be invoked, if this UIComponent is activated by the user, during the Apply Request Values or Invoke Application phase of the request processing lifecycle, depending on the value of the immediate property |
| actionExpression | The action method binding expression |
| actionListener | MethodBinding pointing at method accepting an ActionEvent with return type void |
| ajaxSingle | if "true", submits ONLY one field/link, instead of all form controls |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| bypassUpdates | If "true", after process validations phase it skips updates of model beans on a force render response. It can be used for validating components input |
| data | |

| Attribute Name | Description |
|--------------------|--|
| | Serialized (on default with JSON) data passed on the client by a developer on AJAX request. It's accessible via "data.foo" syntax |
| disabled | If "true" sets state of the item to disabled state. "false" is default |
| eventsQueue | Name of requests queue to avoid send next request before complete other from same event. Can be used to reduce number of requests of frequently events (key press, mouse move etc.) |
| focus | id of element to set focus after request completed on client side |
| icon | Path to the icon to be displayed for the enabled item state |
| iconClass | Class to be applied to icon element |
| iconDisabled | Path to the icon to be displayed for the disabled item state. |
| iconStyle | CSS style rules to be applied to icon element |
| id | Every component may have a unique id that is automatically created if omitted |
| ignoreDupResponses | Attribute allows to ignore an Ajax Response produced by a request if the newest 'similar' request is in a queue already. ignoreDupResponses="true" does not cancel the request while it is processed on the server, but just allows to avoid unnecessary updates on the client side if the response isn't actual now |
| immediate | True means, that the default ActionListener should be executed immediately (i.e. during Apply Request Values phase of the request processing lifecycle), rather than waiting until the Invoke Application phase |
| limitToList | If "true", updates on client side ONLY elements from this 'reRender' property. If "false" (default) updates all rendered by ajax region components |
| onbeforedomupdate | JavaScript code for call before DOM has been updated on client side |
| onclick | HTML: a script expression; a pointer button is clicked |

| Attribute Name | Description |
|----------------|--|
| oncomplete | JavaScript code for call after request completed on client side |
| onmousedown | HTML: script expression; a pointer button is pressed down |
| onmousemove | HTML: a script expression; a pointer is moved within |
| onmouseout | HTML: a script expression; a pointer is moved away |
| onmouseover | HTML: a script expression; a pointer is moved onto |
| onmouseup | HTML: script expression; a pointer button is released |
| onselect | HTML: script expression; The onselect event occurs when a user selects some menu item |
| rendered | If "false", this component is not rendered |
| requestDelay | Attribute defines the time (in ms.) that the request will be wait in the queue before it is ready to send. When the delay time is over, the request will be sent to the server or removed if the newest 'similar' request is in a queue already |
| reRender | Id['s] (in format of call <code>UIComponent.findComponent()</code>) of components, rendered in case of <code>AjaxRequest</code> caused by this component. Can be single id, comma-separated list of Id's, or EL Expression with array or Collection |
| selectClass | Class to be applied to selected items |
| selectStyle | CSS style rules to be applied to selected items |
| status | ID (in format of call <code>UIComponent.findComponent()</code>) of Request status component |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| submitMode | Set the submission mode (ajax, server(Default), none) |
| target | Name of a frame where the resource retrieved via this hyperlink is to be displayed |
| timeout | Response waiting time on a particular request. If a response is not received during this time, the request is aborted |

| Attribute Name | Description |
|----------------|--------------------------------------|
| value | The current value for this component |

Table 6.153. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | org.richfaces.MenuItem |
| component-class | org.richfaces.component.html.HtmlMenuItem |
| component-family | org.richfaces.DropDownMenu |
| renderer-type | org.richfaces.MenuItemRenderer |
| tag-class | org.richfaces.taglib.MenuItemTag |

6.45.3. Creating the Component with a Page Tag

Here is a simple example as it could be used on a page:

Example:

```
...
<rich:dropDownMenu>
    ...
    <rich:menuItem value="Active"/>
    ...
</rich:dropDownMenu>
...
```

6.45.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlMenuItem;
...
HtmlMenuItem myMenuItem = new HtmlMenuItem();
...
```

6.45.5. Details of Usage

The *"value"* attribute defines the text representation for an item element.

There are two icon-related attributes. The *"icon"* attribute defines an icon. The *"iconDisabled"* attribute defines an icon for a disabled item. Also you can use the *"icon"* and *"iconDisabled"* facets. If the facets are defined, the corresponding *"icon"* and *"iconDisabled"* attributes are ignored and the facets content is shown as an icon. It could be used for an item check box implementation.

Here is an example:

```
...
```

```

<f:facet name="icon">
    <h:selectBooleanCheckbox value="#{bean.property}" />
</f:facet>
...

```

The **<rich:menuItem>** *submitMode* attribute can be set to three possible parameters:

- Server (default)

Regular form submission request is used.

- Ajax

Ajax submission is used for switching.

- None

The *action* and *actionListener* item's attributes are ignored. Menu items don't fire any submits themselves. The behavior is fully defined by the components nested into items.

For example, you can put any content into an item, but, in this case, you should set the *mode* attribute as *none*.

Here is an example:

```

...
<rich:dropDownMenu>
    ...
    <rich:menuItem submitMode="none">
        <h:outputLink value="www.jboss.org" />
    </rich:menuItem>
    ...
</rich:dropDownMenu>
...

```

You can use the *disabled* attribute to set the item state.

Here is an example:

```

...
<rich:dropDownMenu>
    <rich:menuItem value="Disable" disabled="true" />
</rich:dropDownMenu>
...

```

Note:

The **<rich:menuItem>** component was designed to be used only for pop-up menu list creation.

6.45.6. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

There are two ways to redefine the appearance of all `<rich:menuItem>` components at once:

- Redefine the corresponding skin parameters
- Add to your style sheets *style classes* used by a `<rich:menuItem>` component

6.45.7. Skin Parameters Redefinition

Table 6.154. Skin parameters redefinition for an item

| Skin parameters | CSS properties |
|-------------------|----------------|
| generalFamilyFont | font-family |
| generalSizeFont | font-size |

Table 6.155. Skin parameters redefinition for a hovered item

| Skin parameters | CSS properties |
|--------------------|------------------|
| tipBorderColor | border-color |
| tipBackgroundColor | background-color |

Table 6.156. Skin parameters redefinition for a disabled item

| Skin parameters | CSS properties |
|----------------------|----------------|
| tabDisabledTextColor | color |

Table 6.157. Skin parameters redefinition for a label

| Skin parameters | CSS properties |
|------------------|----------------|
| generalTextColor | color |

6.45.8. Definition of Custom Style Classes

On the screenshot there are classes names that define styles for component elements.



Figure 6.55. Class names

Table 6.158. Classes names that define an appearance of item elements

| Class name | Description |
|----------------------|--|
| rich-menu-item | Defines styles for a wrapper <div> element for an item |
| rich-menu-item-label | Defines styles for a label of an item |
| rich-menu-item-icon | Defines styles for the left icon of an item |

Table 6.159. Classes names that define different states

| Class name | Description |
|-------------------------------|---|
| rich-menu-item-disabled | Defines styles for a wrapper <div> element of an item |
| rich-menu-item-enabled | Defines styles for a wrapper <div> element of an enabled item |
| rich-menu-item-hover | Defines styles for a wrapper <div> element of a hover item |
| rich-menu-item-label-disabled | Defines styles for a label of a disabled item |
| rich-menu-item-icon-disabled | Defines styles for the left icon of a disabled item |
| rich-menu-item-label-enabled | Defines styles for a label of an enabled item |
| rich-menu-item-icon-enabled | Defines styles for the left icon of an enabled item |
| rich-menu-item-label-selected | Defines styles for a label of a selected item |
| rich-menu-item-icon-selected | Defines styles for the left icon of a selected item |

In order to redefine styles for all **<rich:menuItem>** components on a page using CSS, it's enough to create classes with the same names and define necessary properties in them.

To change styles of particular **<rich:menuItem>** components, define your own style classes in the corresponding **<rich:menuItem>** attributes.

6.45.9. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/dropDownMenu.jsf?c=menuItem>] you can see the example of **<rich:menuItem>** usage and sources for the given example.

6.46. < rich:menuSeparator >

6.46.1. Description

The **<rich:menuSeparator>** component is used for the definition of a horizontal separator that can be placed between groups or items.

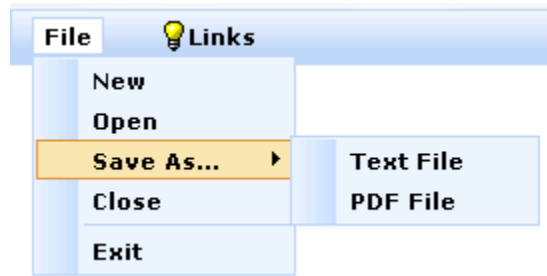


Figure 6.56. <rich:menuSeparator> component

Table 6.160. rich : menuSeparator attributes

| Attribute Name | Description |
|----------------|---|
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| id | Every component may have a unique id that is automatically created if omitted |
| rendered | If "false", this component is not rendered |

Table 6.161. Component identification parameters

| Name | Value |
|------------------|--|
| component-type | org.richfaces.MenuSeparator |
| component-class | org.richfaces.component.html.HtmlMenuSeparator |
| component-family | org.richfaces.DropDownMenu |
| renderer-type | org.richfaces.MenuSeparatorRenderer |
| tag-class | org.richfaces.taglib.MenuSeparatorTag |

6.46.2. Creating the Component with a Page Tag

Here is a simple example as it could be used on a page:

Example:

```
...
<rich:dropDownMenu/>
...
<rich:menuSeparator/>
...
<rich:dropDownMenu/>
...
```

6.46.3. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlMenuSeparator;
...
HtmlMenuSeparator myMenuSeparator = new HtmlMenuSeparator();
...
```

6.46.4. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

There are two ways to redefine the appearance of all `<rich:menuSeparator>` components at once:

- Redefine the corresponding skin parameters
- Add to your style sheets *style classes* used by a `<rich:menuSeparator>` component

6.46.5. Skin Parameters Redefinition

Table 6.162. Skin parameters redefinition for an item

| Skin parameters | CSS properties |
|------------------|------------------|
| panelBorderColor | border-top-color |

6.46.6. Definition of Custom Style Classes

On the screenshot there are classes names that define styles for component elements.

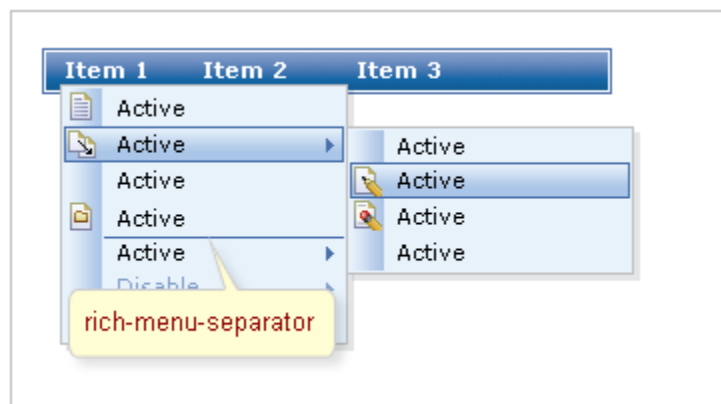


Figure 6.57. Classes names

Table 6.163. Classes names that define separator element appearance.

| Class name | Description |
|---------------------|---|
| rich-menu-separator | Defines styles for a wrapper <code><div></code> element for a separator |

In order to redefine styles for all `<rich:separator>` components on a page using CSS, it's enough to create class with the same name and define necessary properties in it.

To change styles of particular `<rich:separator>` components, define your own style class in the corresponding `<rich:separator>` attributes.

6.46.7. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/dropDownMenu.jsf?c=menuSeparator>] you can see the example of `<rich:menuSeparator>` usage and sources for the given example.

6.47. < rich:effect >

6.47.1. Description

The rich:effect utilizes a set of effects provided by the scriptaculous JavaScript library. It allows to attach effects to JSF components and html tags.

6.47.2. Key Features

- No developers JavaScript writing needed to use it on pages
- Presents scriptaculous JavaScript library functionality

Table 6.164. rich : effect attributes

| Attribute Name | Description |
|----------------|---|
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| disableDefault | Disable default action for target event (append "return false;" to javascript) |
| event | Event on the component or html tag the effect is attached to |
| for | Id of the target component. |
| id | Every component may have a unique id that is automatically created if omitted |
| name | Generated Javascript name. |
| params | Parameters passed to the effect function. Example params="{duration:0.2,from:1.0,to:0.1}" |
| rendered | If "false", this component is not rendered |
| targetId | The id of the element the effect apply to. Might be component id or client id of jsf component or html tag. If targetId is not defined the value of the attribute 'for' or the 'targetId' option effect play its role |
| type | Defines the type of effect. Possible values: "Fade", "Blind", "Opacity". |

Table 6.165. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | org.richfaces.Effect |
| component-class | org.richfaces.component.html.HtmlEffect |
| component-family | org.richfaces.Effect |
| renderer-type | org.richfaces.EffectRenderer |
| tag-class | org.richfaces.taglib.EffectTag |

6.47.3. Creating the Component with a Page Tag

To create the simplest variant of rich:effect on a page, use the following syntax:

Example:

```
...
<rich:effect for="componentId" type="Appear"/>
...
```

6.47.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlRichEffect;
...
HtmlRichEffect myEffect = new HtmlRichEffect();
...
```

6.47.5. Details of Usage

It is possible to use <rich:effect> in two modes:

- attached to the JSF components or html tags and triggered by a particular event. Wiring effect with JSF components might occur on the server or client. Wiring with html tag is possible only on the client side
- invoking from the JavaScript code by an effect name. During the rendering, rich:effect generates the JavaScript function with defined name. When the function is called, the effect is applied

Those a the typical variants of using:

```
...
<!-- attaching by event -->
<rich:panel>
  <rich:effect event="onmouseout" type="Opacity" params="duration:0.8,from:1.0,to:0.3"
  />
  .... panel content ....
</rich:panel>
...
```

```

<!-- invoking from JavaScript -->
<div id="contentDiv">
    ..... div content .....
</div>

<input type="button" onclick="hideDiv({duration:0.7})" value="Hide" />
<input type="button" onclick="showDiv()" value="Show" />

<rich:effect name="hideDiv" for="contentDiv" type="Fade" />
<rich:effect name="showDiv" for="contentDiv" type="Appear" />

<!-- attaching to window on load and applying on particular page element -->
<rich:effect for="window" event="onload" type="Appear"
  params="targetId: 'contentDiv',duration:0.8,from:0.3,to:1.0" />
...

```

The opacity of this panel will be set to 0.3 when the mouse cursor is out set to 1.0 if the mouse is over. The default opacity is set to 0.3 when the page is loaded.

Figure 6.58. Initial:

The opacity of this panel will be set to 0.3 when the mouse cursor is out set to 1.0 if the mouse is over. The default opacity is set to 0.3 when the page is loaded.

Figure 6.59. When the mouse cursor is over:

"name" attribute defines a name of the JavaScript function that is be generated on a page when the component is rendered. You can invoke this function to activate the effect. The function accesses one parameter. It is a set of effect options in JSON format.

"type" attribute defines the type of an effect. For example, "Fade", "Blind", "Opacity". Have a look at scriptaculous documentation [<http://script.aculo.us>] for set of available effect.

"for" attribute defines the id of the component or html tag, the effect will be attached to. Richfaces converts the *"for"* attribute value to the client id of the component if such component is found. If not, the value is left as is for possible wiring with on the DOM element's id on the client side. By default, the target of the effect is the same element that effect pointed to. However, the target element is might be overridden with *"targetId"* option passed with *"params"* attribute of with function parameter.

"params" attribute allows to define the set of options possible for particular effect. For example, 'duration', 'delay', 'from', 'to'. Additionally to the options used by the effect itself, there are two option that might override the rich:effect attribute. Those are:

- *"targetId"* allows to re-define the target of effect. The option is override the value of *"for"* attribute.
- *"type"* defines the effect type. The option is override the value of *"type"* attribute.

You can use a set of effects directly without defining the **<rich:effect>** component on a page if it's convenient for you. For that, load the scriptaculous library to the page with the following code:

Example:

```
...
<a4j:loadScript src="resource://scriptaculous/effect.js" />
...
```

If you do use the `<rich:effect>` component, there is no need to include this library because it's already here.

For more information look at RichFaces Users Forum [<http://jboss.com/index.html?module=bb&op=viewtopic&t=119044>].

6.47.6. Look-and-Feel Customization

`<rich:effect>` has no skin parameters and custom style classes, as the component isn't visual.

6.47.7. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/effect.jsf?c=effect>] you can see the example of `<rich:effect>` usage.

How to save `<rich:effect>` status see on the RichFaces Users Forum [<http://www.jboss.com/index.html?module=bb&op=viewtopic&t=118833>].

6.48. < rich:gmap >

6.48.1. Description

Component that presents the Google map in the JSF applications.

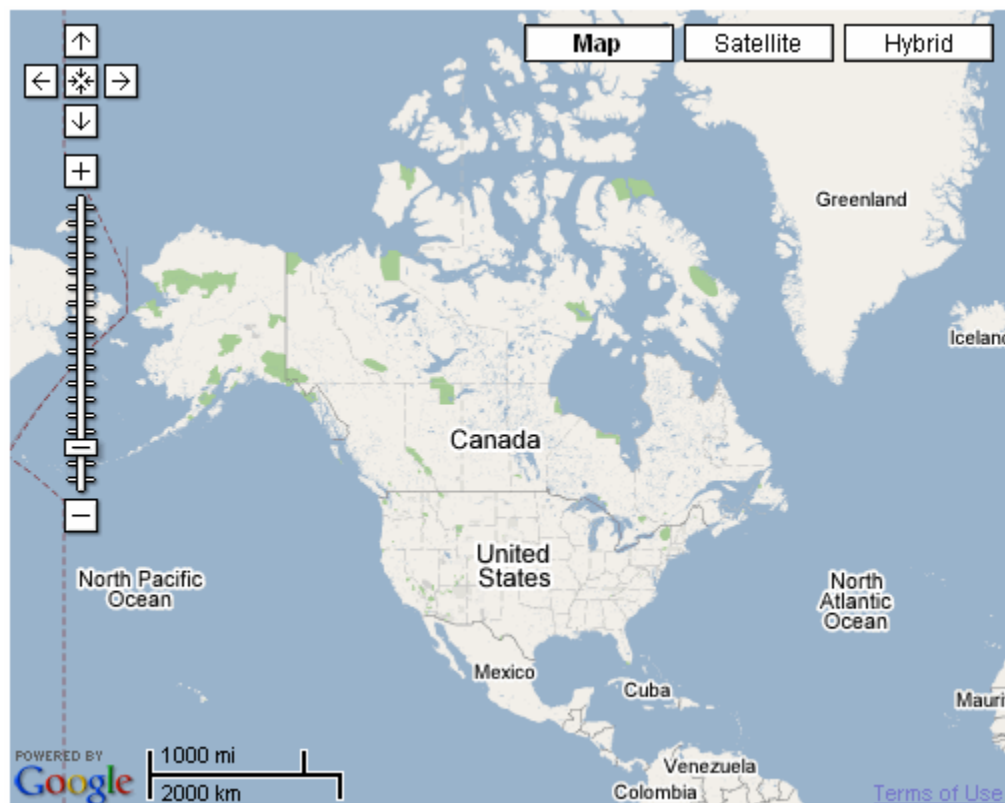


Figure 6.60. `<rich:gmap>` component

6.48.2. Key Features

- Presents all the Google map functionality
- Highly customizable via attributes
- No developers JavaScript writing needed to use on a pages

Table 6.166. rich : gmap attributes

| Attribute Name | Description |
|-----------------------|---|
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| enableContinuousZoom | Enables continuous smooth zooming for selected browsers. The default value is "false" |
| enableDoubleClickZoom | Enables zooming in by a double click. The default value is "false" |
| enableDragging | Enables a map dragging with the mouse. The default value is "true" |
| enableInfoWindow | Enables Info Window. The default value is "true" |
| gmapKey | Google Map key. A single Maps API key is valid for a single "directory" on your web server |
| gmapVar | The JavaScript variable that is used to access the Google Map API. If you have more than one Google Map components on the same page, use individual key for each of them. The default variable name is "map" (without quotes) |
| id | Every component may have a unique id that is automatically created if omitted |
| lat | Initial latitude coordinate in degrees, as a number between -90 and +90 |
| lng | Initial longitude coordinate in degrees, as a number between -180 and +180 |
| mapType | Initial map type. The possible values are G_NORMAL_MAP, G_SATELLITE_MAP, G_HYBRID_MAP. The default value is G_SATELLITE_MAP |
| onclick | HTML: a script expression; a pointer button is clicked |
| ondblclick | HTML: a script expression; a pointer button is double-clicked |

| Attribute Name | Description |
|----------------------|---|
| oninit | JavaScript code invoked just after the Google Map object is initiated. |
| onkeydown | HTML: a script expression; a key is pressed down |
| onkeypress | HTML: a script expression; a key is pressed and released |
| onkeyup | HTML: a script expression; a key is released |
| onmousedown | HTML: script expression; a pointer button is pressed down |
| onmousemove | HTML: a script expression; a pointer is moved within |
| onmouseout | HTML: a script expression; a pointer is moved away |
| onmouseover | HTML: a script expression; a pointer is moved onto |
| onmouseup | HTML: script expression; a pointer button is released |
| rendered | If "false", this component is not rendered |
| showGLargeMapControl | Shows the GLarge control. The default value is "true" |
| showGMapTypeControl | Shows the Type switch control. The default value is "true" |
| showGScaleControl | It shows the scale control. The default value is "true" |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| warningMessage | The warning message that appears if a browser is not compatible with Google Map. The default value is "Your browser does not support Google Maps" |
| zoom | Initial zoom level as a number between 1 and 18. The default value is 17 |

Table 6.167. Component identification parameters

| Name | Value |
|------------------|---------------------------------------|
| component-type | org.richfaces.Gmap |
| component-class | org.richfaces.component.html.HtmlGmap |
| component-family | org.richfaces.Gmap |
| renderer-type | org.richfaces.GmapRenderer |

| Name | Value |
|-----------|------------------------------|
| tag-class | org.richfaces.taglib.GmapTag |

6.48.3. Creating the Component with a Page Tag

Here is a simple example as it could be used on a page:

Example:

```
...
<rich:gmap gmapKey="..." />
...
```

6.48.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlGmap;
...
HtmlGmap myMap = new HtmlGmap();
...
```

6.48.5. Details of Usage

To use *Google Map* in your application, generate a key on Google Map official resource [<http://google.com/apis/maps>]. One key could be used for one directory on the server.

Here are the main settings of initial rendering performed with a component map that are accessible with the following attributes:

- *"zoom"* defines an approximation size (boundary values 1-18)
- *"lat"* specifies an initial latitude coordinate in degrees, as a number between -90 and +90
- *"lng"* specifies an initial longitude coordinate in degrees, as a number between -180 and +180
- *"mapType"* specifies a type of a rendered map (G_NORMAL_MAP, G_SATELLITE_MAP (DEFAULT), G_HYBRID_MAP)

For example, the city of Paris is shown after rendering with the following initial settings: *"lat"* = 48.44, *"lng"* = 2.24 and *"zoom"* = 5.



Figure 6.61. <rich:gmap> initial rendering

It's also possible to set accessible controls on the map with the help of the attributes:

- *"showGMapTypeControl"* determines whether the controls for a map type definition are switched on
- *"showGScaleControl"* determines whether the controls for scaling are switched on
- *"showGLargeMapControl"* determines whether the control for map scale rendering is rendered

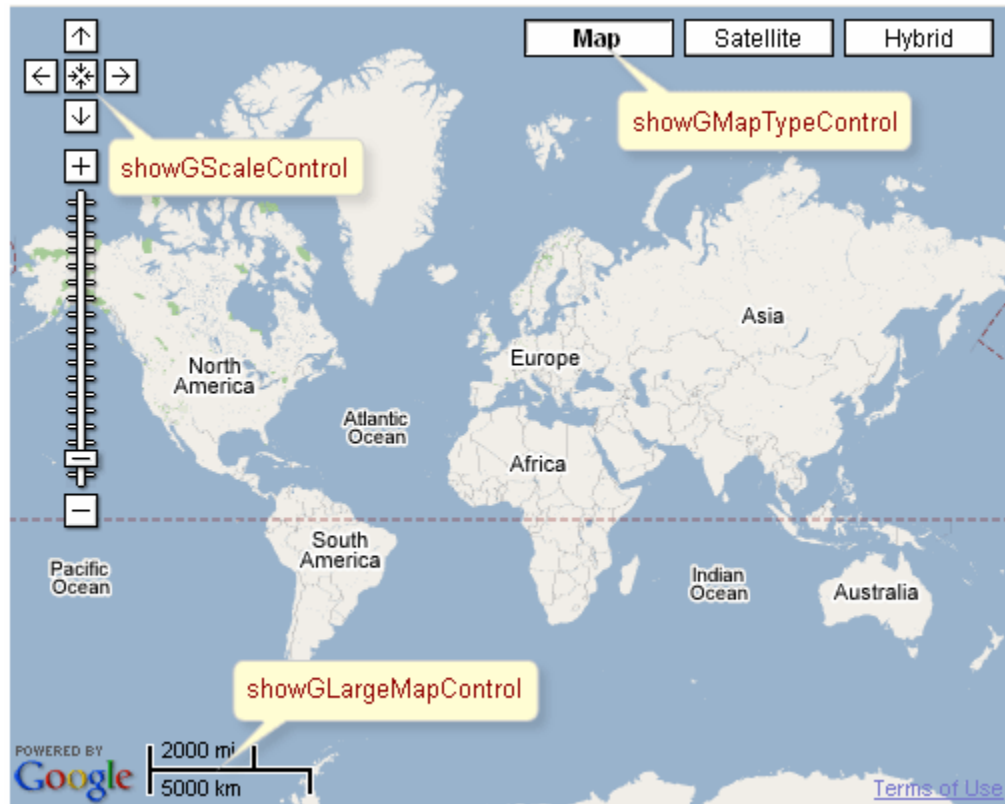


Figure 6.62. <rich:gmap> accessible controls

To set all these parameters and perform some activity (Zoom In/Out etc.) is possible with your JavaScript, i.e. declare a name of an object on a map in the *"gmapVar"* attribute and then call the object directly with API *Google Map*.

For example, to approximate a map for *"gmapVar" = "map"* declared inside the component, call *map.zoomIn()* on an event.

Moreover, to add e.g. some JavaScript effects, events defined on it are used.

- onmouseover
- onclick
- onmouseout
- etc.

Note

Google Map does not support XHTML format of the page. Thus, if you use Facelets and JSF 1.2, do not forget to put the following tags somewhere on the page:

```
...
<f:view contentType="text/html">...</f:view>
...
```

6.48.6. Look-and-Feel Customization

`<rich:gmap>` component isn't tied to skin parameters, as there is no additional elements on it, except the ones provided with *Google Map*.

6.48.7. Definition of Custom Style Classes

Table 6.168. Classes names that define a component appearance

| Class name | Description |
|------------|--|
| rich-gmap | Defines styles for a wrapper <code><div></code> element of a component |

In order to redefine styles for all `<rich:gmap>` components on a page using CSS, it's enough to create class with the same name and define necessary properties in it.

To change styles of particular `<rich:gmap>` components, define your own style class in the corresponding `<rich:gmap>` attribute.

6.48.8. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/gmap.jsf?c=gmap>] you can see the example of `<rich:gmap>` usage and sources for the given example.

6.49. < rich:virtualEarth >

6.49.1. Description

The component presents the Microsoft Virtual Earth map in the JSF applications.

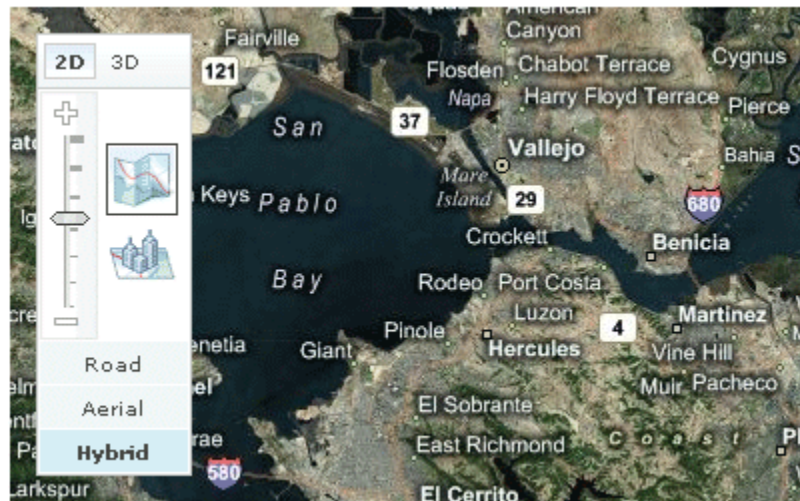


Figure 6.63. <rich:virtualEarth> component

6.49.2. Key Features

- Presents the Microsoft Virtual Earth map functionality

- Highly customizable via attributes
- No developers JavaScript writing is needed to use it on a pages

Table 6.169. rich : virtualEarth attributes

| Attribute Name | Description |
|----------------|---|
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| dashboardSize | Initial map type. The possible values are Normal,Small,Tiny. The default value is Normal |
| id | Every component may have a unique id that is automatically created if omitted |
| lat | Initial latitude coordinate in degrees, as a number between -90 and +90 |
| lng | Initial longitude coordinate in degrees, as a number between -180 and +180 |
| mapStyle | Navigation control size. The possible values are Road,Aerial,Hybrid,Birdseye. The default value is Road |
| onclick | HTML: a script expression; a pointer button is clicked |
| ondblclick | HTML: a script expression; a pointer button is double-clicked |
| onkeydown | HTML: a script expression; a key is pressed down |
| onkeypress | HTML: a script expression; a key is pressed and released |
| onkeyup | HTML: a script expression; a key is released |
| onLoadMap | JavaScript code invoked just after the Virtual Earth object is initiated. |
| onmousedown | HTML: script expression; a pointer button is pressed down |
| onmousemove | HTML: a script expression; a pointer is moved within |
| onmouseout | HTML: a script expression; a pointer is moved away |
| onmouseover | HTML: a script expression; a pointer is moved onto |
| onmouseup | HTML: script expression; a pointer button is released |
| rendered | If "false", this component is not rendered |

| Attribute Name | Description |
|----------------|---|
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| var | The JavaScript variable that is used to access the Virtual Earth API. If you have more than one Virtual Earth components on the same page, use individual key for each of them. The default variable name is "map" (without quotes) |
| version | Virtual earth version, default - "6" |
| zoom | Initial zoom level as a number between 1 and 18. The default value is 17 |

Table 6.170. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | org.richfaces.VirtualEarth |
| component-class | org.richfaces.component.html.HtmlVirtualEarth |
| component-family | org.richfaces.VirtualEarth |
| renderer-type | org.richfaces.VirtualEarthRenderer |
| tag-class | org.richfaces.taglib.VirtualEarthTag |

6.49.3. Creating the Component with a Page Tag

Here is a simple example as it could be used on a page:

Example:

```
...
<rich:virtualEarth lat="..." lng="..." />
...
```

6.49.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlVirtualEarth;
...
HtmlVirtualEarth myMap = new HtmlVirtualEarth();
...
```

6.49.5. Details of Usage

Here are the main settings of initial rendering performed with a component map that are accessible with the following attributes:

- `"zoom"` defines an approximation size (boundary values 1-18)
- `"lat"` specifies an initial latitude coordinate in degrees, as a number between -90 and +90
- `"lng"` specifies an initial longitude coordinate in degrees, as a number between -180 and +180
- `"dashboardSize"` specifies a type of a rendered map (Normal, Small, Tiny)

For example, the city of Paris is shown after rendering with the following initial settings: `"lat"` = 48.833, `"lng"` = 2.40 and `"zoom"` = 11.

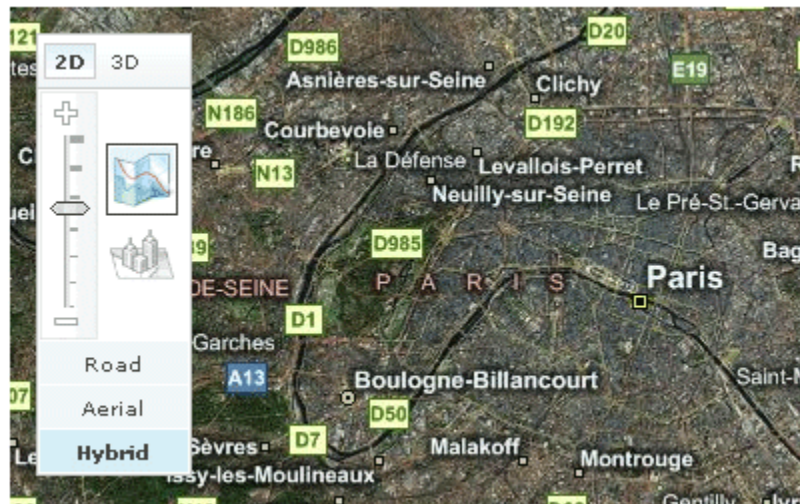


Figure 6.64. `<rich:virtualEarth>` initial rendering

Code for this example is placed below:

Example:

```
...
    <rich:virtualEarth style="width:800px;" id="vm" lat="48.833" lng="2.40"
                        dashboardSize="Normal" zoom="11"
    mapStyle="Hybrid" var="map" />
...
```

To set all these parameters and perform some activity (Zoom In/Out etc.) is possible with your JavaScript, i.e. declare a name of an object on a map in the `"var"` attribute and then call the object directly with API *Microsoft Virtual Earth map*.

For example, to approximate a map for `"var" = "map"` declared inside the component, call `map.ZoomIn()` on an event.

Moreover, to add e.g. some JavaScript effects, events defined on it are used.

- `onmouseover`

- onclick
- onmouseout
- etc.

Note

Virtual Earth does not support XHTML format of the page. Thus, if you use Facelets and JSF 1.2, do not forget to put the following tags somewhere on the page:

```
...
<f:view contentType="text/html">...</f:view>
...
```

6.49.6. Look-and-Feel Customization

`<rich:virtualEarth>` component isn't tied to skin parameters, as there is no additional elements on it, except the ones provided with *Virtual Earth map*.

6.49.7. Definition of Custom Style Classes

Table 6.171. Classes names that define a component appearance

| Class name | Description |
|-------------------|--|
| rich-virtualEarth | Defines styles for a wrapper <code><div></code> element of a component |

In order to redefine styles for all `<rich:virtualEarth>` components on a page using CSS, it's enough to create class with the same name and define necessary properties in it.

To change styles of particular `<rich:virtualEarth>` components, define your own style class in the corresponding `<rich:virtualEarth>` attribute.

6.49.8. Relevant Resources Links

Here [<http://msdn2.microsoft.com/en-us/library/bb429619.aspx>] you can found additional information about Microsoft Virtual Earth map.

6.50. < rich:inputNumberSlider >

6.50.1. Description

A component that lets selecting a number from a numeric region. It's a horizontal aligned scroll-like control with its own input field (optional) present. The keyboard input in a field is possible (optional). Also it's possible to see the current value in the toolTip above a dragged handle control.



Figure 6.65. <rich:inputNumberSlider> component

6.50.2. Key Features

- Fully skinnable control and input elements
- Optional value text field with an attribute-managed position
- Optional disablement of the component on a page
- Optional toolTip to display the current value while a handle is dragged
- Dragged state is stable after the mouse moves
- Optional manual input possible if a text input field is present
- Validation of manual input

Table 6.172. rich : inputNumberSlider attributes

| Attribute Name | Description |
|---------------------|---|
| accesskey | This attribute assigns an access key to an element. An access key is a single character from the document character set. Note: Authors should consider the input method of the expected reader when specifying an accesskey |
| barClass | A name of CSS class for the bar element |
| barStyle | Style for a slider control line |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| converter | Id of Converter to be used or reference to a Converter |
| converterMessage | A ValueExpression enabled attribute that, if present, will be used as the text of the converter message, replacing any message that comes from the converter |
| disabled | When set for a form control, this boolean attribute disables the control for user input |
| enableManualInput | False value for this attribute makes a text field "read-only", so the value can be changed only from a handle |
| handleClass | A name of CSS class for a control handle element |
| handleSelectedClass | A name of CSS class for a selected control handle element |
| id | Every component may have a unique id that is automatically created if omitted |
| immediate | |

| Attribute Name | Description |
|-----------------|---|
| | A flag indicating that this component value must be converted and validated immediately (that is, during Apply Request Values phase), rather than waiting until a Process Validations phase |
| inputClass | Style Class attribute for a text field |
| inputPosition | If "right" the InputText Box would be rendered on the right side of the ruler |
| inputSize | Similar to the "Size" attribute of h:inputText |
| inputStyle | Style attribute for text field |
| label | A localized user presentable name for this component. |
| maxLength | When the type attribute has the value "text" or "password", this attribute specifies the maximum number of characters the user may enter. This number may exceed the specified size, in which case the user agent should offer a scrolling mechanism. The default value for this attribute is an unlimited number |
| maxValue | Attribute to set an "end" value |
| minValue | Attribute to set a "start" value |
| onblur | HTML: script expression; the element lost the focus |
| onchange | HTML: script expression; the element value was changed |
| onclick | HTML: a script expression; a pointer button is clicked |
| ondblclick | HTML: a script expression; a pointer button is double-clicked |
| onerror | This error is called when a non-number value or a number value that is out of the range is input |
| onfocus | HTML: script expression; the element got the focus |
| oninputclick | HTML: a script expression; a pointer button is clicked |
| oninputdblclick | HTML: a script expression; a pointer button is double-clicked |
| oninputkeydown | HTML: a script expression; a key is pressed down |
| oninputkeypress | HTML: a script expression; a key is pressed and released |

| Attribute Name | Description |
|--------------------|--|
| oninputkeyup | HTML: a script expression; a key is released |
| oninputmousedown | HTML: script expression; a pointer button is pressed down |
| oninputmousemove | HTML: a script expression; a pointer is moved within |
| oninputmouseout | HTML: a script expression; a pointer is moved away |
| oninputmouseover | HTML: a script expression; a pointer is moved onto |
| oninputmouseup | HTML: script expression; a pointer button is released |
| onmousedown | HTML: script expression; a pointer button is pressed down |
| onmousemove | HTML: a script expression; a pointer is moved within |
| onmouseout | HTML: a script expression; a pointer is moved away |
| onmouseover | HTML: a script expression; a pointer is moved onto |
| onmouseup | HTML: script expression; a pointer button is released |
| onselect | HTML: script expression; The onselect event occurs when a user selects some text in a text field. This attribute may be used with the INPUT and TEXTAREA elements |
| rendered | If "false", this component is not rendered |
| required | If "true", this component is checked for non-empty input |
| requiredMessage | A ValueExpression enabled attribute that, if present, will be used as the text of the validation message for the "required" facility, if the "required" facility is used |
| showBoundaryValues | If the min/max values are shown on the right/left borders of a control. Default=true |
| showInput | False value for this attribute makes text a field invisible |
| showToolTip | If the current value will be shown in the tooltip when a handle control in a "dragged" state.Default=true. |
| step | Parameter that determines a step between the nearest values while using a handle |
| style | Styles for main div element of the slider control |
| styleClass | Name of a CSS class |

| Attribute Name | Description |
|---------------------|--|
| tabindex | This attribute specifies the position of the current element in the tabbing order for the current document. This value must be a number between 0 and 32767. User agents should ignore leading zeros |
| tipClass | A name of CSS class for the tool tip element |
| tipStyle | A style for the tool tip element |
| validator | MethodBinding pointing at a method that is called during Process Validations phase of the request processing lifecycle, to validate the current value of this component |
| validatorMessage | A ValueExpression enabled attribute that, if present, will be used as the text of the validator message, replacing any message that comes from the validator |
| value | The initial value to set when rendered for the first time |
| valueChangeListener | Listener for value changes |
| width | The width of a slider control |

Table 6.173. Component identification parameters

| Name | Value |
|------------------|--|
| component-type | org.richfaces.inputNumberSlider |
| component-class | org.richfaces.component.html.HtmlInputNumberSlider |
| component-family | org.richfaces.inputNumberSlider |
| renderer-type | org.richfaces.InputNumberSliderRenderer |
| tag-class | org.richfaces.taglib.InputNumberSliderTag |

6.50.3. Creating the Component with a Page Tag

Here is a simple example as it could be used on a page:

Example:

```
...
<rich:inputNumberSlider minValue="0" maxValue="100" step="1"/>
...
```

6.50.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlInputNumberSlider;
...
HtmlInputNumberSlider mySlider = new HtmlInputNumberSlider();
...
```

6.50.5. Details of Usage

`<rich:inputNumberSlider>` is used to facilitate your data input with rich UI Controls.

Here is the simplest variant of a slider definition with `"minValue"`, `"maxValue"` and `"step"` (on default = "1") attributes, which define the beginning and the end of a numerical area and a slider property step.

Example:

```
<rich:inputNumberSlider></rich:inputNumberSlider>
```

It generates on a page:



Figure 6.66. Generated `<rich:inputNumberSlider>`

Using `"showInput"` (default is true) and `"enableManualInput"` (default value is true) attributes, it's possible to output the input area near the slider, and make it read-only or editable.

To remove input area use `"showInput=false"`:

Example:

```
<rich:inputNumberSlider minValue="1" maxValue="100" showInput="false"/>
```

It looks at page like:



Figure 6.67. `<rich:inputNumberSlider>` without input field

It's also possible to switch off displaying of "boundary values" and a tooltip showing on a handle drawing. This could be performed with the help of the component defined attributes: `"showBoundaryValues"` which is responsible for "boundary values" displaying (default is true) and `"showToolTip"` which is responsible for tooltip displaying (default is true).

Moreover, to add e.g. some JavaScript effects, events defined on it are used.

- `onchange`
- `onmouseover`
- `onclick`

- onfocus
- onmouseout
- etc.

6.50.6. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

There are two ways to redefine the appearance of all `<rich:inputNumberSlider>` components at once:

- Redefine the corresponding skin parameters
- Add to your style sheets *style classes* used by a `<rich:inputNumberSlider>` component

6.50.7. Skin Parameters Redefinition

Table 6.174. Skin parameters redefinition for a bar

| Skin parameters | CSS properties |
|------------------------|------------------|
| controlBackgroundColor | background-color |

Table 6.175. Skin parameters redefinition for numbers

| Skin parameters | CSS properties |
|-------------------|----------------|
| generalFamilyFont | font-family |
| generalSizeFont | font-size |
| generalTextColor | color |
| panelBorderColor | border-color |
| generalSizeFont | line-height |

Table 6.176. Skin parameters redefinition for a text field

| Skin parameters | CSS properties |
|------------------------|---------------------|
| controlBackgroundColor | background-color |
| generalFamilyFont | font-family |
| generalSizeFont | font-size |
| controlTextColor | color |
| panelBorderColor | border-color |
| subBorderColor | border-bottom-color |

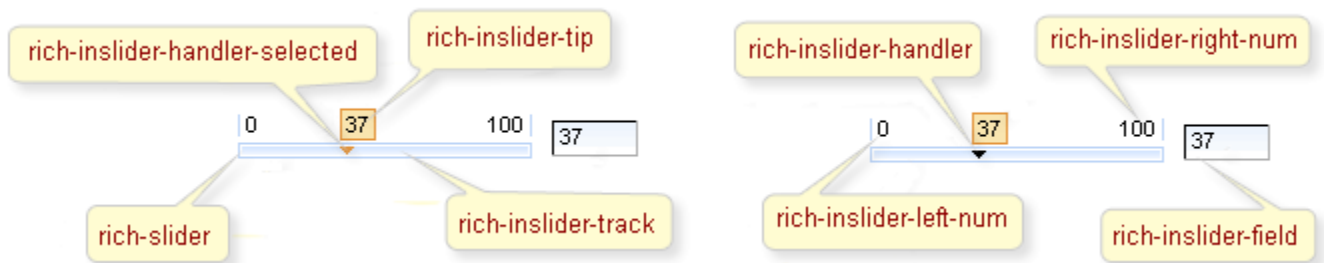
| Skin parameters | CSS properties |
|-----------------|--------------------|
| subBorderColor | border-right-color |

Table 6.177. Skin parameters redefinition for a hint

| Skin parameters | CSS properties |
|--------------------|------------------|
| tipBackgroundColor | background-color |
| tipBorderColor | border-color |
| generalFamilyFont | font-family |
| generalSizeFont | font-size |

6.50.8. Definition of Custom Style Classes

On the screenshot there are classes names that define styles for component elements.

**Figure 6.68. Style classes****Table 6.178. Classes names that define a component appearance**

| Class name | Description |
|--------------------------------|---|
| rich-slider | Defines styles for a wrapper table element of a component |
| rich-inslider-track | Defines styles for a bar |
| rich-inslider-handler | Defines styles for a slider handler |
| rich-inslider-handler-selected | Defines styles for a selected handler |
| rich-inslider-field | Defines styles for a text field |
| rich-inslider-right-num | Defines styles for the right number |
| rich-inslider-left-num | Defines styles for the left number |
| rich-inslider-tip | Defines styles for a hint |

In order to redefine styles for all `<rich:inputNumberSlider>` components on a page using CSS, it's enough to create classes with the same names (possible classes could be found in the table above) and define necessary properties in them. An example is placed below:

Example:

```
...
.rich-inslider-tip{
    background-color: #FFDAB9;
    font-family: Arial Black;
}
...
```

This is a result:

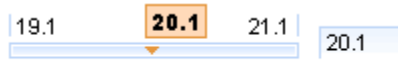


Figure 6.69. Redefinition styles with predefined classes

In the example a tip background color and font family was changed.

Also it's possible to change styles of particular `<rich:inputNumberSlider>` component. In this case you should create own style classes and use them in corresponding `<rich:inputNumberSlider>` *styleClass* attributes. An example is placed below:

Example:

```
...
.myClass{
    font-style: italic;
}
...
```

The *"inputClass"* attribute for `<rich:inputNumberSlider>` is defined as it's shown in the example below:

Example:

```
<rich: inputNumberSlider ... inputClass="myClass" />
```

This is a result:

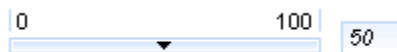


Figure 6.70. Redefinition styles with own classes and styleClass attributes

As it could be seen on the picture above, the font style for input text was changed.

6.50.9. Relevant Resources Links

Here [\[http://livedemo.exadel.com/richfaces-demo/richfaces/inputNumberSlider.jsf?c=inputNumberSlider\]](http://livedemo.exadel.com/richfaces-demo/richfaces/inputNumberSlider.jsf?c=inputNumberSlider) you can see the example of `<rich:inputNumberSlider>` usage and sources for the given example.

6.51. < rich:inputNumberSpinner >

6.51.1. Description

A single line input field that lets selecting a number using controls near a text field. It's possible to change a value using "Up/Down" keyboard keys. The keyboard input in a field is possible if it isn't locked by the *"manualInput"* attribute. When arrow controls are pressed, the cursor can be moved in any way without losing a dragged state.



Figure 6.71. <rich:InputNumberSpinner> component

6.51.2. Key Features

- Fully skinnable control and input elements
- 3D look and feel with an easily customizable appearance
- Attribute-managed positions of the controls (inside/outside of the input field)
- Keyboard controls support
- Optional disablement of the component on a page
- Optional *"cycled"* mode of scrolling values
- Optional manual/controls-only input into a value text field
- Validation of manual input

Table 6.179. rich : inputNumberSpinner attributes

| Attribute Name | Description |
|------------------|---|
| accesskey | This attribute assigns an access key to an element. An access key is a single character from the document character set. Note: Authors should consider the input method of the expected reader when specifying an accesskey |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| converter | Id of Converter to be used or reference to a Converter |
| converterMessage | A ValueExpression enabled attribute that, if present, will be used as the text of the converter message, replacing any message that comes from the converter |
| cycled | If "true" after the current value reaches the border value it is reversed to another border value after next |

| Attribute Name | Description |
|-------------------|---|
| | increasing/decreasing. In other case possibilities of next increasing (or decreasing) will be locked |
| disabled | When set for a form control, this boolean attribute disables the control for user input |
| enableManualInput | if "false" user's input to the text field using keyboard will be locked |
| id | Every component may have a unique id that is automatically created if omitted |
| immediate | A flag indicating that this component value must be converted and validated immediately (that is, during Apply Request Values phase), rather than waiting until a Process Validations phase |
| inputClass | Class attribute for text field |
| inputSize | Attribute specifies the initial length of input in characters. Default value is 10 |
| inputStyle | Style attribute for text field |
| label | A localized user presentable name for this component. |
| maxValue | Maximum value |
| minValue | Minimum value |
| onblur | HTML: script expression; the element lost the focus |
| onchange | HTML: script expression; the element value was changed |
| onclick | HTML: a script expression; a pointer button is clicked |
| ondblclick | HTML: a script expression; a pointer button is double-clicked |
| onmousedown | HTML: a script expression; a button "Down" is clicked |
| onerror | HTML: a script expression; event fires whenever an JavaScript error occurs |
| onfocus | HTML: script expression; the element got the focus |
| oninputclick | HTML: a script expression; a pointer button is clicked |
| oninputdblclick | HTML: a script expression; a pointer button is double-clicked |
| oninputkeydown | HTML: a script expression; a key is pressed down |

| Attribute Name | Description |
|------------------|--|
| oninputkeypress | HTML: a script expression; a key is pressed and released |
| oninputkeyup | HTML: a script expression; a key is released |
| oninputmousedown | HTML: script expression; a pointer button is pressed down |
| oninputmousemove | HTML: a script expression; a pointer is moved within |
| oninputmouseout | HTML: a script expression; a pointer is moved away |
| oninputmouseover | HTML: a script expression; a pointer is moved onto |
| oninputmouseup | HTML: script expression; a pointer button is released |
| onmousedown | HTML: script expression; a pointer button is pressed down |
| onmousemove | HTML: a script expression; a pointer is moved within |
| onmouseout | HTML: a script expression; a pointer is moved away |
| onmouseover | HTML: a script expression; a pointer is moved onto |
| onmouseup | HTML: script expression; a pointer button is released |
| onselect | HTML: script expression; The onselect event occurs when a user selects some text in a text field. This attribute may be used with the INPUT and TEXTAREA elements |
| onupclick | HTML: a script expression; a button "Up" is clicked |
| rendered | If "false", this component is not rendered |
| required | If "true", this component is checked for non-empty input |
| requiredMessage | A ValueExpression enabled attribute that, if present, will be used as the text of the validation message for the "required" facility, if the "required" facility is used |
| step | Parameter that determines the step between nearest values while using controls |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| tabindex | This attribute specifies the position of the current element in the tabbing order for the current document. |

| Attribute Name | Description |
|---------------------|---|
| | This value must be a number between 0 and 32767. User agents should ignore leading zeros |
| validator | MethodBinding pointing at a method that is called during Process Validations phase of the request processing lifecycle, to validate the current value of this component |
| validatorMessage | A ValueExpression enabled attribute that, if present, will be used as the text of the validator message, replacing any message that comes from the validator |
| value | The initial value to set when rendered for the first time |
| valueChangeListener | Listener for value changes |

Table 6.180. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | org.richfaces.inputNumberSpinner |
| component-class | org.richfaces.component.html.HtmlInputNumberSpinner |
| component-family | org.richfaces.inputNumberSpinner |
| renderer-type | org.richfaces.InputNumberSpinnerRenderer |
| tag-class | org.richfaces.taglib.InputNumberSpinnerTag |

6.51.3. Creating the Component with a Page Tag

Here is a simple example as it could be used on a page:

Example:

```
...
<rich:inputNumberSpinner minValue="0" maxValue="100" step="1"/>
...
```

6.51.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlInputNumberSpinner;
...
HtmlInputNumberSpinner mySpinner = new HtmlInputNumberSpinner ();
...
```

6.51.5. Details of Usage

<rich:inputNumberSpinner> is used to facilitate your data input with rich UI Controls.

Here is the simplest variant of spinner definition with *"minValue"*, *"maxValue"* and *"step"* (on default = "1") attributes, which define the beginning and the end of numerical area and a spinner step.

Example:

```
...
<rich:inputNumberSpinner minValue="1" maxValue="100" />
...
```

It generates on a page:



Figure 6.72. Generated <rich:inputNumberSpinner>

There are also several attributes to define functionality peculiarities:

- *"cycled"* if the attribute is *"true"* after the current value reaches the border value it's be reversed to another border value after next increasing/decreasing. In other case possibilities of next increasing/decreasing are locked
- *"disabled"* is an attribute that defines whether a component is active on a page
- *"manualInput"* is an attribute that defines whether a keyboard input is possible or only UI controls could be used

Moreover, to add e.g. some JavaScript effects, events defined on it are used

- onchange
- onmouseover
- onclick
- onfocus
- onmouseout
- etc.

6.51.6. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

There are two ways to redefine the appearance of all **<rich:inputNumberSpinner>** components at once:

- Redefine the corresponding skin parameters
- Add to your style sheets *style classes* used by a **<rich:inputNumberSpinner>** component

6.51.7. Skin Parameters Redefinition

Table 6.181. Skin parameters redefinition for a container

| Skin parameters | CSS properties |
|------------------------|---------------------|
| controlBackgroundColor | background-color |
| panelBorderColor | border-color |
| subBorderColor | border-bottom-color |
| subBorderColor | border-right-color |

Table 6.182. Skin parameters redefinition for an input field

| Skin parameters | CSS properties |
|------------------|----------------|
| buttonSizeFont | font-size |
| buttonFamilyFont | font-family |

6.51.8. Definition of Custom Style Classes

On the screenshot there are classes names that define styles for component elements.

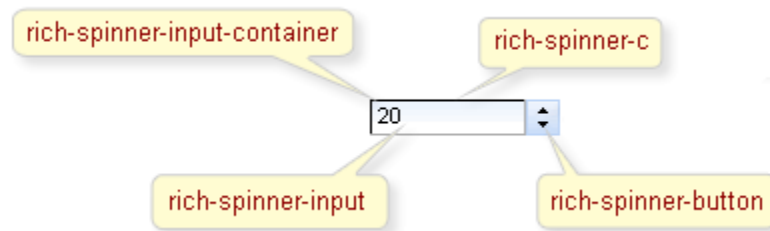


Figure 6.73. Style classes

Table 6.183. Classes names that define a component appearance

| Class name | Description |
|------------------------------|---|
| rich-spinner-c | Defines styles for a wrapper table element of a component |
| rich-spinner-input-container | Defines styles for a container |
| rich-spinner-input | Defines styles for an input field |
| rich-spinner-button | Defines styles for a button |

In order to redefine styles for all `<rich:inputNumberSpinner>` components on a page using CSS, it's enough to create classes with the same names (possible classes could be found in the table above) and define necessary properties in them. An example is placed below:

Example:

```
...
.rich-spinner-input{
    font-style:italic;
}
...
```

This is a result:

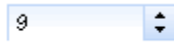


Figure 6.74. Redefinition styles with predefined classes

In the example an input text font style was changed.

Also it's possible to change styles of particular `<rich:inputNumberSpinner>` component. In this case you should create own style classes and use them in corresponding `<rich:inputNumberSpinner>` `styleClass` attributes. An example is placed below:

Example:

```
...
.myClass{
    font-family: Arial Black;
}
...
```

The `"inputClass"` attribute for `<rich:inputNumberSpinner>` is defined as it's shown in the example below:

Example:

```
<rich: inputNumberSpinner ... inputClass="myClass" />
```

This is a result:

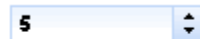


Figure 6.75. Redefinition styles with own classes and styleClass attributes

As it could be seen on the picture above, the font family for input text was changed.

6.51.9. Relevant Resources Links

Here [\[http://livedemo.exadel.com/richfaces-demo/richfaces/inputNumberSpinner.jsf?c=inputNumberSpinner\]](http://livedemo.exadel.com/richfaces-demo/richfaces/inputNumberSpinner.jsf?c=inputNumberSpinner) you can see the example of `<rich:inputNumberSpinner>` usage and sources for the given example.

6.52. < rich:insert >

6.52.1. Description

The **<rich:insert>** component is used for highlighting, source code inserting and, optionally, format the file from the application context into the page.

6.52.2. Key Features

- Source code highlighting
- Variety of formats for source code highlighting

Table 6.184. rich : insert attributes

| Attribute Name | Description |
|----------------|---|
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| encoding | Attribute defines encoding for inserted content |
| errorContent | Attribute defines the alternative content that will be shown in case component cannot read the resource defined with 'src' attribute. If "errorContent" attribute is not defined, the component shown the actual error message in the place where the content is expected |
| highlight | Defines a type of code |
| id | Every component may have a unique id that is automatically created if omitted |
| rendered | If "false", this component is not rendered |
| src | Defines the path to the file with source code |

Table 6.185. Component identification parameters

| Name | Value |
|------------------|--|
| component-type | org.richfaces.ui.Insert |
| component-class | org.richfaces.ui.component.html.HtmlInsert |
| component-family | org.richfaces.ui.Insert |
| renderer-type | org.richfaces.ui.InsertRenderer |
| tag-class | org.richfaces.ui.taglib.InsertTag |

6.52.3. Creating the Component with a Page Tag

To create the simplest variant on a page use the following syntax:

Example:

```
...
    <rich:insert src="/pages/sourcePage.xhtml" highlight="xhtml"/>
...
```

6.52.4. Creating the Component Dynamically Using Java**Example:**

```
import org.richfaces.ui.component.html.HtmlInsert;
...
HtmlInsert myInsert = new HtmlInsert();
...
```

6.52.5. Details of Usage

There are two basic attributes. The *"src"* attribute defines the path to the file with source code. The *"highlight"* attribute defines the type of a syntax highlighting.

If *"highlight"* attribute is defined and JHighlight [<https://jhighlight.dev.java.net/>] open source library is in the classpath, the text from the file will be formatted and colorized.

An example is placed below.

Example:

```
...
    <rich:insert src="/pages/sourcePage.xhtml" highlight="xhtml"/>
...
```

The result of using **<rich:insert>** component is shown on the picture:

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:a4j="http://richfaces.org/a4j"
    xmlns:rich="http://richfaces.org/rich">

    <h:form>
        <rich:panel>
            <a4j:commandButton value="Set Name to Alex" reRender="rep" >
                <a4j:actionparam name="username" value="Alex" assignTo="#{userBean.name}"/>
            </a4j:commandButton>
            <rich:spacer width="20" />
            <a4j:commandButton value="Set Name to John" reRender="rep" >
                <a4j:actionparam name="username" value="John" assignTo="#{userBean.name}"/>
            </a4j:commandButton>
        </rich:panel>
        <rich:panel>
            <h:outputText id="rep" value="Selected Name:#{userBean.name}"/>
        </rich:panel>
    </h:form>
</ui:composition>
```

Figure 6.76. Source code highlighting

The `<rich:insert>` component provides the same functionality as JHighlight [<https://jhighlight.dev.java.net/>]. Thus, all names of highlight style classes for source code of particular language could be changed to your names, which are used by the JHighlight [<https://jhighlight.dev.java.net/>] library.

6.52.6. Look-and-Feel Customization

`<rich:insert>` has no skin parameters and custom style classes, as the component doesn't have own visual representation.

6.52.7. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/insert.jsf?c=insert>] you can see the example of `<rich:insert>` usage and sources for the given example.

6.53. < rich:jQuery >

6.53.1. Description

The `<rich:jQuery>` allows to apply styles and behaviour to DOM objects.

6.53.2. Key Features

- Presents jQuery JavaScript framework functionality
- Able to apply onto JSF components and other DOM objects.
- Works without conflicts with prototype.js library

Table 6.186. rich : jQuery attributes

| Attribute Name | Description |
|----------------|---|
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| id | Every component may have a unique id that is automatically created if omitted |
| name | The name of a function that will be generated to execute a query. The "name" attribute is required if "timing" attribute equals to "onJScall" |
| query | The query string that is executed for a given selector. |
| rendered | If "false", this component is not rendered |
| selector | Selector for query. The "selector" attribute uses defined by w3c consortium syntax for CSS rule selector with some jQuery extensions. |
| timing | The attribute that defines when to perform the query. The possible values are "immediate","onload" and |

| Attribute Name | Description |
|----------------|---|
| | "onJScall". "immediate" performs the query right away. "onload" adds the task to the time when a document is loaded (the DOM tree is created). "onJScall" allows to invoke the query by Javascript function name defined with "name" attribute. The default value is "immediate". |

Table 6.187. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | org.richfaces.JQuery |
| component-class | org.richfaces.component.html.HtmljQuery |
| component-family | org.richfaces.JQuery |
| renderer-type | org.richfaces.JQueryRenderer |
| tag-class | org.richfaces.taglib.JQueryTag |

6.53.3. Creating the Component with a Page Tag

To create the simplest variant on a page, use the following syntax:

Example:

```
...
    <rich:jQuery selector="#customList tr:odd" timing="onload"
    query="addClass(odd)" />
...
```

6.53.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmljQuery;
...
HtmljQuery myjQuery = new HtmljQuery();
...
```

6.53.5. Details of Usage

`<rich:jQuery>` can be used in two main modes:

- as a one-time query applied immediately or on a document ready event
- as a JavaScript function that can be invoked from the JavaScript code

The mode is chosen with *"timing"* attribute that has the following options:

- immediate - applying a query immediately

- `onload` - applying a query when a document is loaded
- `onJScall` - applying a query by invoked JavaScript function defined with the `"name"` attribute

Definition of the `"name"` attribute is mandatory when the value of `"timing"` attribute is `"onJScall"`. If the `"name"` attribute is defined when `"timing"` value equals to `"immediate"` or `"onload"`, the query is applied according to this value, but you still have an opportunity to invoke it by a function name.

The `"selector"` attribute defines an object or a list of objects. The query is defined with the `"query"` attribute.

Here is an example of how to highlight odd rows in a table:

Example:

```
...
<style>
    .odd {
        background-color: #FFC;
    }
</style>
...
```

```
...
<rich:table id="customList" ...>
    ...
</rich:table>
...
<rich:jQuery selector="#customList tr:odd" timing="onload" query="addClass(odd)"
/>
...
```

The `"selector"` attribute uses defined by w3c consortium syntax for CSS rule selector [<http://www.w3.org/TR/REC-CSS2/selector.html>] with some jQuery extensions.

Those are typical examples of using selector in the `<rich:jQuery>` component.

Table 6.188. Examples of using selector

| Selector | Comment |
|--------------------------------|---|
| "p[a]" | In a document all "p" tags with "a" tag inside are selected |
| "ul/li" | All "li" elements of unordered "ul" lists are selected |
| "p.foo[a]" | All "p" tags with "foo" class and inserted "a" tag are selected |
| "input[@name=bar]" | All "input" tags with "name" attribute which value is "bar" are selected |
| "input[@type=radio][@checked]" | All "input" tags with attribute "type"="radio" and attribute value = "checked" are selected |

| Selector | Comment |
|------------------|--|
| "p,span,td" | All tag elements "p" or "span" or "td" are selected |
| "p#secret" | "p" paragraph element with "id" identification = "secret" is selected |
| "p span" | "span" tag is a (direct or non-direct) child of "p" tag. If it's necessary, use "p > span" or "p/span" is selected |
| "p[@foo^=bar]" | "p" tag containing "foo" attribute with textual value beginning with "bar" word is selected |
| "p[@foo\$=bar] " | "p" tag containing "foo" attribute with textual value ending with "bar" word is selected |
| "p[@foo*=bar] " | "p" tag with "foo" attribute containing substring "bar" in any place is selected |
| "p//span " | "span" tag that is a (direct or non-direct) child of "p" tag is selected |
| "p/../span " | "span" tag that is a grandchild of "p" tag is selected |

In addition, RichFaces allows using either a component id or client id if you apply the query to a JSF component. When you define a selector, RichFaces examines its content and tries to replace the defined in the selector id with a component id if it's found.

For example, you have the following code:

```
...
<h:form id="form">
    ...
    <h:panelGrid id="menu">
        <h:graphicImage ... />
        <h:graphicImage ... />
        ...
    </h:panelGrid>
</h:form>
...
```

The actual id of the `<h:panelGrid>` table in the browser DOM is "form:menu". However, you still can reference to images inside this table using the following selector:

```
...
<rich:jQuery selector="#menu img" query="..." />
...
```

You can define the exact id in the selector if you want. The following code reference to the same set of a DOM object:

```
...
<rich:jQuery selector="#form\\:menu img" query="..." />
```

...

Pay attention to double slashes that escape a colon in the id.

In case when the *"name"* attribute is defined, **<rich:jQuery>** generates a JavaScript function that might be used from any place of JavaScript code on a page.

There is an example of how to enlarge the picture smoothly on a mouse over event and return back to the normal size on mouse out:

```
...
    <h:graphicImage onmouseover="enlargePic(this)" width="50"
    value="/images/price.png"
        onmouseover="enlargePic(this, {pwidth:'60px'})"
    onmouseover="releasePic(this)" />
    <h:graphicImage onmouseover="enlargePic(this)" width="50"
    value="/images/discount.png"
        onmouseover="enlargePic(this, {pwidth:'100px'})"
    onmouseover="releasePic(this)" />
    ...
    <rich:jQuery name="enlargePic" timing="onJScall"
    query="animate({width:param.pwidth})" />
    <rich:jQuery name="releasePic" timing="onJScall"
    query="animate({width:'50px'})" />
    ...
```

The JavaScript could use two parameters. The first parameter is a replacement for the selector attribute. Thus, you can share the same query, applying it to the different DOM objects. You can use a literal value or a direct reference for an existing DOM object. The second parameter can be used to path the specific value inside the query. The JSON syntax is used for the second parameter. The "param." namespace is used for referencing data inside the parameter value.

<rich:jQuery> adds styles and behavior to the DOM object dynamically. This means if you replace something on a page during an Ajax response, the applied artifacts will be overwritten. But you are allowed to apply them again after the Ajax response is complete.

Usually, it could be done with reRendering the **<rich:jQuery>** components in the same Ajax interaction with the components these queries are applied to. Note, that queries with *"timing"* attribute set to "onload" are not invoked even if the query is reRendered, because a DOM document is not fully reloaded during the Ajax interaction. If you need to re-applies query with "onload" value of *"timing"* attribute, define the *"name"* attribute and invoke the query by name in the *"oncomplete"* attribute of the Ajax component.

RichFaces includes jQuery JavaScript framework. You can use the futures of jQuery directly without defining the **<rich:jQuery>** component on a page if it is convenient for you. To start using the jQuery feature on the page, include the library into a page with the following code:

```
...
    <a4j:loadScript src="resource://jquery.js"/>
    ...
```

Refer to the jQuery documentation [<http://docs.jquery.com/>] for the right syntax. Remember to use jQuery() function instead of \$(), as soon as jQuery works without conflicts with prototype.js.

6.53.6. Look-and-Feel Customization

`<rich:jQuery>` has no skin parameters and custom style classes, as the component isn't visual.

6.53.7. Relevant Resources Links

More information about jQuery framework and its features you can read here [<http://jquery.com/>].

How to use jQuery with other libraries see here [http://docs.jquery.com/Using_jQuery_with_Other_Libraries].

6.54. < rich:listShuttle >

6.54.1. Description

The `<rich:listShuttle>` component is used for moving chosen items from one list into another with their optional reordering there.



Figure 6.77. `<rich:ListShuttle>` component

6.54.2. Key Features

- Highly customizable look and feel
- Reordering possibility for lists items
- Multiple selection of lists items
- Keyboard support

Table 6.189. `rich : listShuttle` attributes

| Attribute Name | Description |
|---------------------------------|---|
| <code>activeItem</code> | Stores active item |
| <code>ajaxKeys</code> | Defines strings that are updated after an Ajax request |
| <code>binding</code> | The attribute takes a value-binding expression for a component property of a backing bean |
| <code>bottomControlClass</code> | CSS class for bottom control |
| <code>bottomControlLabel</code> | Defines a label for a bottom control |

| Attribute Name | Description |
|---------------------------|---|
| columnClasses | Comma-separated list of CSS classes for columns |
| componentState | It defines EL-binding for a component state for saving or redefinition |
| controlsType | Defines type of a control: button or none. |
| converter | Id of Converter to be used or reference to a Converter |
| copyAllControlClass | CSS class for copyAll control |
| copyAllControlLabel | Defines a label for a copyAll control |
| copyControlClass | CSS class for copy control |
| copyControlLabel | Defines a label for a copy control |
| disabledControlClass | CSS class for a disabled control |
| downControlClass | CSS class for down control |
| downControlLabel | Defines a label for a down control |
| fastMoveControlsVisible | If "false", 'Copy All' and 'Remove All' controls aren't displayed |
| fastOrderControlsVisible | If "false", 'Top' and 'Bottom' controls aren't displayed. |
| first | A zero-relative row number of the first row to display |
| id | Every component may have a unique id that is automatically created if omitted |
| immediate | A flag indicating that this component value must be converted and validated immediately (that is, during Apply Request Values phase), rather than waiting until a Process Validations phase |
| listClass | CSS class for a list |
| listsHeight | Defines height of the list |
| moveControlsVerticalAlign | Customize vertically position of move controls relatively to lists |
| moveControlsVisible | If "false", 'Copy' and 'Remove' controls aren't displayed |
| onbottomclick | A JavaScript event handler; a button "Bottom" is clicked |
| onclick | HTML: a script expression; a pointer button is clicked |
| oncopyallclick | A JavaScript event handler; a button "Copy All" is clicked |

| Attribute Name | Description |
|----------------------------|--|
| oncopyclick | HTML: a script expression; a button "Copy" is clicked |
| ondblclick | HTML: a script expression; a pointer button is double-clicked |
| onmousedown | A JavaScript event handler; a button "Down" is clicked |
| onlistchanged | A JavaScript event handler called on a list change operation |
| onmousemove | HTML: a script expression; a pointer is moved within |
| onmouseout | HTML: a script expression; a pointer is moved away |
| onmouseover | HTML: a script expression; a pointer is moved onto |
| onorderchanged | HTML: script expression; called after ordering action |
| onremoveallclick | A JavaScript event handler; a button "Remove All" is clicked |
| onremoveclick | A JavaScript event handler; a button "Remove" is clicked |
| ontopclick | A JavaScript event handler; a button "Top" is clicked |
| onupclick | HTML: a script expression; a button "Up" is clicked |
| orderControlsVerticalAlign | Customizes vertically position of order controls relatively to lists |
| orderControlsVisible | If "false", 'Up' and 'Down' controls aren't displayed. |
| removeAllControlClass | CSS class for remove all control |
| removeAllControlLabel | Defines a label for a removeAll control |
| removeControlClass | CSS class for remove control |
| removeControlLabel | Defines a label for a remove control |
| rendered | If "false", this component is not rendered |
| required | If "true", this component is checked for non-empty input |
| rowClasses | CSS class for a row |
| rowKey | RowKey is a representation of an identifier for a specific data row |
| rowKeyVar | The attribute provides access to a row key in a Request scope |

| Attribute Name | Description |
|---------------------|---|
| rows | A number of rows to display, or zero for all remaining rows in the table |
| showButtonLabels | Shows a label for a button |
| sourceCaptionLabel | Defines source list caption representation text |
| sourceListWidth | Defines width of a source list |
| sourceSelection | Manages selection in a source list from the server side |
| sourceValue | Define the collection to be shown in source list |
| stateVar | The attribute provides access to a component state on the client side |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| switchByClick | If "true", dragging between lists realized by click |
| targetCaptionLabel | Defines target list caption representation text |
| targetListWidth | Defines width of a target list |
| targetSelection | Manages selection in a target list from the server side |
| targetValue | Define the collection to be shown in target list |
| topControlClass | CSS class for top control |
| topControlLabel | Defines a label for a "Top" control |
| upControlClass | CSS class for up control |
| upControlLabel | Defines a label for an "Up" control |
| validator | MethodBinding pointing at a method that is called during Process Validations phase of the request processing lifecycle, to validate the current value of this component |
| valueChangeListener | Listener for value changes |
| var | Defines a list on the page |

6.54.3. Creating the Component with a Page Tag

Here is a simple example as it could be used on a page:

Example:


```

...
<rich:listShuttle var="item" sourceValue="#{bean.source}"
  targetValue="#{bean.target}" converter="listShuttleconverter">
  <h:column>
    <f:facet name="header">
      <h:outputText value="Cars" />
    </f:facet>
    <h:outputText value="#{item.name}" />
  </h:column>
</rich:listShuttle>
...

```

6.54.4. Creating the Component Dynamically Using Java

Example:

```

import org.richfaces.component.html.listShuttle;
...
HtmllistShuttle mylistShuttle = new HtmllistShuttle();
...

```

6.54.5. Details of Usage

The **<rich:listShuttle>** component consists of the following parts:

- two item lists (source and target). List consists of items. Each item has three different representations: common, selected, active
- optional caption element
- optional ordering controls set is a set of controls that performs reordering
- move controls set is a set of controls, which performs moving items between lists

The *"sourceValue"* attribute defines a List or Array of items to be shown in the source list.

The *"targetValue"* attribute defines a List or Array of items to be shown in the target list.

The *"var"* attribute could be shared between both Lists or Arrays to define lists on the page.

The *"converter"* attribute is used to convert component data to a particular component's value. For example, when you select items in a list, a converter is used to format a set of objects to a strings to be displayed.

Note

It is necessary to override the *"equals"* and *"hashCode"* methods in your own class!

The *"sourceSelection"* attribute stores the collection of items selected by you in the source list. The *"targetSelection"* attribute stores the collection of items selected by you in the target list.

Captions could be added to a list only after it was defined as a *"sourceCaption"* and *"targetCaption"* named facets inside the component or defined with the *"sourceCaptionLabel"* and *"targetCaptionLabel"* attribute.

```
...
<rich:listShuttle var="item" sourceValue="#{bean.source}"
  targetValue="#{bean.target}" sourceSelection="#{bean.sourceSelection}"
  targetSelection="#{bean.targetSelection}"
  converter="listShuttleconverter">
  <f:facet name="sourceCaption">
    <h:outputText value="Cars Store #1" />
  </f:facet>
  <f:facet name="targetCaption">
    <h:outputText value="Cars Store #2" />
  </f:facet>
  <rich:column>
    <h:outputText value="#{items.name}" />
  </rich:column>
</rich:listShuttle>
...
```

The **<rich:listShuttle>** component provides the possibility to use ordering controls set, which performs reordering in the target item list. Every control has possibility to be disabled.

An ordering controls set could be defined with *"topControlLabel"* , *"bottomControlLabel"* , *"upControlLabel"* , *"downControlLabel"* attributes.

It is also possible to use *"topControl"* , *"topControlDisabled"* , *"bottomControl"* , *"bottomControlDisabled"* , *"upControl"* , *"upControlDisabled"* , *"downControl"* , *"downControlDisabled"* facets in order to replace the default controls with facets content.

Example:

```
...
<rich:listShuttle var="item" sourceValue="#{bean.source}"
  targetValue="#{bean.target}" converter="listShuttleconverter">
...
  <f:facet name="topControl">
    <h:outputText value="Move to top" />
  </f:facet>
  <f:facet name="upControl">
    <h:outputText value="Move up" />
  </f:facet>
  <f:facet name="downControl">
    <h:outputText value="Move down" />
  </f:facet>
  <f:facet name="bottomControl">
    <h:outputText value="Move to bottom" />
  </f:facet>
</rich:listShuttle>
...
```

The **<rich:listShuttle>** component also provides 4 predefined controls in move controls set for moving items between source and target lists. Every control has possibility to be disabled.

A move controls set could be defined with `"copyControlLabel"` , `"removeControlLabel"` , `"copyAllControlLabel"` , `"removeAllControlLabel"` attributes.

It is also possible to use `"copyControl"` , `"removeControl"` , `"copyAllControl"` , `"removeAllControl"` facets in order to replace the default controls with facets content.

```
...
<rich:listShuttle var="item" sourceValue="#{bean.source}"
  targetValue="#{bean.target}" converter="listShuttleconverter"
    copyControlLabel="Copy" removeControlLabel="Remove"
    copyAllControlLabel="Copy all" removeAllControlLabel="Remove
all">
  <h:column>
    <f:facet name="header">
      <h:outputText value="Cars" />
    </f:facet>
    <h:outputText value="#{item.name}" />
  </h:column>
</rich:listShuttle>
...
```

Controls rendering is based on the `"controlsType"` attribute. Possible types are button and none.

Note

Currently the button controls type is based on `<div>` element.

Table 6.190. Keyboard usage for elements selection

| Keys and combinations | Description |
|-----------------------|---|
| CTRL+click | Inverts selection for an item |
| SHIFT+click | Selects all rows from active one to a clicked row if they differ, else select the active row. All other selections are cleared |
| CTRL+A | Selects all elements inside the list if some active element is already present in a list |
| Up, Down arrows | Changes the active element to the next or previous in a list and make it the only selected. Scroll follows the selection to keep it visible |

Table 6.191. Keyboard usage for elements reordering

| Keys and combinations | Description |
|-----------------------|--|
| Home | Moves selected set to the top of a list |
| End | Moves selected set to the bottom of a list |
| CTRL+Up arrow | Moves selected item to one position upper |

| Keys and combinations | Description |
|-----------------------|---|
| CTRL+Down arrow | Moves selected item to one position lower |

6.54.6. JavaScript API

Table 6.192. JavaScript API

| Function | Description |
|----------------|---|
| Hide() | Hides ordering control |
| Show() | Shows ordering control |
| isShown() | Checks if current control is shown |
| Enable() | Enables ordering control |
| Disable() | Disables ordering control |
| isEnabled() | Checks if current control is enabled |
| moveUp() | Moves up selected item in the list |
| moveDown() | Moves down selected item in the list |
| moveTop() | Moves top selected item in the list |
| moveBottom() | Moves bottom selected item in the list |
| copy() | Copies selected item from the source list to the target list |
| remove() | Removes selected item from the target list to the source list |
| copyAll() | Copies all items from the source list to the target list |
| removeAll() | Removes all items from the target list to the source list |
| getSelection() | Returns currently selected item |
| getItems() | Returns the collection of all items |

6.54.7. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

There are two ways to redefine the appearance of all `<rich:listShuttle>` components at once:

- Redefine the corresponding skin parameters
- Add to your style sheets style classes used by a `<rich:listShuttle>` component

6.54.8. Skin Parameters Redefinition

Table 6.193. Skin parameters redefinition for items in the source and target lists

| Skin parameters | CSS properties |
|------------------------|------------------|
| generalBackgroundColor | background-color |
| tableBorderColor | border-color |
| tableBorderWidth | border-width |

Table 6.194. Skin parameters redefinition for caption in the source and target lists

| Skin parameters | CSS properties |
|------------------|----------------|
| headerFamilyFont | font-family |
| headerSizeFont | font-size |
| headerWeightFont | font-weight |

Table 6.195. Skin parameters redefinition for a selected rows in the source and target lists

| Skin parameters | CSS properties |
|---------------------------|------------------|
| additionalBackgroundColor | background-color |

Table 6.196. Skin parameters redefinition for a header cell

| Skin parameters | CSS properties |
|-----------------------|---------------------|
| headerBackgroundColor | background-color |
| headerTextColor | color |
| headerFamilyFont | font-family |
| headerSizeFont | font-size |
| tableBorderWidth | border-width |
| subBorderColor | border-top-color |
| panelBorderColor | border-bottom-color |
| panelBorderColor | border-right-color |

Table 6.197. Skin parameters redefinition for a selected cell

| Skin parameters | CSS properties |
|-------------------|----------------|
| generalTextColor | color |
| generalFamilyFont | font-family |
| generalSizeFont | font-size |

Table 6.198. Skin parameters redefinition for an active cell

| Skin parameters | CSS properties |
|-------------------|----------------|
| generalSizeFont | font-size |
| generalFamilyFont | font-family |

Table 6.199. Skin parameters redefinition for controls

| Skin parameters | CSS properties |
|------------------|----------------|
| tableBorderColor | border-color |

Table 6.200. Skin parameters redefinition for a button

| Skin parameters | CSS properties |
|------------------|------------------|
| trimColor | background-color |
| generalTextColor | color |
| headerFamilyFont | font-family |
| headerSizeFont | font-size |

Table 6.201. Skin parameters redefinition for a disabled button

| Skin parameters | CSS properties |
|----------------------|------------------|
| trimColor | background-color |
| tabDisabledTextColor | color |
| headerFamilyFont | font-family |
| headerSizeFont | font-size |

Table 6.202. Skin parameters redefinition for a button highlight

| Skin parameters | CSS properties |
|--------------------|------------------|
| trimColor | background-color |
| selectControlColor | border-color |
| tableBorderWidth | border-width |
| headerFamilyFont | font-family |
| headerSizeFont | font-size |
| generalTextColor | color |

Table 6.203. Skin parameters redefinition for a pressed button

| Skin parameters | CSS properties |
|---------------------------|------------------|
| additionalBackgroundColor | background-color |
| tableBorderColor | border-color |
| tableBorderWidth | border-width |
| headerFamilyFont | font-family |
| headerSizeFont | font-size |
| generalTextColor | color |

Table 6.204. Skin parameters redefinition for a button content

| Skin parameters | CSS properties |
|------------------|----------------|
| headerFamilyFont | font-family |
| headerSizeFont | font-size |

Table 6.205. Skin parameters redefinition for a button selection

| Skin parameters | CSS properties |
|------------------|----------------|
| generalTextColor | color |

6.54.9. Definition of Custom Style Classes

On the screenshot there are classes names that define styles for component elements.

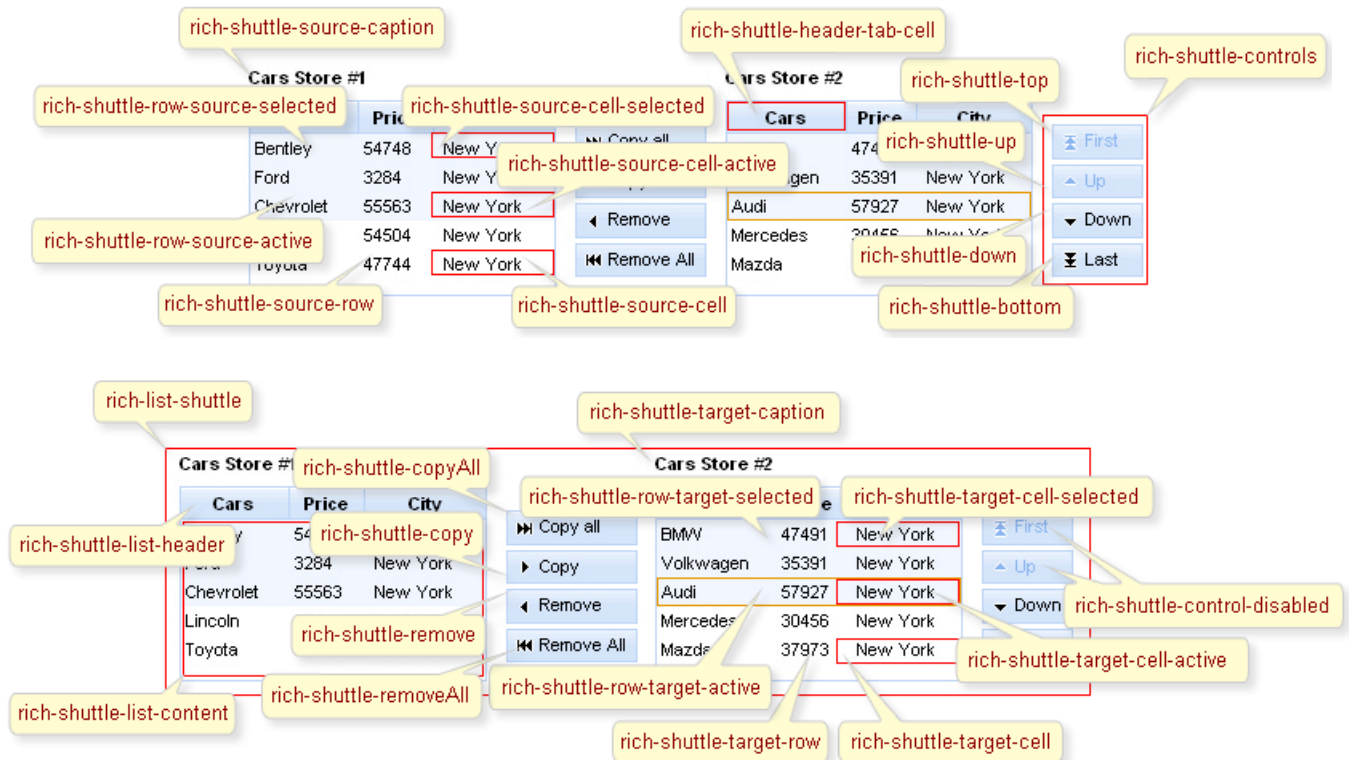


Figure 6.78. Style classes

Table 6.206. Classes names that define a list representation

| Class name | Description |
|------------------------------|---|
| rich-list-shuttle | Defines styles for a wrapper table element of a listShuttle |
| rich-list-shuttle-caption | Defines styles for a list caption |
| rich-shuttle-body | Defines styles for a list body |
| rich-shuttle-list-content | Defines styles for a list content |
| rich-shuttle-source-items | Defines styles for a wrapper <div> element for source list |
| rich-shuttle-target-items | Defines styles for a wrapper <div> element for target list |
| rich-shuttle-list-header | Defines styles for a lists header |
| rich-shuttle-header-tab-cell | Defines styles for a header cell |

Table 6.207. Classes names that define a caption representations in a source and target lists

| Class name | Description |
|-----------------------------|---|
| rich-shuttle-source-caption | Defines styles for a caption in a source list |

| Class name | Description |
|-----------------------------|---|
| rich-shuttle-target-caption | Defines styles for a caption in a target list |

Table 6.208. Classes names that define a rows representations in a source list

| Class name | Description |
|----------------------------------|--|
| rich-shuttle-source-row | Defines styles for a row in a source list |
| rich-shuttle-source-row-selected | Defines styles for a selected row in a source list |
| rich-shuttle-source-row-active | Defines styles for an active row in a source list |

Table 6.209. Classes names that define a rows representations in a target list

| Class name | Description |
|----------------------------------|--|
| rich-shuttle-target-row | Defines styles for a row in a target list |
| rich-shuttle-target-row-selected | Defines styles for a selected row in a target list |
| rich-shuttle-target-row-active | Defines styles for an active row in a target list |

Table 6.210. Classes names that define a cells representations in a source list

| Class name | Description |
|-----------------------------------|---|
| rich-shuttle-source-cell | Defines styles for a cell in a source list |
| rich-shuttle-source-cell-selected | Defines styles for a selected cell in a source list |
| rich-shuttle-source-cell-active | Defines styles for an active cell in a source list |

Table 6.211. Classes names that define a cells representations in a target list

| Class name | Description |
|-----------------------------------|---|
| rich-shuttle-target-cell | Defines styles for a cell in a target list |
| rich-shuttle-target-cell-selected | Defines styles for a selected cell in a target list |
| rich-shuttle-target-cell-active | Defines styles for an active cell in a target list |

Table 6.212. Classes names that define controls representations

| Class name | Description |
|-----------------------|---------------------------------------|
| rich-shuttle-controls | Defines styles for a controls group |
| rich-shuttle-top | Defines styles for a "Top" control |
| rich-shuttle-bottom | Defines styles for a "Bottom" control |
| rich-shuttle-up | Defines styles for a "Up" control |

| Class name | Description |
|-------------------------------|--|
| rich-shuttle-down | Defines styles for a "Down" control |
| rich-shuttle-copy | Defines styles for a "Copy" control |
| rich-shuttle-remove | Defines styles for a "Remove" control |
| rich-shuttle-copyAll | Defines styles for a "copyAll" control |
| rich-shuttle-removeAll | Defines styles for a "removeAll" control |
| rich-shuttle-control-disabled | Defines styles for a control in a disabled state |

Table 6.213. Classes names that define a button representation

| Class name | Description |
|------------------------------------|---------------------------------------|
| rich-list-shuttle-button | Defines styles for a button |
| rich-list-shuttle-button-disabled | Defines styles for a disabled button |
| rich-list-shuttle-button-light | Defines styles for a button highlight |
| rich-list-shuttle-button-press | Defines styles for a pressed button |
| rich-list-shuttle-button-content | Defines styles for a button content |
| rich-list-shuttle-button-selection | Defines styles for a button selection |

In order to redefine styles for all `<rich:listShuttle>` components on a page using CSS, it's enough to create classes with the same names and define necessary properties in them.

To change styles of particular `<rich:listShuttle>` components, define your own style classes in the corresponding `<rich:listShuttle>` component attributes.

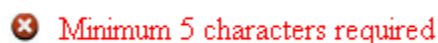
6.54.10. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/listShuttle.jsf?c=listShuttle>] you can see an example of `<rich:listShuttle>` usage and sources for the given example.

6.55. < rich:message >

6.55.1. Description

The component is used for rendering a single message for a specific component.

**Figure 6.79. <rich:message> component**

6.55.2. Key Features

- Highly customizable look and feel
- Tracking both traditional and Ajax based requests
- Optional toolTip to display the detail portion of the message
- Additionally customizable with attributes and facets
- Additionally provides two parts to be optionally defined: marker and label

Table 6.214. rich : message attributes

| Attribute Name | Description |
|------------------|---|
| ajaxRendered | ajaxRendered |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| errorClass | CSS style class to apply to any message with a severity class of "ERROR" |
| errorLabelClass | CSS style class to apply to any message label with a severity class of "ERROR" |
| errorMarkerClass | CSS style class to apply to any message marker with a severity class of "ERROR" |
| fatalClass | CSS style class to apply to any message with a severity class of "FATAL" |
| fatalLabelClass | CSS style class to apply to any message label with a severity class of "FATAL" |
| fatalMarkerClass | CSS style class to apply to any message marker with a severity class of "FATAL" |
| for | Client identifier of the component for which to display messages |
| id | Every component may have a unique id that is automatically created if omitted |
| infoClass | CSS style class to apply to any message with a severity class of "INFO" |
| infoLabelClass | CSS style class to apply to any message label with a severity class of "INFO" |
| infoMarkerClass | CSS style class to apply to any message marker with a severity class of "INFO" |
| keepTransient | keepTransient |

| Attribute Name | Description |
|-----------------|---|
| labelClass | CSS style class to apply to label |
| level | comma-separated list of messages categories which should be displayed |
| markerClass | CSS style class to apply to marker |
| markerStyle | CSS style(s) is/are to be applied to marker when this component is rendered |
| passedLabel | Attribute should define the label to be displayed when no message appears |
| rendered | If "false", this component is not rendered |
| showDetail | Flag indicating whether the summary portion of displayed messages should be included. Default value is "true" |
| showSummary | Flag indicating whether the summary portion of displayed messages should be included. Default value is "false" |
| style | The CSS style for message |
| styleClass | Space-separated list of CSS style class(es) to be applied when this element is rendered. This value must be passed through as the "class" attribute on generated markup |
| title | Advisory title information about markup elements generated for this component |
| tooltip | Flag indicating whether the detail portion of the message should be displayed as a tooltip |
| warnClass | CSS style class to apply to any message with a severity class of "WARN" |
| warnLabelClass | CSS style class to apply to any message label with a severity class of "WARN" |
| warnMarkerClass | CSS style class to apply any message marker with a severity class of "WARN" |

Table 6.215. Component identification parameters

| Name | Value |
|-----------------|--|
| component-type | org.richfaces.component.RichMessage |
| component-class | org.richfaces.component.html.HtmlRichMessage |

| Name | Value |
|------------------|---|
| component-family | org.richfaces.component.RichMessage |
| renderer-type | org.richfaces.renderkit.html.RichMessagesHtmlBaseRender |
| tag-class | org.richfaces.taglib.RichMessageTag |

6.55.3. Creating the Component with a Page Tag

To create the simplest variant of message on a page, use the following syntax:

Example:

```
...
<rich:message for="id"/>
...
```

6.55.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlRichMessage;
...
HtmlRichMessage myMessage = new HtmlRichMessage();
...
```

6.55.5. Details of Usage

The component has the same behavior as standard **<h:message>** component except next two features:

- It's ajaxRendered. It means that the component is reRendered after Ajax request automatically without outputPanel usage
- The component optionally provides "passed" state which will be shown if no message is displayed
- Provides possibility to add some marker to message. By default a marker element isn't shown

A set of facets which can be used for marker defining:

- passedMarker. This facet is provided to allow setting a marker to display if there is no message
- errorMarker. This facet is provided to allow setting a marker to display if there is a message with a severity class of "ERROR"
- fatalMarker. This facet is provided to allow setting a marker to display if there is a message with a severity class of "FATAL"
- infoMarker. This facet is provided to allow setting a marker to display if there is a message with a severity class of "INFO"
- warnMarker. This facet is provided to allow setting a marker to display if there is a message with a severity class of "WARN"

The following example shows different variants for component customization. The attribute 'passedLabel' is used for definition of the label to display when no message appears. But the message component doesn't appear before the form submission even when state is defined as passed (on initial rendering). Boolean attribute "showSummary" defines possibility to display summary portion of displayed messages. The facets "errorMarker" and 'passedMarker' set corresponding images for markers.

Example:

```
...
    <rich:message for="id" passedLabel="No errors" showSummary="true">
      <f:facet name="errorMarker">
        <h:graphicImage url="/image/error.png"/>
      </f:facet>
      <f:facet name="passedMarker">
        <h:graphicImage url="/image/passed.png"/>
      </f:facet>
    </rich:message>
  ...
```

6.55.6. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*.

There are no skin parameters and default predefined values. To redefine the appearance of all `<rich:message>` components at once, you should only add to your style sheets *style classes* used by a `<rich:message>` component.

6.55.7. Definition of Custom Style Classes

On the screenshot there are classes names that define styles for component elements.

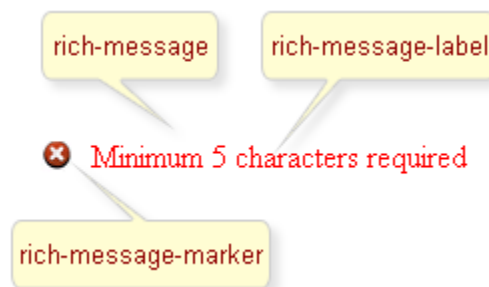


Figure 6.80. Classes names

Table 6.216. Classes names that define a component appearance

| Class name | Description |
|---------------------|--------------------------------------|
| rich-message | Defines styles for a wrapper element |
| rich-message-marker | Defines styles for a marker |
| rich-message-label | Defines styles for a label |

In order to redefine styles for all `<rich:message>` components on a page using CSS, it's enough to create classes with the same names and define necessary properties in them.

To change styles of particular `<rich:message>` components, define your own style classes in the corresponding `<rich:message>` attributes.

6.55.8. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/message.jsf?c=message>] you can see the example of `<rich:message>` usage and sources for the given example.

6.56. < rich:messages >

6.56.1. Description

The `<rich:messages>` component is similar to `<rich:message>` component but used for rendering all messages for the components.

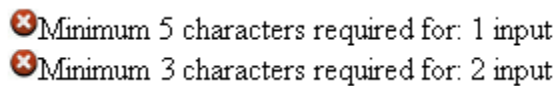


Figure 6.81. <rich:message> component

6.56.2. Key Features

- Highly customizable look and feel
- Track both traditional and Ajax based requests
- Optional ToolTip to display a detailed part of the messages
- Additionally customizable via attributes and facets
- Additionally provides of three parts to be optionally defined: marker, label and header

Table 6.217. rich : messages attributes

| Attribute Name | Description |
|------------------|---|
| ajaxRendered | ajaxRendered |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| errorClass | CSS style class to apply to any message with a severity class of "ERROR" |
| errorLabelClass | CSS style class to apply to any message label with a severity class of "ERROR" |
| errorMarkerClass | CSS style class to apply to any message marker with a severity class of "ERROR" |

| Attribute Name | Description |
|-------------------------------|---|
| <code>fatalClass</code> | CSS style class to apply to any message with a severity class of "FATAL" |
| <code>fatalLabelClass</code> | CSS style class to apply to any message label with a severity class of "FATAL" |
| <code>fatalMarkerClass</code> | CSS style class to apply to any message marker with a severity class of "FATAL" |
| <code>globalOnly</code> | Flag indicating that only global messages (that is, messages not associated with any client identifier) are to be displayed. Default value is "false" |
| <code>id</code> | Every component may have a unique id that is automatically created if omitted |
| <code>infoClass</code> | CSS style class to apply to any message with a severity class of "INFO" |
| <code>infoLabelClass</code> | CSS style class to apply to any message label with a severity class of "INFO" |
| <code>infoMarkerClass</code> | CSS style class to apply to any message marker with a severity class of "INFO" |
| <code>keepTransient</code> | <code>keepTransient</code> |
| <code>labelClass</code> | CSS style class to apply to label |
| <code>layout</code> | The type of layout markup to use when rendering error messages. Valid values are "table" (an HTML table), "list" (an HTML list) and iterator. If not specified, the default value is "list" |
| <code>level</code> | comma-separated list of messages categories which should be displayed |
| <code>markerClass</code> | CSS style class to apply to marker |
| <code>markerStyle</code> | CSS style(s) is/are to be applied to marker when this component is rendered |
| <code>passedLabel</code> | Attribute should define the label to be displayed when no message appears |
| <code>rendered</code> | If "false", this component is not rendered |
| <code>showDetail</code> | Flag indicating whether the summary portion of displayed messages should be included. Default value is "true" |

| Attribute Name | Description |
|-----------------|---|
| showSummary | Flag indicating whether the summary portion of displayed messages should be included. Default value is "false" |
| style | The CSS style for message |
| styleClass | Space-separated list of CSS style class(es) to be applied when this element is rendered. This value must be passed through as the "class" attribute on generated markup |
| title | Advisory title information about markup elements generated for this component |
| tooltip | Flag indicating whether the detail portion of the message should be displayed as a tooltip |
| warnClass | CSS style class to apply to any message with a severity class of "WARN" |
| warnLabelClass | CSS style class to apply to any message label with a severity class of "WARN" |
| warnMarkerClass | CSS style class to apply any message marker with a severity class of "WARN" |

Table 6.218. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | org.richfaces.component.RichMessages |
| component-class | org.richfaces.component.html.HtmlRichMessages |
| component-family | org.richfaces.component.RichMessages |
| renderer-type | org.richfaces.renderkit.html.HtmlRichMessagesRender |
| tag-class | org.richfaces.taglib.RichMessagesTag |

6.56.3. Creating the Component with a Page Tag

To create the simplest variant of message on a page, use the following syntax:

Example:

```
...
<rich:messages/>
...
```

6.56.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlRichMessages;
...
HtmlRichMessages myMessages = new HtmlRichMessages();
...
```

6.56.5. Details of Usage

The component has the same behavior as standard `<h:message>` component except next features:

- It's ajaxRendered. It means that the component is reRendered after Ajax request automatically without outputPanel usage.
- The component optionally provides "passed" state which will be shown if no message to be displayed.
- Provides possibility to add some marker to message. By default, a marker element isn't shown.

The component provides two parts to be optionally defined: marker and informational label before the marker for every message.

Set of facet which can be used for a marker defining:

- passedMarker. This facet is provided to allow setting a marker to be displayed if there is no message.
- errorMarker. This facet is provided to allow setting a marker to be displayed if there is a message with a severity class of "ERROR".
- fatalMarker. This facet is provided to allow setting a marker to be displayed if there is a message with a severity class of "FATAL".
- infoMarker. This facet is provided to allow setting a marker to be displayed if there is a message with a severity class of "INFO".
- warnMarker. This facet is provided to allow setting a marker to be displayed if there is an message with a severity class of "WARN".

The following example shows different variants of customization of the component.

Example:

```
...
<rich:messages layout="table" tooltip="true" showDetail="false"
showSummary="true" passedLabel="No Errors" var="messages">
  <f:facet name="errorMarker">
    <h:graphicImage url="/image/error.png"/>
  </f:facet>
  <f:facet name="infoMarker">
    <h:graphicImage url="/image/info.png"/>
  </f:facet>
</rich:messages>
```

```

    </f:facet>
    <f:facet name="passedMarker">
        <h:graphicImage url="/image/passed.png"/>
    </f:facet>
</rich:messages>
...

```

6.56.6. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*.

There are no skin parameters and default predefined values. To redefine the appearance of all **<rich:messages>** components at once, you should only add to your style sheets *style classes* used by a **<rich:messages>** component.

6.56.7. Definition of Custom Style Classes

On the screenshot there are classes names that define styles for component elements.

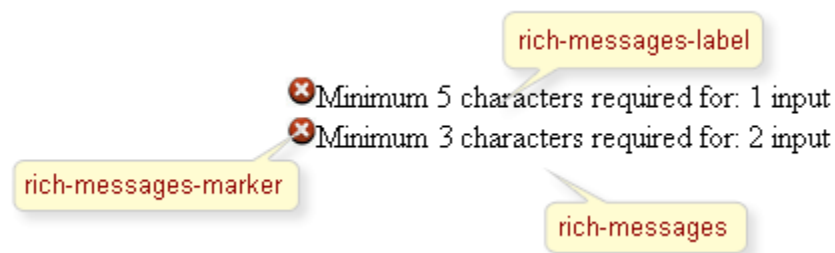


Figure 6.82. Classes names

Table 6.219. Classes names that define a component appearance

| Class name | Description |
|----------------------|--------------------------------------|
| rich-messages | Defines styles for a wrapper element |
| rich-messages-marker | Defines styles for a marker |
| rich-messages-label | Defines styles for a label |

In order to redefine styles for all **<rich:messages>** components on a page using CSS, it's enough to create classes with the same names and define necessary properties in them.

To change styles of particular **<rich:messages>** components, define your own style classes in the corresponding **<rich:messages>** attributes.

6.56.8. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/messsages.jsf?c=messages>] you can see the example of **<rich:messages>** usage and sources for the given example.

6.57. < rich:modalPanel >

6.57.1. Description

The component implements a modal dialog window. All operations in the main application window are locked out while this window is active. Opening and closing the window is done through client JavaScript code.



Figure 6.83. <rich:modalPanel> component

6.57.2. Key Features

- Highly customizable look and feel
- Support of draggable operations and size changes by you
- Easy positioning for the modal dialog window
- Possibility to restore of the previous component state on a page (including position on the screen) after submitting and reloading

Table 6.220. rich : modalPanel attributes

| Attribute Name | Description |
|----------------|---|
| autosized | If 'true' modalPanel should be autosizeable |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| controlsClass | CSS style(s) is/are to be applied to component controls when this component is rendered |

| Attribute Name | Description |
|-------------------|--|
| converter | Id of Converter to be used or reference to a Converter |
| converterMessage | A ValueExpression enabled attribute that, if present, will be used as the text of the converter message, replacing any message that comes from the converter |
| headerClass | CSS style(s) is/are to be applied to component header when this component is rendered |
| height | Attribute defines height of component |
| id | Every component may have a unique id that is automatically created if omitted |
| keepVisualState | If "true" modalPanel should save state after submission |
| label | A localized user presentable name for this component. |
| left | Attribute defines X position of component left-top corner |
| minHeight | Attribute defines min height of component |
| minWidth | Attribute defines min width of component |
| moveable | if "true" there is possibility to move component |
| onhide | Event must occurs after panel closed |
| onmaskclick | <code>\${prop.xmlEncodedDescription}</code> |
| onmaskcontextmenu | <code>\${prop.xmlEncodedDescription}</code> |
| onmaskdblclick | <code>\${prop.xmlEncodedDescription}</code> |
| onmaskmousedown | <code>\${prop.xmlEncodedDescription}</code> |
| onmaskmousemove | <code>\${prop.xmlEncodedDescription}</code> |
| onmaskmouseout | <code>\${prop.xmlEncodedDescription}</code> |
| onmaskmouseover | <code>\${prop.xmlEncodedDescription}</code> |
| onmaskmouseup | <code>\${prop.xmlEncodedDescription}</code> |
| onshow | Event must occurs after panel opened |
| rendered | If "false", this component is not rendered |
| resizeable | if "true" there is possibility to change component size |
| shadowDepth | Pop-up shadow depth for suggestion content |
| shadowOpacity | HTML CSS class attribute of element for pop-up suggestion content |

| Attribute Name | Description |
|-------------------------------|--|
| showWhenRendered | If "true" value for this attribute makes a modal panel opened as default. |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| top | Attribute defines Y position of component left-top corner |
| tridentIVEngineSelectBehavior | How to handle HTML SELECT-based controls in IE 6? - "disable" - default, handle as usual, use disabled="true" to hide SELECT controls - "hide" - use visibility="hidden" to hide SELECT controls |
| validator | MethodBinding pointing at a method that is called during Process Validations phase of the request processing lifecycle, to validate the current value of this component |
| validatorMessage | A ValueExpression enabled attribute that, if present, will be used as the text of the validator message, replacing any message that comes from the validator |
| value | The initial value to set when rendered for the first time |
| valueChangeListener | Listener for value changes |
| visualOptions | Defines options that were specified on the client side |
| width | Attribute defines width of component |
| zindex | Attribute is similar to the standard HTML attribute and can specify window placement relative to the content |

Table 6.221. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | org.richfaces.ModalPanel |
| component-class | org.richfaces.component.html.HtmlModalPanel |
| component-family | org.richfaces.ModalPanel |
| renderer-type | org.richfaces.ModalPanelRenderer |
| tag-class | org.richfaces.taglib.ModalPanelTag |

6.57.3. Creating the Component with a Page Tag

Here is a simple example as it could be used on a page:

Example:

```
...
<rich:modalPanel id="panel">
  <f:facet name="header">
    <h:outputText value="header">
  </f:facet>
  ...
  <!--Any Content inside-->
  ...
  <a href="javascript:RichFaces.hideModalPanel('form:panel')">Hide</a>
</rich:modalPanel>
...
<a href="javascript:RichFaces.showModalPanel('form:panel')">Show</a>
...
```

6.57.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlModalPanel;
...
HtmlModalPanel myPanel = new HtmlModalPanel();
...
```

6.57.5. Details of Usage

The component is defined as a panel with some content inside that displays its content as a modal dialog. To call it and to close it, the client API for the window is used.

Table 6.222. Functions description

| Function | Description |
|--------------------------------------|--|
| RichFaces.showModalPanel (client Id) | Opens a window with a specified client Id |
| RichFaces.hideModalPanel (client Id) | Closes a window with a specified client Id |

New:

In order to avoid a bug in IE, the root node of the dialog is moved on the top of a DOM tree. However, you should have a separate form inside the modal panel if you want to perform submits from this panel.

It's possible to add a *"header"* facet to the component to set the content for the header.

Example:

```

...
<form jsfc="h:form" id="form">
  <rich:modalPanel id="panel" width="400" height="300">
    <f:facet name="header">
      <h:outputText value="Modal Panel"/>
    </f:facet>
    <h:graphicImage value="/pages/california_large.png"/>
    <a href="javascript:Richfaces.hideModalPanel('form:panel')">Close</a>
  </rich:modalPanel>
  <a href="javascript:Richfaces.showModalPanel('form:panel');">Open</a>
</form>
...

```

This defines a window with a particular size and ID. It includes one "Open" link. Clicking on this link makes the modal window content appear.



Figure 6.84. `<rich:modalPanel>` with links

A facet named *"controls"* can be added to the component to place control elements on a header.

Example:

```

...
<rich:modalPanel id="mp">
  <f:facet name="header"><h:outputText value="Modal Panel"/></f:facet>
  <f:facet name="controls">
    <h:graphicImage value="/pages/close.png" style="cursor:pointer"
onclick="Richfaces.hideModalPanel('mp')"/>
  </f:facet>
  <h:graphicImage value="/pages/california_large.png"/>
</rich:modalPanel>

```

The result is displayed here:



Figure 6.85. <rich:modalPanel> with control element

To manage the placement of inserted windows, use the *"zindex"* attribute that is similar to the standard HTML attribute and can specify window placement relative to the content.

To manage window placement relative to the component, there are *"left"* and *"top"* attributes defining a window shifting relative to the top-left corner of the window.

Modal windows can also support resize and move operations on the client side. To allow or disallow these operations, set the *"resizeable"* and *"moveable"* attributes to *"true"* or *"false"* values. Window resizing is also limited by *"minWidth"* and *"minHeight"* attributes specifying the minimal window sizes.

You can pass your parameters during modalPanel opening or closing. This passing could be performed in the following way:

Example:

```
Richfaces.showModalPanel('panelId', {left: auto}, {param1: value1});
```

Thus, except the standard modalPanel parameters you can pass any of your own parameters.

Also modalPanel allows to handle its own opening and closing events on the client side. The *"onshow"* and *"onclose"* attributes are used in this case.

The following example shows how on the client side to define opening and closing event handling in such a way that your own parameters could also be obtained:

Example:

```
onshow="alert(event.parameters.param1)"
```

Here, during modalPanel opening the value of a passing parameter is output.

More information about this problem could be found on the RichFaces Development Forum [<http://www.jboss.com/index.html?module=bb&op=viewtopic&t=111804>].

There is a possibility to restore of the previous component state on a page (including position on the screen) after submitting and reloading. The modalPanel has some special attributes like *"showWhenRendered"* and *"keepVisualState"*.

"showWhenRendered" - This boolean attribute is used if modalPanel should be rendered after first page loading.

"keepVisualState" - Used if modalPanel should save state after submission. If *"keepVisualState" = true* then parameters which modalPanel has during opening should be submitted and passed to new page.

Example:

```
...
<a href="javascript:Richfaces.showModalPanel('_panel', {top:'10px', left:'10px',
  height:'400'});">Show</a>
...
```

Here, if you open modal dialog window using current link and after submits data then modalPanel destination and height on new loaded page will be restored.

if you need the content of the modalPanel to be submitted - you need to remember two important rules:

- modalPanel must have its own form if it has form elements (input or/and command components) inside (as it was shown in the example above)
- modalPanel must not be included into the form (on any level up) if it has the form inside.

Simple example of using commandButton within modalPanel is placed below.

Example:

```
...
    <rich:modalPanel>
        <f:facet name="header">
            <h:outputText value="Test" />
        </f:facet>
        <f:facet name="controls">
            <h:commandLink value="Close" style="cursor:pointer"
onclick="Richfaces.hideModalPanel('mp')" />
        </f:facet>
        <h:form>
            <t:commandButton value="Test" action="#{TESTCONTROLLER.test}"
/>
        </h:form>
    </rich:modalPanel>
    ...
    <h:form>
        <!--Some other Page content-->
    </h:form>
...
```

See also discussion about this problem on the RichFaces Users Forum [<http://www.jboss.com/index.html?module=bb&op=viewtopic&p=4064191>].

6.57.6. JavaScript API

Table 6.223. JavaScript API

| Function | Description |
|----------|---------------------------------|
| Show() | Shows the corresponding ToolTip |
| Hide() | Hides the corresponding ToolTip |

6.57.7. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

There are two ways to redefine the appearance of all `<rich:modalPanel>` components at once:

- Redefine the corresponding skin parameters
- Add to your style sheets *style classes* used by a `<rich:modalPanel>` component

6.57.8. Skin Parameters Redefinition

Table 6.224. Skin parameters for a component

| Skin parameters | CSS properties |
|------------------------|------------------|
| generalBackgroundColor | background-color |
| panelBorderColor | border-color |

Table 6.225. Skin parameters redefinition for a header element

| Skin parameters | CSS properties |
|-----------------------|------------------|
| headerBackgroundColor | background-color |
| headerBackgroundColor | border-color |

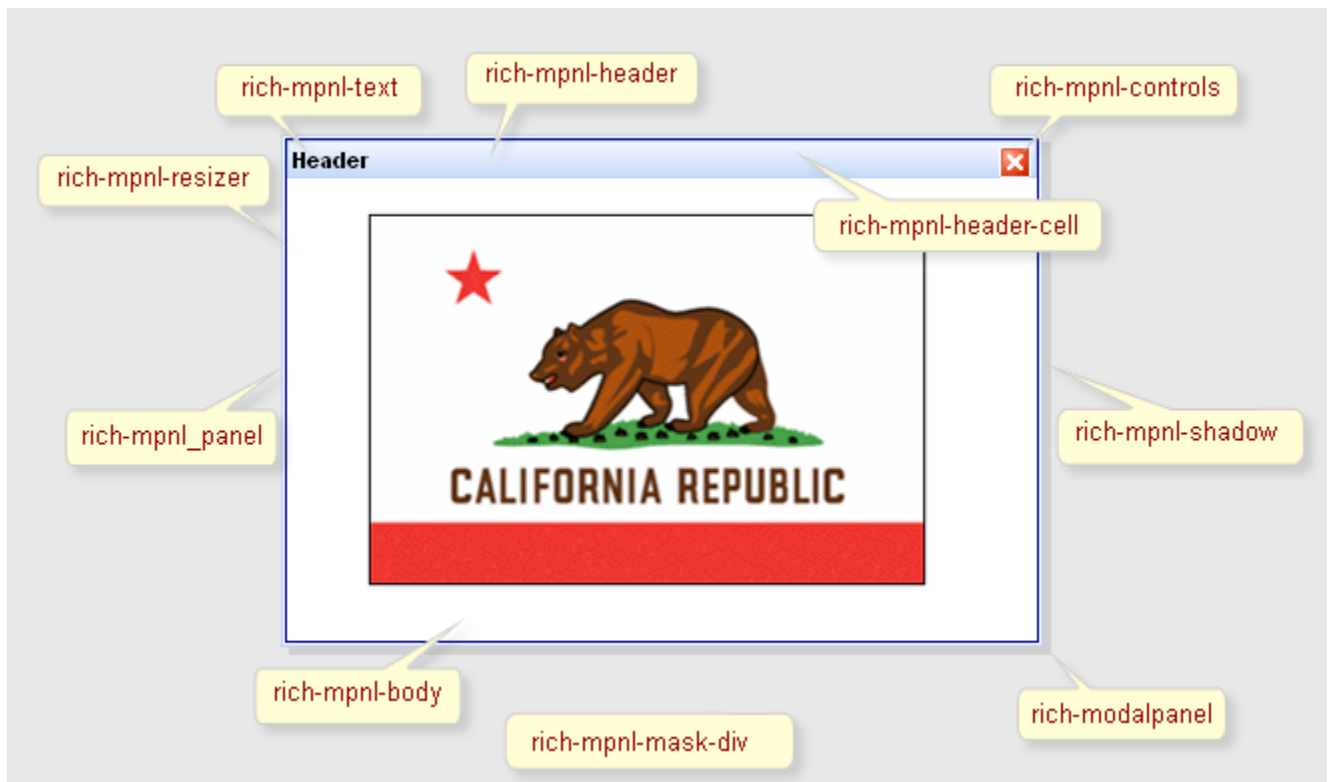
Table 6.226. Skin parameters redefinition for a header content

| Skin parameters | CSS properties |
|------------------|------------------|
| headerSizeFont | background-color |
| headerTextColor | font-size |
| headerWeightFont | color |
| headerFamilyFont | font-family |

Table 6.227. Skin parameters redefinition for a body element

| Skin parameters | CSS properties |
|-------------------|----------------|
| generalSizeFont | font-size |
| generalTextColor | color |
| generalFamilyFont | font-family |

6.57.9. Definition of Custom Style Classes

**Figure 6.86. <rich:modalPanel> class name**

The screenshot shows the classes names for defining different elements.

Table 6.228. Classes names that define a component appearance

| Class name | Description |
|--------------------|--|
| rich-modalpanel | Defines styles for a wrapper <div> element of a modalpanel |
| rich-mpnl_panel | Defines styles for a modalpanel |
| rich-mpnl-mask-div | Defines styles for a wrapper <div> element of a mask |
| rich-mpnl-resizer | Defines styles for a wrapper <div> element of a resizing element |

| Class name | Description |
|-----------------------|--|
| rich-mpnl-shadow | Defines styles for a modalpanel shadow |
| rich-mpnl-header | Defines styles for a modalpanel header |
| rich-mpnl-header-cell | Defines styles for a header cell |
| rich-mpnl-text | Defines styles for a wrapper <div> element of a header text |
| rich-mpnl-body | Defines styles for a content inside a modalpanel |
| rich-mpnl-controls | Defines styles for a wrapper <div> element of a modalpanel control |

In order to redefine styles for all **<rich:modalPanel>** components on a page using CSS, it's enough to create classes with the same names and define necessary properties in them.

To change styles of particular **<rich:modalPanel>** components, define your own style classes in the corresponding **<rich:modalPanel>** attributes.

6.57.10. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/modalPanel.jsf?c=modalPanel>] you can see the example of **<rich:modalPanel>** usage and sources for the given example.

Information about wizards using the **<rich:modalPanel>** component could be found in the Wiki article [<http://labs.jboss.com/wiki/ModalPanelWizards>] and in the FAQ chapter of the guide.

Examples of validation in **<rich:modalPanel>** could be found in the Wiki article [<http://labs.jboss.com/wiki/ModalPanelValidation>] and on the RichFaces Users Forum [<http://www.jboss.com/index.html?module=bb&op=viewtopic&p=4061517>].

6.58. < rich:orderingList >

6.58.1. Description

The **<rich:orderingList>** is a component for ordering items in a list. This component provides possibilities to reorder a list and sort it on the client side.



Figure 6.87. **<rich:orderingList>** component

6.58.2. Key Features

- Highly customizable look and feel
- Reordering possibility for list items
- Multiple selection of list items
- Keyboard support

Table 6.229. rich : orderingList attributes

| Attribute Name | Description |
|--------------------------|--|
| activeItem | Stores active item |
| ajaxKeys | Defines strings that are updated after an Ajax request |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| bottomControlLabel | Defines a label for a 'Bottom' control |
| captionLabel | Defines caption representation text |
| columnClasses | CSS class for a column |
| componentState | It defines EL-binding for a component state for saving or redefinition |
| controlsHorizontalAlign | Controls horizontal rendering. Possible values: left - controls should be rendered to the left side of a list. right(Default)- controls should be rendered to the right side of a list. |
| controlsType | Defines type of a control: button or none. |
| controlsVerticalAlign | Controls vertical rendering. Possible values: top - controls should be rendered aligned to top side of a list. bottom - controls should be rendered aligned to bottom side of a list. center(Default) - controls should be rendered centered relatively to a list. |
| converter | Id of Converter to be used or reference to a Converter |
| downControlLabel | Defines a label for a 'Down' control |
| fastOrderControlsVisible | If "false", 'Top' and 'Bottom' controls aren't displayed. |
| id | Every component may have a unique id that is automatically created if omitted |
| immediate | A flag indicating that this component value must be converted and validated immediately (that is, during |

| Attribute Name | Description |
|----------------------|--|
| | Apply Request Values phase), rather than waiting until a Process Validations phase |
| listHeight | Defines height of a list |
| listWidth | Defines width of a list |
| onbottomclick | A JavaScript event handler; a button "Bottom" is clicked |
| onclick | HTML: a script expression; a pointer button is clicked |
| ondblclick | HTML: a script expression; a pointer button is double-clicked |
| onmousedown | A JavaScript event handler; a button "Down" is clicked |
| onheaderclick | A JavaScript event handler; a header is clicked |
| onmousemove | HTML: a script expression; a pointer is moved within |
| onmouseout | HTML: a script expression; a pointer is moved away |
| onmouseover | HTML: a script expression; a pointer is moved onto |
| onorderchanged | A JavaScript event handler called on an order operation |
| ontopclick | A JavaScript event handler; a button "Top" is clicked |
| onupclick | HTML: a script expression; a button "Up" is clicked |
| orderControlsVisible | If "false", 'Up' and 'Down' controls aren't displayed. |
| rendered | If "false", this component is not rendered |
| required | If "true", this component is checked for non-empty input |
| rowClasses | CSS class for a row |
| rowKey | RowKey is a representation of an identifier for a specific data row |
| rowKeyVar | The attribute provides access to a row key in a Request scope |
| rows | A number of rows to display, or zero for all remaining rows in the list |
| selection | Collection which stores a set of selected items |
| showButtonLabels | If "true", shows a label for a button |
| style | CSS style(s) is/are to be applied when this component is rendered |

| Attribute Name | Description |
|---------------------|---|
| styleClass | Corresponds to the HTML class attribute |
| topControlLabel | Defines a label for a 'Top' control |
| upControlLabel | Defines a label for a 'Up' control |
| validator | MethodBinding pointing at a method that is called during Process Validations phase of the request processing lifecycle, to validate the current value of this component |
| value | Define the collection to be shown in a list |
| valueChangeListener | Listener for value changes |
| var | Defines a list on the page |

Table 6.230. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | org.richfaces.OrderingList |
| component-class | org.richfaces.component.html.HtmlOrderingList |
| component-family | org.richfaces.OrderingList |
| renderer-type | org.richfaces.OrderingListRenderer |
| tag-class | org.richfaces.taglib.OrderingListTagHandler |

6.58.3. Creating the Component with a Page Tag

Here is a simple example as it could be used on a page:

Example:

```
...
<rich:orderingList value="#{bean.list}" var="list">
    <rich:column>
        <f:facet name="header">
            <h:outputText value="Name" />
        </f:facet>
        <h:inputText value="#{list.name}" />
    </rich:column>
</rich:orderingList>
...
```

6.58.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlOrderingList;
```



```
...
HtmlOrderingList myOrderingList = new HtmlOrderingList();
...
```

6.58.5. Details of Usage

The `<rich:orderingList>` component consists of

- Item list element that displays a list of items. It has three different representations for a single element: common, selected, active. Combination of these states is possible.
- Ordering controls set

The `"value"` and `"var"` attributes are used to access the values of a list.

Controls rendering is based on the `"controlsType"` attribute. Possible types are button or none.

Note

Currently the button controls type is based on `<div>` element.

The `"selection"` attribute stores the collection of items selected by you. In the example below after submitting the form the current collection is placed in the object's property and then `<rich:dataTable>` with selected items will be shown.

Example:

```
...
<h:form>
    <rich:orderingList value="#{bean.simpleItems}" var="item"
selection="#{bean.selection}" controlsType="button">
        <rich:column>
            <f:facet name="header">
                <h:outputText value="Cars" />
            </f:facet>
            <h:outputText value="#{item}" />
        </rich:column>
    </rich:orderingList>
    <rich:dataTable id="infoPanelID" value="#{bean.info}" var="info"
rendered="true">
        <rich:column>
            <h:outputText value="#{info}" />
        </rich:column>
    </rich:dataTable>
    <a4j:commandButton value="reRender" reRender="infoPanelID" />
</h:form>
...
```

The `<rich:orderingList>` component allows to use `"caption"` facet. A caption could be also defined with `"captionLabel"` attribute.

Simple example is placed below.

Example:

```

...
<rich:orderingList value="#{bean.simpleItems}" var="item" controlsType="button"
  selection="#{bean.selection}">
  <f:facet name="caption">
    <h:outputText value="Caption Facet" />
  </f:facet>
  <rich:column>
    <f:facet name="header">
      <h:outputText value="Cars" />
    </f:facet>
    <h:outputText value="#{item.name}" />
  </rich:column>
  <rich:column>
    <f:facet name="header">
      <h:outputText value="Price" />
    </f:facet>
    <h:outputText value="#{item.price}" />
  </rich:column>
</rich:orderingList>
...

```

The **<rich:orderingList>** component provides the possibility to use ordering controls set, which performs reordering. Every control has possibility to be disabled.

An ordering controls set could be defined with *"topControlLabel"* , *"bottomControlLabel"* , *"upControlLabel"* , *"downControlLabel"* attributes.

It is also possible to use *"topControl"* , *"topControlDisabled"* , *"bottomControl"* , *"bottomControlDisabled"* , *"upControl"* , *"upControlDisabled"* , *"downControl"* , *"downControlDisabled"* facets in order to replace the default controls with facets content.

Example:

```

...
<rich:orderingList value="#{bean.simpleItems}" var="item" controlsType="button"
  selection="#{bean.selection}">
  <f:facet name="topControl">
    <h:outputText value="Move to top" />
  </f:facet>
  <f:facet name="upControl">
    <h:outputText value="Move up" />
  </f:facet>
  <f:facet name="downControl">
    <h:outputText value="Move down" />
  </f:facet>
  <f:facet name="bottomControl">
    <h:outputText value="Move to bottom" />
  </f:facet>
</rich:orderingList>
...

```

The position of the controls relatively to a list could be customized with:

- *"controlsHorizontalAlign"* attribute. Possible values:
 - left - controls render to the left side of a list
 - right(default) - controls render to the right side of a list
 - center - controls is centered
- *"controlsVerticalAlign"* attribute. Possible values:
 - top - controls render aligned to the top side of a list
 - bottom - controls render aligned to the bottom side of a list
 - center(default) - controls is centered relatively to a list

The **<rich:orderingList>** component has a possibility to hide any of the controls by pairs using following attributes:

- *"orderControlsVisible"* attribute has two values: true or false. If false Up and Down controls are not displayed.
- *"fastOrderControlsVisible"* attribute has two values: true or false. If false Top and Bottom controls are not displayed.

Table 6.231. Keyboard usage for elements selection

| Keys and combinations | Description |
|-----------------------|--|
| CTRL+click | Inverts selection for an item |
| SHIFT+click | Selects all rows from active one to a clicked row if they differ, else select the active row. All other selections are cleared |
| CTRL+A | Selects all elements inside the list if some active element is already present in a list |
| Up, Down arrows | Changes the active and selected elements to the next or previous in a list |

Table 6.232. Keyboard usage for elements reordering

| Keys and combinations | Description |
|-----------------------|--|
| Page Up | Moves selected set to the top of a list |
| Page Down | Moves selected set to the bottom of a list |
| CTRL+Up arrow | Moves selected item to one position upper |
| CTRL+Down arrow | Moves selected item to one position lower |

6.58.6. JavaScript API

Table 6.233. JavaScript API

| Function | Description |
|----------------|--|
| Hide() | Hides ordering control |
| Show() | Shows ordering control |
| isShown() | Checks if current control is shown |
| Enable() | Enables ordering control |
| Disable() | Disables ordering control |
| isEnabled() | Checks if current control is enabled |
| moveUp() | Moves up selected item in the list |
| moveDown() | Moves down selected item in the list |
| moveTop() | Moves top selected item in the list |
| moveBottom() | Moves bottom selected item in the list |
| getSelection() | Returns currently selected item |
| getItems() | Returns the collection of all items |

6.58.7. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

There are two ways to redefine the appearance of all `<rich:orderingList>` components at once:

- Redefine the corresponding skin parameters
- Add to your style sheets style classes used by a `<rich:orderingList>` component

6.58.8. Skin Parameters Redefinition

Table 6.234. Skin parameters redefinition for a wrapper `<div>` element of a list

| Skin parameters | CSS properties |
|----------------------|------------------|
| tableBackgroundColor | background-color |
| tableBorderColor | border-color |

Table 6.235. Skin parameters redefinition for a header cell of a list

| Skin parameters | CSS properties |
|-----------------|------------------|
| trimColor | background-color |

| Skin parameters | CSS properties |
|------------------|---------------------|
| generalTextColor | color |
| headerFamilyFont | font-family |
| headerSizeFont | font-size |
| tableBorderWidth | border-right-width |
| tableBorderWidth | border-bottom-width |
| tableBorderColor | border-right-color |
| tableBorderColor | border-bottom-color |

Table 6.236. Skin parameters redefinition for caption element

| Skin parameters | CSS properties |
|------------------|----------------|
| headerFamilyFont | font-family |
| headerSizeFont | font-size |
| headerWeightFont | font-weight |

Table 6.237. Skin parameters redefinition for row element

| Skin parameters | CSS properties |
|---------------------|------------------|
| headerGradientColor | background-color |

Table 6.238. Skin parameters redefinition for selected row element

| Skin parameters | CSS properties |
|---------------------------|------------------|
| additionalBackgroundColor | background-color |

Table 6.239. Skin parameters redefinition for cell element

| Skin parameters | CSS properties |
|-------------------|----------------|
| generalTextColor | color |
| generalFamilyFont | font-family |
| generalSizeFont | font-size |

Table 6.240. Skin parameters redefinition for selected cell element

| Skin parameters | CSS properties |
|-------------------|----------------|
| generalTextColor | color |
| generalFamilyFont | font-family |

| Skin parameters | CSS properties |
|-----------------|----------------|
| generalSizeFont | font-size |

Table 6.241. Skin parameters redefinition for active cell element

| Skin parameters | CSS properties |
|-------------------|----------------|
| generalFamilyFont | font-family |
| generalSizeFont | font-size |

Table 6.242. Skin parameters redefinition for a button

| Skin parameters | CSS properties |
|------------------|------------------|
| trimColor | background-color |
| generalTextColor | color |
| headerFamilyFont | font-family |
| headerSizeFont | font-size |

Table 6.243. Skin parameters redefinition for a disabled button

| Skin parameters | CSS properties |
|----------------------|------------------|
| trimColor | background-color |
| tabDisabledTextColor | color |
| headerFamilyFont | font-family |
| headerSizeFont | font-size |

Table 6.244. Skin parameters redefinition for a button highlight

| Skin parameters | CSS properties |
|--------------------|------------------|
| trimColor | background-color |
| selectControlColor | border-color |
| tableBorderWidth | border-width |
| headerFamilyFont | font-family |
| headerSizeFont | font-size |
| generalTextColor | color |

Table 6.245. Skin parameters redefinition for a pressed button

| Skin parameters | CSS properties |
|---------------------------|------------------|
| additionalBackgroundColor | background-color |
| tableBorderColor | border-color |
| tableBorderWidth | border-width |
| headerFamilyFont | font-family |
| headerSizeFont | font-size |
| generalTextColor | color |

Table 6.246. Skin parameters redefinition for a button content

| Skin parameters | CSS properties |
|------------------|----------------|
| headerFamilyFont | font-family |
| headerSizeFont | font-size |

Table 6.247. Skin parameters redefinition for a button selection

| Skin parameters | CSS properties |
|------------------|----------------|
| generalTextColor | color |

Table 6.248. Skin parameters redefinition for top, bottom, up, down controls and for controls in disabled state

| Skin parameters | CSS properties |
|------------------|----------------|
| panelBorderColor | border-color |

6.58.9. Definition of Custom Style Classes

On the screenshot there are classes names that define styles for component elements.

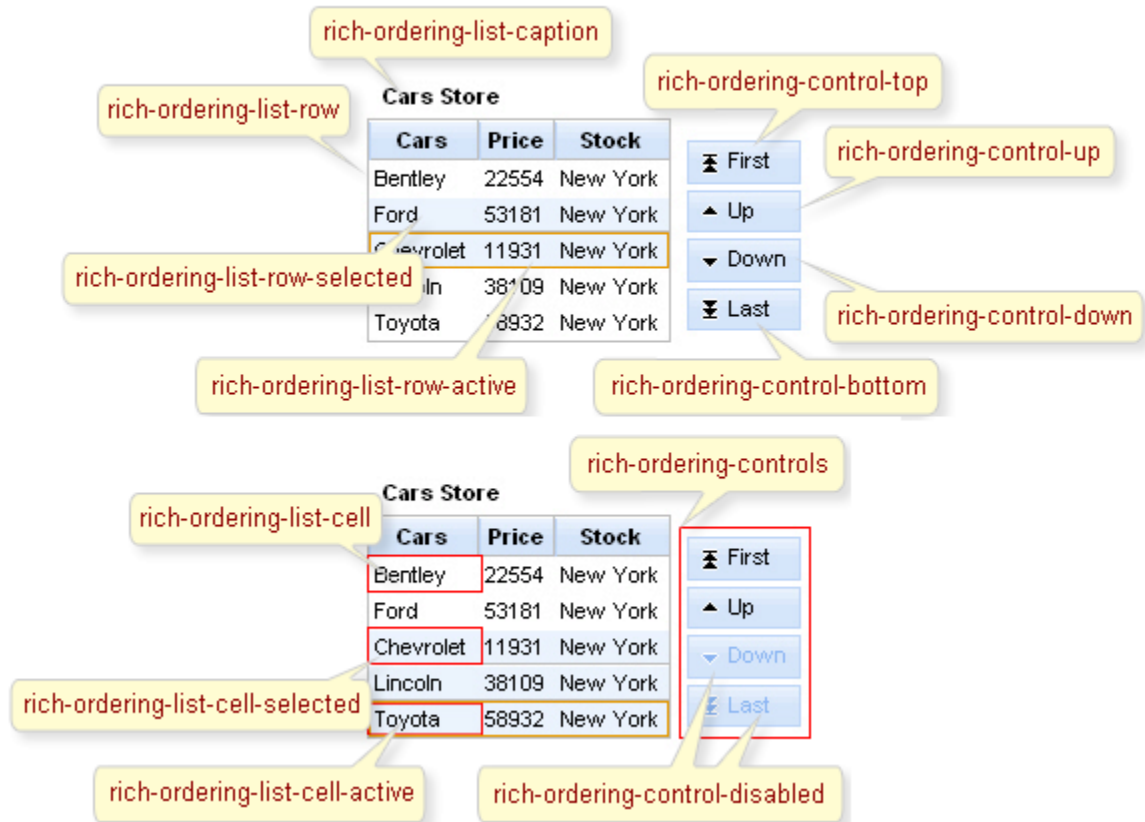


Figure 6.88. Classes names

Table 6.249. Classes names that define a list representation

| Class name | Description |
|--------------------------------------|---|
| rich-ordering-list-body | Defines styles for a wrapper table element of an orderingList |
| rich-ordering-list-output | Defines styles for a wrapper <div> element of a list |
| rich-ordering-list-items | Defines styles for a wrapper table element of items in the list |
| rich-ordering-list-content | Defines styles for a list content |
| rich-ordering-list-header | Defines styles for a wrapper <div> element for a list header |
| rich-ordering-list-table-header | Defines styles for a wrapper <tr> element for a list header |
| rich-ordering-list-table-header-cell | Defines styles for a header cell |

Table 6.250. Classes names that define a caption representation

| Class name | Description |
|-------------------------------------|--|
| rich-ordering-list-caption | Defines styles for a caption |
| rich-ordering-list-caption-disabled | Defines styles for a caption in disabled state |
| rich-ordering-list-caption-active | Defines styles for a caption in active state |

Table 6.251. Classes names that define rows representation

| Class name | Description |
|---------------------------------|-----------------------------------|
| rich-ordering-list-row | Defines styles for a row |
| rich-ordering-list-row-selected | Defines styles for a selected row |
| rich-ordering-list-row-active | Defines styles for an active row |
| rich-ordering-list-row-disabled | Defines styles for a disabled row |

Table 6.252. Classes names that define cells representation

| Class name | Description |
|----------------------------------|------------------------------------|
| rich-ordering-list-cell | Defines styles for a cell |
| rich-ordering-list-cell-selected | Defines styles for a selected cell |
| rich-ordering-list-cell-active | Defines styles for an active cell |
| rich-ordering-list-cell-disabled | Defines styles for a disabled cell |

Table 6.253. Classes names that define a button representation

| Class name | Description |
|-------------------------------------|--|
| rich-ordering-list-button | Defines styles for a button |
| rich-ordering-list-button-disabled | Defines styles for a disabled button |
| rich-ordering-list-button-light | Defines styles for a button highlight |
| rich-ordering-list-button-press | Defines styles for a pressed button |
| rich-ordering-list-button-content | Defines styles for a button content |
| rich-ordering-list-button-selection | Defines styles for a button selection |
| rich-ordering-list-button-valign | Defines styles for a wrapper <td> element for buttons vertical align |
| rich-ordering-list-button-layout | Defines styles for a wrapper <div> element of buttons layout |

Table 6.254. Classes names that define controls representation

| Class name | Description |
|--------------------------------|---|
| rich-ordering-controls | Defines styles for a controls group |
| rich-ordering-control-top | Defines styles for a "top" control |
| rich-ordering-control-bottom | Defines styles for a "bottom" control |
| rich-ordering-control-up | Defines styles for a "up" control |
| rich-ordering-control-down | Defines styles for a "down" control |
| rich-ordering-control-disabled | Defines styles for controls in disabled state |

In order to redefine styles for all `<rich:orderingList>` components on a page using CSS, it's enough to create classes with the same names and define necessary properties in them.

To change styles of particular `<rich:orderingList>` components, define your own style classes in the corresponding `<rich:orderingList>` component attributes.

6.58.10. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/orderingList.jsf?c=orderingList>] you can see an example of `<rich:orderingList>` usage and sources for the given example.

6.59. < rich:paint2D >

6.59.1. Description

Create image by painting from a managed bean method, same as *"paint"* (Graphics2D) in *"SWING"* components.



Figure 6.89. <rich:paint2D> component

6.59.2. Key Features

- Simple Graphics2D - painting style directly on the Web page
- Supports client/server caching for generated images

- Fully supports *"JPEG"* (24-bit, default), *"GIF"* (8-bit with transparency), and *"PNG"* (32-bit with transparency) formats for sending generated images
- Easily customizable borders and white space to wrap the image
- Dynamically settable paint parameters using tag attributes

Table 6.255. rich : paint2D attributes

| Attribute Name | Description |
|----------------|---|
| align | bottom middle top left right Deprecated. This attribute specifies the position of an IMG, OBJECT, or APPLET with respect to its context. The following values for align concern the object's position with respect to surrounding text: * bottom: means that the bottom of the object should be vertically aligned with the current baseline. This is the default value. * middle: means that the center of the object should be vertically aligned with the current baseline. * top: means that the top of the object should be vertically aligned with the top of the current text line |
| bgcolor | Background color of painted image. Default value is 'transparent' which means no background fill. Hex colors can be used, as well as common color names. Invalid values are treated as transparent. Note, that JPEG format doesn't support transparency, and transparent background is painted black. Also note, that several browsers (e.g. IE6) do not support PNG transparency |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| border | Deprecated. This attribute specifies the width of an IMG or OBJECT border, in pixels. The default value for this attribute depends on the user agent |
| cacheable | Supported (or not) client/server caching for generated images. Caching on client supported by properly sending and processing of HTTP headers (Last-Modified, Expires, If-Modified-Since, etc.) Server-side caching is supported by application-scope object cache. For build of cache key use "value" attribute, serialized to URI |
| converter | Id of Converter to be used or reference to a Converter |
| data | |

| Attribute Name | Description |
|----------------|---|
| | Value calculated at render time and stored in Image URI (as part of cache Key), at paint time passed to a paint method. It can be used for updating cache at change of image generating conditions, and for creating paint beans as "Lightweight" pattern components (request scope). IMPORTANT: Since serialized data stored in URI, avoid using big objects |
| format | format Name of format for sending a generated image. It currently supports "jpeg" (24 bit, default), "gif" (8 bit with transparency), "png" (32 bit with transparency) |
| height | Height in pixels of image (for paint canvas and HTML attribute) |
| hspace | Deprecated. This attribute specifies the amount of white space to be inserted to the left and right of an IMG, APPLET, or OBJECT. The default value is not specified, but is generally a small, non-zero length |
| id | Every component may have a unique id that is automatically created if omitted |
| lang | Code describing the language used in the generated markup for this component |
| paint | The method calls expression to paint Image on prepared Buffered image. It must have two parameters with a type of java.awt.Graphics2D (graphics to paint) and Object (restored from URI "data" property). For painting used 32-bit RGBA color model (for 8-bit images used Diffusion filtration before sending) |
| rendered | If "false", this component is not rendered |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| title | Advisory title information about markup elements generated for this component |
| value | The initial value to set when rendered for the first time |
| vspace | Deprecated. This attribute specifies the amount of white space to be inserted above and below an IMG, APPLET, or OBJECT. The default value is not specified, but is generally a small, non-zero length |

| Attribute Name | Description |
|----------------|--|
| width | Width in pixels of image (for paint canvas and HTML attribute) |

Table 6.256. Component identification parameters

| Name | Value |
|------------------|--|
| component-type | org.richfaces.Paint2D |
| component-class | org.richfaces.component.html.HtmlPaint2D |
| component-family | javax.faces.Output |
| renderer-type | org.richfaces.Paint2DRenderer |
| tag-class | org.richfaces.taglib.Paint2DTag |

6.59.3. Creating the Component with a Page Tag

Here is a simple example as it could be used on a page:

Example:

```
...
<rich:paint2D paint="#{paint2D.paint}" data="#{paint2DModel}" />
...
```

Here *"paint"* specifies the method performing drawing and *"data"* specifies Managed Bean property keeping the data used by the method.

6.59.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlPaint2D;
...
HtmlPaint2D myImage = new HtmlPaint2D();
...
```

6.59.5. Details of Usage

The example shows two main attributes of the component:

- *"paint"*

Specify a method receiving an object specified in data as a parameter and sending graphical information into the stream

- *"data"*

Specifies a bean class keeping your data for rendering

Note:

data object should implement serializable interface

The *"format"* attribute of the component defines a format of visual data passing to the server.

Generated data can be used as a cacheable or non-cacheable resource. It's defined with *"cacheable"* attribute. If cache support is turned on, a key is created in URI with a mix of size (width/height), *"paint"* method, *"format"* and *"data"* attributes.

Example:

```

paintBean.java:

    public void paint(Graphics2D g2, Object obj) {
        // code that gets data from the data Bean (PaintData)
        PaintData data = (PaintData) obj;
        ...
        // a code drawing a rectangle
        g2.drawRect(0, 0, data.Width, data.Height);
        ...
        // some more code placing graphical data into g2 stream below
    }

dataBean.java:

    public class PaintData implements Serializable{
        private static final long serialVersionUID = 1L;
        Integer Width=100;
        Integer Height=50;
        ...
    }

page.xhtml:
...
<rich:paint2D paint="#{paint2D.paint}" data="#{paint2DModel.data}"/>
...

```

6.59.6. Look-and-Feel Customization

Paint2D has no skin parameters and special *style classes*, as it consists of one element generated with a your method on the server.

To define some style properties such as an indent or a border, it's possible to use *"style"* and *"styleClass"* attributes on the component.

6.59.7. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/paint2D.jsf?c=paint2d>] you can see the example of **<rich:paint2D>** usage and sources for the given example.

6.60. < rich:panel >

6.60.1. Description

A skinnable panel that is rendered as a bordered rectangle with or without a header.

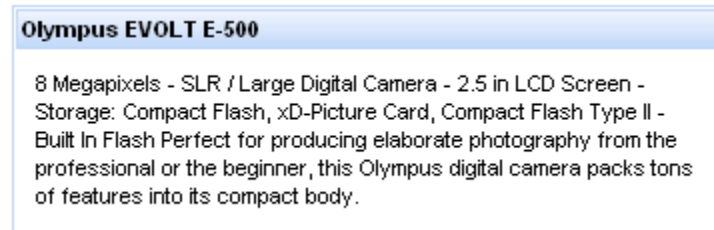


Figure 6.90. <rich:panel> component

6.60.2. Key Features

- Highly customizable look and feel
- Support for any content inside
- Header adding feature

Table 6.257. rich : panel attributes

| Attribute Name | Description |
|----------------|---|
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| bodyClass | A class that defines a style for a panel content |
| header | Label text appears on a panel header |
| headerClass | A class that defines a style for a panel header |
| id | Every component may have a unique id that is automatically created if omitted |
| onclick | HTML: a script expression; a pointer button is clicked |
| ondblclick | HTML: a script expression; a pointer button is double-clicked |
| onkeydown | HTML: a script expression; a key is pressed down |
| onkeypress | HTML: a script expression; a key is pressed and released |
| onkeyup | HTML: a script expression; a key is released |

| Attribute Name | Description |
|----------------|---|
| onmousedown | HTML: script expression; a pointer button is pressed down |
| onmousemove | HTML: a script expression; a pointer is moved within |
| onmouseout | HTML: a script expression; a pointer is moved away |
| onmouseover | HTML: a script expression; a pointer is moved onto |
| onmouseup | HTML: script expression; a pointer button is released |
| rendered | If "false", this component is not rendered |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |

Table 6.258. Component identification parameters

| Name | Value |
|------------------|--|
| component-type | org.richfaces.panel |
| component-class | org.richfaces.component.html.HtmlPanel |
| component-family | org.richfaces.panel |
| renderer-type | org.richfaces.PanelRenderer |
| tag-class | org.richfaces.taglib.PanelTag |

6.60.3. Creating the Component with a Page Tag

Here is a simple example as it could be used on a page:

Example:

```
...
<rich:panel header="Panel Header">
    ...
    <!--Any Content inside-->
    ...
</rich:panel>
...
```

6.60.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlPanel;
...
HtmlPanel myPanel = new HtmlPanel();
```


...

6.60.5. Details of Usage

The *"header"* attribute defines text to be represented. If you can use the *"header"* facet, you can even not use the *"header"* attribute.

Example:

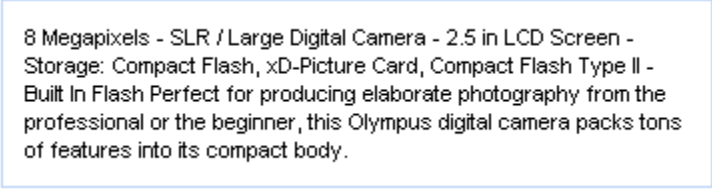
```
...
<rich:panel>
  <f:facet name="header">
    <h:graphicImage value="/images/img1.png"/>
  </f:facet>
  ...
  <!--Any Content inside-->
  ...
</rich:panel>
...
```

<rich:panel> components are used to group page content pieces on similarly formatted rectangular panels.

Example:

```
...
<rich:panel>
  ...
</rich:panel>
...
```

It's generating on a page in the following way:



8 Megapixels - SLR / Large Digital Camera - 2.5 in LCD Screen -
Storage: Compact Flash, xD-Picture Card, Compact Flash Type II -
Built In Flash Perfect for producing elaborate photography from the
professional or the beginner, this Olympus digital camera packs tons
of features into its compact body.

Figure 6.91. Generated panel without header

The example shows that similar rectangular areas are formed with a particular style.

When creating a panel with a header element, one more **<div>** element is added with content defined for a header.

Example:

```
...
<rich:panel>
  <f:facet name="header">
    <h:outputText value="Olympus EVOLT E-500 "/>
  </f:facet>
```

```

...
</rich:panel>
...

```

It's displayed on a page in the following way:

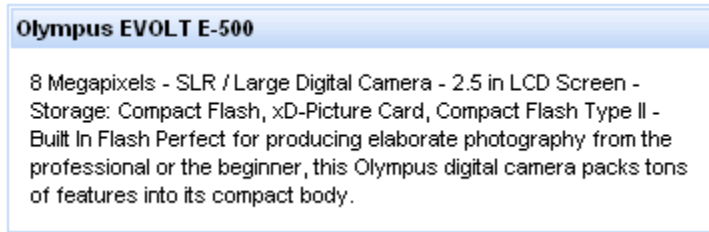


Figure 6.92. <rich:panel> with header

As it has been mentioned above, the component is mostly used for a page style definition, hence the main attributes are style ones.

- `"styleClass"` and `"style"`
- `"headerClass"` and `"headerStyle"`
- `"bodyClass"` and `"bodyStyle"`

Moreover, to add e.g. some JavaScript effects, events defined on it are used.

- `"onmouseover"`
- `"onclick"`
- `"onmouseout"`
- etc.

6.60.7. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

There are two ways to redefine the appearance of all `<rich:panel>` components at once:

- Redefine the corresponding skin parameters
- Add to your style sheets *style classes* used by a `<rich:panel>` component

6.60.8. Skin Parameters Redefinition

Table 6.259. Skin parameters redefinition for a whole component

| Skin parameters | CSS properties |
|-------------------------------------|-------------------------------|
| <code>generalBackgroundColor</code> | <code>background-color</code> |

| Skin parameters | CSS properties |
|------------------|----------------|
| panelBorderColor | border-color |

Table 6.260. Skin parameters redefinition for a header element

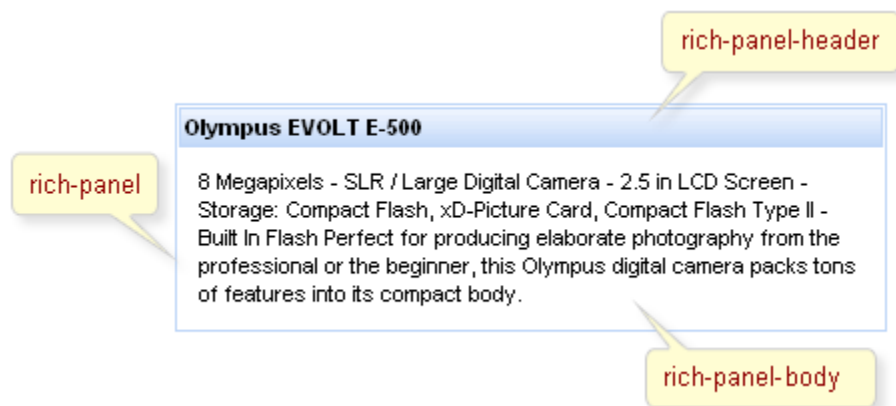
| Skin parameters | CSS properties |
|-----------------------|------------------|
| headerBackgroundColor | background-color |
| headerBackgroundColor | border-color |
| headerSizeFont | font-size |
| headerTextColor | color |
| headerWeightFont | font-weight |
| headerFamilyFont | font-family |

Table 6.261. Skin parameters redefinition for a body element

| Skin parameters | CSS properties |
|-------------------|----------------|
| generalSizeFont | font-size |
| generalTextColor | color |
| generalFamilyFont | font-family |

6.60.9. Definition of Custom Style Classes

On the screenshot there are classes names that define styles for component elements.

**Figure 6.93. Style classes****Table 6.262. Classes names that define a component appearance**

| Class name | Class description |
|------------|--|
| rich-panel | Defines styles for a wrapper <code><div></code> element of a component |

| Class name | Class description |
|-------------------|-------------------------------------|
| rich-panel-header | Defines styles for a header element |
| rich-panel-body | Defines styles for a body elemnt |

In order to redefine styles for all **<rich:panel>** components on a page using CSS, it's enough to create classes with the same names and define necessary properties in them.

To change styles of particular **<rich:panel>** components, define your own style classes in the corresponding **<rich:panel>** attribute.

CSS code piece used on a page:

Example:

```
...
    .rich-panel-header{
        background-color:#F99;
    }
    .myClass{
        font-style:italic;
    }
...
```

Hence, a header class is redefined for all panels (its color changed) of this page and body class is extended with the custom style properties (font-style) for this particular panel. As a result, the panel with a header redefined color and a text style in body is got.

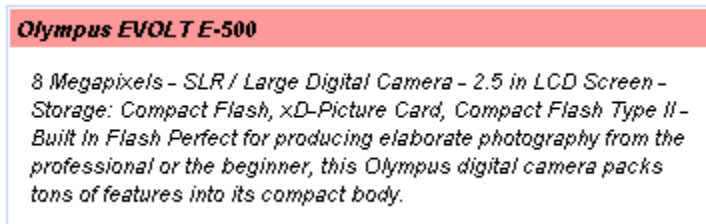


Figure 6.94. <rich:panel> with redefined header and body text style

6.60.10. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/panel.jsf?c=panel>] you can see the example of **<rich:panel>** usage and sources for the given example.

6.61. < rich:panelBar >

6.61.1. Description

panelBar is used for grouping any content which is loaded on the client side and appears as groups divided on child panels after the header is clicked.



Figure 6.95. <rich:panelBar> with content inside

6.61.2. Key Features

- Skinnable slide panel and child items
- Groups any content inside each panel

Table 6.263. rich : panelBar attributes

| Attribute Name | Description |
|-------------------|---|
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| contentClass | The component content style class |
| contentStyle | The component content style |
| converter | Id of Converter to be used or reference to a Converter |
| converterMessage | A ValueExpression enabled attribute that, if present, will be used as the text of the converter message, replacing any message that comes from the converter. |
| headerClass | The component header style class |
| headerClassActive | The component header style class active |
| headerStyle | The component header style |
| headerStyleActive | The component header style active |
| height | The height of the slide panel. Might be defined as pixels or as percentage. The default height is 100% |
| id | Every component may have a unique id that is automatically created if omitted |
| immediate | |

| Attribute Name | Description |
|---------------------|---|
| | A flag indicating that this component value must be converted and validated immediately (that is, during Apply Request Values phase), rather than waiting until a Process Validations phase |
| label | A localized user presentable name for this component. |
| onclick | JavaScript code for call before header onclick |
| rendered | If "false", this component is not rendered |
| required | If "true", this component is checked for non-empty input |
| requiredMessage | A ValueExpression enabled attribute that, if present, will be used as the text of the validation message for the "required" facility, if the "required" facility is used. |
| selectedPanel | Attribure defines name of selected item |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute. |
| validator | MethodBinding pointing at a method that is called during Process Validations phase of the request processing lifecycle, to validate the current value of this component |
| validatorMessage | A ValueExpression enabled attribute that, if present, will be used as the text of the validator message, replacing any message that comes from the validator. |
| value | The initial value to set when rendered for the first time |
| valueChangeListener | Listener for value changes |
| width | The width of the slide panel. Might be defined as pixels or as percentage. The default width is 100% |

Table 6.264. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | org.richfaces.PanelBar |
| component-class | org.richfaces.component.html.HtmlPanelBar |
| component-family | org.richfaces.PanelBar |
| renderer-type | org.richfaces.PanelBarRenderer |

| Name | Value |
|-----------|----------------------------------|
| tag-class | org.richfaces.taglib.PanelBarTag |

6.61.3. Creating the Component with a Page Tag

Here is a simple example as it could be used on a page:

Example:

```
...
<rich:panelBar>
  <!--//... -->
  <rich:panelBarItem label="Canon">
    ...
  </rich:panelBarItem>
  <rich:panelBarItem label="Nikon">
    ...
  </rich:panelBarItem>
</rich:panelBar>
...
```

6.61.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlPanelBar;
...
HtmlPanelBar myBar = new HtmlPanelBar();
...
```

6.61.5. Details of Usage

As it was mentioned above, panelBar is used for grouping any content on the client, thus its customization deals only with specification of sizes and styles for rendering.

"width" and "height" (both are 100% on default) attributes stand apart.

Style attributes are described further.

panelBar could contain any number of child panelBarItem components inside, which content is uploaded onto the client and headers are controls to open the corresponding child element.

6.61.6. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

There are two ways to redefine the appearance of all **<rich:panelBar>** components at once:

- Redefine the corresponding skin parameters

- Add to your style sheets *style classes* used by a `<rich:panelBar>` component

6.61.7. Skin Parameters Redefinition

Table 6.265. Skin parameter redefinition for a whole component

| Skin parameter | CSS properties |
|-----------------------|----------------|
| headerBackgroundColor | border-color |

6.61.8. Definition of Custom Style Classes

There is one predefined class for the `<rich:panelBar>`, which is applicable to a whole component, specifying padding, borders, and etc.

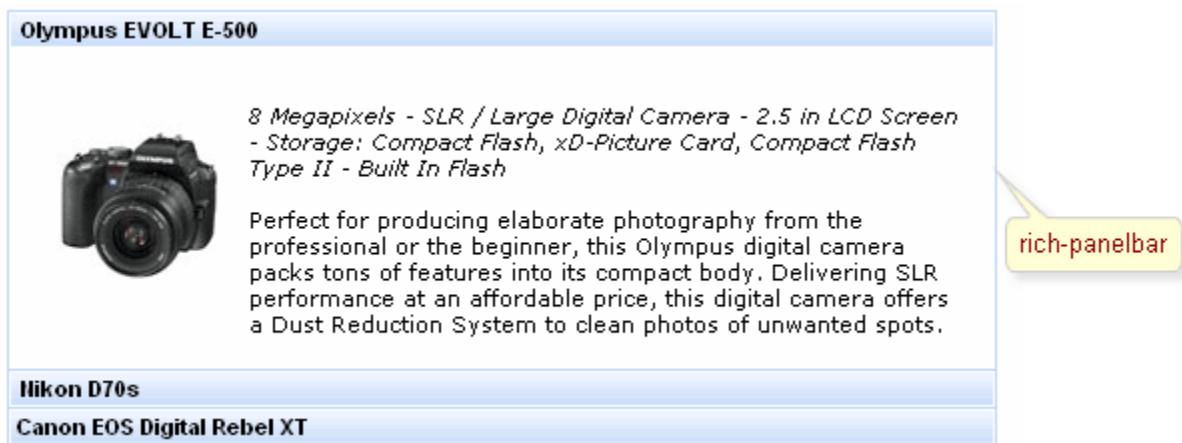


Figure 6.96. Style classes

Table 6.266. Class name that define a component appearance

| Class name | Class description |
|---------------|--|
| rich-panelbar | Defines styles for a wrapper <code><div></code> element of a component |

Other classes responsible for elements rendering are described for child `<rich:panelBarItem>` elements and could be found in the components chapters.

Table 6.267. Style component classes

| A class attribute | A component element defined by an attribute |
|-------------------|---|
| styleClass | Applicable to a whole component (together with headers) |
| headerClass | Applicable to a header element |
| contentClass | Applicable to a content |

In order to redefine styles for all **<rich:panelBar>** components on a page using CSS, it's enough to create classes with the same names and define necessary properties in them.

To change styles of particular **<rich:panelBar>** components, define your own style classes in the corresponding **<rich:panelBar>** attributes.

CSS code piece used on a page:

Example:

```
...
    . rich-panelbar{
        padding:10px;
    }
    .myClass{
        font-style:italic;
    }
...
```

When using `headerClass` and `headerClassActive` attributes the declaration of `headerClass` should precede the one of `headerClassActive`:

Example:

```
...
    .headerClass{
        ...
    }
    .headerClassActive{
        ...
    }
...
```

The component is defined in the following way:

Example:

```
...
    <rich:panelBar contentClass="myClass">
        <rich:panelBarItem>
            ...
        </rich:panelBarItem>
    </rich:panelBar>
...
```

Hence, padding for all **<rich:panelBar>** is changed on a page as well as a font for particular **<rich:panelBarItem>** content.

6.61.9. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/panelBar.jsf?c=panelBar>] you can see the example of **<rich:panelBar>** usage and sources for the given example.

6.62. < rich:panelBarItem >

6.62.1. Description

panelBarItem is used for grouping any content inside within one panelBar which is loaded on client side and appears as groups divided on child panels after header is clicked.



Figure 6.97. <rich:panelBarItem> component

6.62.2. Key Features

- Highly customizable look and feel
- Groups any content inside each Panels

Table 6.268. rich : panelBarItem attributes

| Attribute Name | Description |
|-------------------|---|
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| contentClass | The component content style class |
| contentStyle | The component content style |
| headerClass | The component header style class |
| headerClassActive | The component header style class active |
| headerStyle | The component header style |
| headerStyleActive | The component header style active |
| id | Every component may have a unique id that is automatically created if omitted |
| label | Label text appears on a panel item header |
| name | Attribute defines item name |

| Attribute Name | Description |
|----------------|--|
| rendered | If "false", this component is not rendered |

Table 6.269. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | org.richfaces.PanelBarItem |
| component-class | org.richfaces.component.html.HtmlPanelBarItem |
| component-family | org.richfaces.PanelBarItem |
| renderer-type | org.richfaces.PanelBarItemRenderer |
| tag-class | org.richfaces.taglib.PanelBarItemTag |

6.62.3. Creating the Component with a Page Tag

Here is a simple example as it could be used on a page:

Example:

```
...
<rich:panelBar>
  <rich:panelBarItem label="Canon">
    ...
  </rich:panelBarItem>
  <rich:panelBarItem label="Nikon">
    ...
  </rich:panelBarItem>
</rich:panelBar>
...
```

6.62.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlPanelBarItem;
...
HtmlPanelBarItem myBarItem = new HtmlPanelBarItem();
...
```

6.62.5. Details of Usage

The *"label"* attribute defines text to be represented. If you can use the *"label"* facet, you can even not use the *"label"* attribute.

Example:

```
...
<rich:panelBarItem...>
  <f:facet name="label">
```

```

        <h:graphicImage value="/images/img1.png"/>
    </f:facet>
    ...
    <!--Any Content inside-->
    ...
</rich:panelBarItem>
...

```

As it was mentioned above, `panelBarItem` is used for grouping any content inside within one `panelBar`, thus its customization deals only with specification of sizes and styles for rendering.

`panelBar` could contain any number of child `panelBarItem` components inside, which content is uploaded onto the client and headers are controls to open the corresponding child element.

6.62.6. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

There are two ways to redefine the appearance of all `<rich:panelBarItem>` components at once:

- Redefine the corresponding skin parameters
- Add to your style sheets *style classes* used by a `<rich:panelBarItem>` component

6.62.7. Skin Parameters Redefinition

Table 6.270. Skin parameters redefinition for a content

| Skin parameters | CSS properties |
|---------------------------------------|--------------------------|
| <code>generalTextColor</code> | <code>color</code> |
| <code>preferableDataSizeFont</code> | <code>font-size</code> |
| <code>preferableDataFamilyFont</code> | <code>font-family</code> |

Table 6.271. Skin parameters redefinition for a header element (active or inactive)

| Skin parameters | CSS properties |
|------------------------------------|-------------------------------|
| <code>headerTextColor</code> | <code>color</code> |
| <code>headerBackgroundColor</code> | <code>background-color</code> |
| <code>headerSizeFont</code> | <code>font-size</code> |
| <code>headerWeightFont</code> | <code>font-weight</code> |
| <code>headerFamilyFont</code> | <code>font-family</code> |

6.62.8. Definition of Custom Style Classes

On the screenshot there are classes names that define styles for component elements.

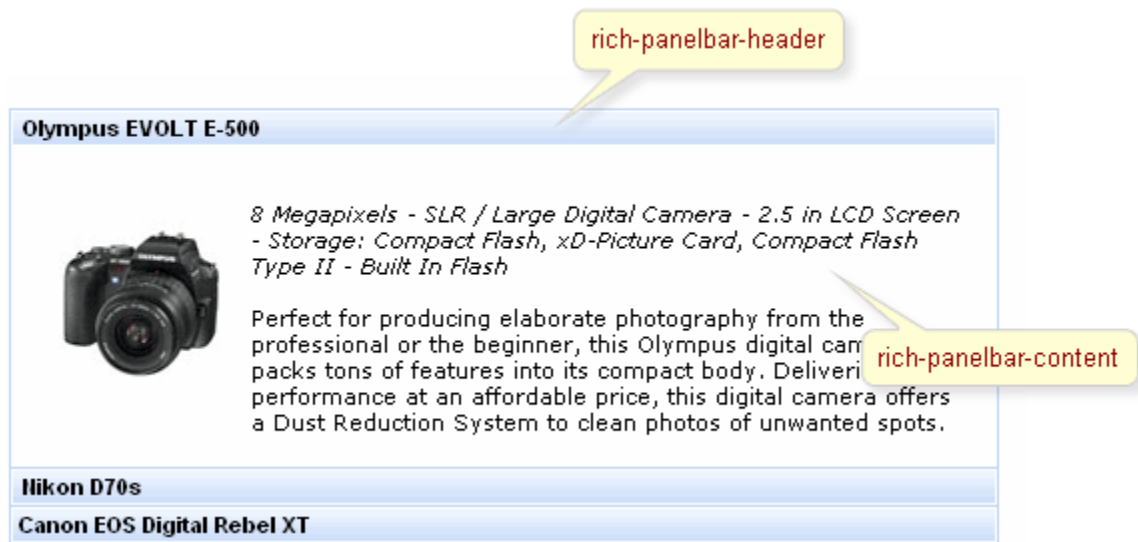


Figure 6.98. Style classes

Table 6.272. Classes names that define a component appearance

| Class name | Class description |
|--------------------------|--|
| rich-panelbar-header | Defines styles for a wrapper <div> element of a header element |
| rich-panelbar-header-act | Defines styles for a wrapper <div> element of an active header element |
| rich-panelbar-content | Defines styles for a content |

Table 6.273. Style component classes

| A class attribute | A component element defined by an attribute |
|-------------------|---|
| headerClass | Applicable to a header element |
| contentClass | Applicable to a content |

In order to redefine styles for all **<rich:panelBarItem>** components on a page using CSS, it's enough to create classes with the same names and define necessary properties in them.

To change styles of particular **<rich:panelBarItem>** components, define your own style classes in the corresponding **<rich:panelBarItem>** attributes.

CSS code piece used on a page:

Example:

```
...
.rich-panelbar-header{
    font-size:14px;
}
.myClass{
```

```

        font-style:italic;
    }
    ...

```

The component is defined in the following way:

Example:

```

...
<rich:panelBar>
    <rich:panelBarItem contentClass="myClass">
        ...
    </rich:panelBarItem>
</rich:panelBar>
...

```

Hence, a font size of all **<rich:panelBarItem>** headers is changed on a page as well as a font for the particular **<rich:panelBarItem>** content.

6.63. < rich:panelMenu >

6.63.1. Description

The **<rich:panelMenu>** component is used to define an inline vertical menu on a page.

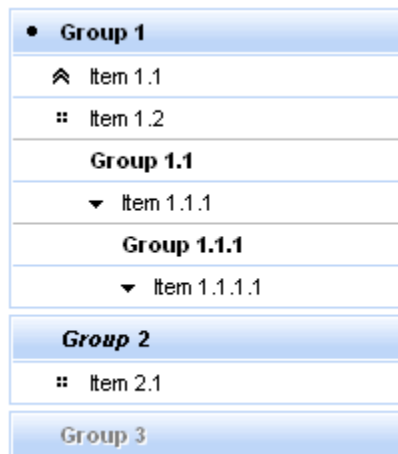


Figure 6.99. <rich:panelMenu> component

6.63.2. Key Features

- Highly customizable look and feel
- Different submission modes
- Collapsing/expanding sublevels with optional request sending
- Custom and predefined icons support
- Disablement support

Table 6.274. rich : panelMenu attributes

| Attribute Name | Description |
|--------------------|---|
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| converter | Id of Converter to be used or reference to a Converter |
| converterMessage | A ValueExpression enabled attribute that, if present, will be used as the text of the converter message, replacing any message that comes from the converter |
| disabled | If true sets state of the item to disabled state. false is default. |
| disabledGroupClass | Space-separated list of CSS style class(es) that are be applied to disabled group of this component |
| disabledGroupStyle | CSS style(s) is/are to be applied to disabled group when this component is rendered |
| disabledItemClass | Space-separated list of CSS style class(es) that are be applied to disabled item of this component |
| disabledItemStyle | CSS style(s) is/are to be applied to disabled item when this component is rendered. |
| event | Defines the event on the representation element that triggers the submenu's expand/collapse. (default=onclick) |
| expandMode | Set the submission mode for all panel menu groups after expand/collapse except ones where this attribute redefined. (ajax, server, none(Default)) |
| expandSingle | Whether only one panel menu node on top level can be opened at a time. If the value of this attribute is true, the previously opened node on the top level is closed. If the value is false, the node is left opened. The default value is false. |
| groupClass | Space-separated list of CSS style class(es) that are be applied to group of this component |
| groupStyle | CSS style(s) is/are to be applied to group when this component is rendered |
| hoveredGroupClass | Space-separated list of CSS style class(es) that are be applied to hovered group of this component |
| hoveredGroupStyle | CSS style(s) is/are to be applied to hovered group when this component is rendered |

| Attribute Name | Description |
|-----------------------|--|
| hoveredItemClass | Space-separated list of CSS style class(es) that are applied to hovered item of this component |
| hoveredItemStyle | CSS style(s) is/are to be applied to hovered item when this component is rendered |
| iconCollapsedGroup | Path to the icon to be displayed for the collapsed Group state |
| iconCollapsedTopGroup | Path to the icon to be displayed for the collapsed top group state |
| iconDisabledGroup | Path to the icon to be displayed for the disabled group state |
| iconDisabledItem | Path to the icon to be displayed for the disabled item state |
| iconExpandedGroup | Path to the icon to be displayed for the expanded Group state |
| iconExpandedTopGroup | Path to the icon to be displayed for the expanded top group state |
| iconGroupPosition | Position of the icon (left, right none (default)) for the group icon |
| iconGroupTopPosition | Position of the icon (left, right none (default)) for the top group icon |
| iconItem | Path to the icon to be displayed for the enabled item state |
| iconItemPosition | Position of the icon (left, right none (default)) for the item icon |
| iconItemTopPosition | Position of the icon (left, right none (default)) for the top item icon |
| iconTopDisabledItem | Path to the icon to be displayed for the disabled top item state |
| iconTopDisableGroup | Path to the icon to be displayed for the disabled top Group state |
| iconTopItem | Path to the icon to be displayed for the enabled top item state |
| id | Every component may have a unique id that is automatically created if omitted |

| Attribute Name | Description |
|-----------------|---|
| immediate | A flag indicating that this component value must be converted and validated immediately (that is, during Apply Request Values phase), rather than waiting until a Process Validations phase |
| itemClass | Space-separated list of CSS style class(es) that are be applied to item of this component |
| itemStyle | CSS style(s) is/are to be applied to item when this component is rendered. |
| label | A localized user presentable name for this component. |
| mode | Set the submission mode for all panel menu items on the panel menu except ones where this attribute redefined. (ajax, server,(Default), none) |
| onclick | HTML: a script expression; a pointer button is clicked |
| ondblclick | HTML: a script expression; a pointer button is double-clicked |
| ongroupcollapse | HTML: script expression; some group was closed |
| ongroupeexpand | HTML: script expression; some group was activated |
| onitemhover | HTML: script expression; some item was hovered |
| onkeydown | HTML: a script expression; a key is pressed down |
| onkeypress | HTML: a script expression; a key is pressed and released |
| onkeyup | HTML: a script expression; a key is released |
| onmousedown | HTML: script expression; a pointer button is pressed down |
| onmousemove | HTML: script expression; a pointer was moved within. |
| onmouseout | HTML: script expression; a pointer was moved away. |
| onmouseover | HTML: script expression; a pointer was moved onto. |
| onmouseup | HTML: script expression; a pointer button is released |
| rendered | If "false", this component is not rendered |
| required | If "true", this component is checked for non-empty input |

| Attribute Name | Description |
|---------------------|--|
| requiredMessage | A ValueExpression enabled attribute that, if present, will be used as the text of the validation message for the "required" facility, if the "required" facility is used |
| selectedChild | contain the name or the clientId of any of the item or group, the child defined in this attribute should be highlighted on PanelMenu rendering |
| style | The CSS style for the panel menu. |
| styleClass | The CSS class for the panel menu. |
| topGroupClass | Space-separated list of CSS style class(es) that are be applied to top group of this component |
| topGroupStyle | CSS style(s) is/are to be applied to top group when this component is rendered |
| topItemClass | Space-separated list of CSS style class(es) that are be applied to top item of this component |
| topItemStyle | CSS style(s) is/are to be applied to top item when this component is rendered |
| validator | MethodBinding pointing at a method that is called during Process Validations phase of the request processing lifecycle, to validate the current value of this component |
| validatorMessage | A ValueExpression enabled attribute that, if present, will be used as the text of the validator message, replacing any message that comes from the validator |
| value | The initial value to set when rendered for the first time |
| valueChangeListener | Listener for value changes |
| width | Set minimal width for the menu. |

Table 6.275. Component identification parameters

| Name | Value |
|------------------|--|
| component-type | org.richfaces.PanelMenu |
| component-class | org.richfaces.component.html.HtmlPanelMenu |
| component-family | org.richfaces.PanelMenu |
| renderer-type | org.richfaces.PanelMenuRenderer |
| tag-class | org.richfaces.taglib.PanelMenuTag |

6.63.3. Creating the Component with a Page Tag

To create the simplest variant on a page use the following syntax:

Example:

```
...
    <rich:panelMenu event="onmouseover">
        <!--Nested panelMenu components-->
    </rich:panelMenu>
...
```

6.63.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlPanelMenu;
...
HtmlPanelMenu myPanelMenu = new HtmlPanelMenu();
...
```

6.63.5. Details of Usage

All attributes are not required.

Use *"event"* attribute to define an event for appearance of collapsing/expanding sublevels. Default value is *"onclick"*. An example could be seen below.

Example:

```
...
    <rich:panelMenu event="onmouseover">
        <!--Nested panelMenu components-->
    </rich:panelMenu>
...
```

Switching mode could be chosen with the *"mode"* attribute for all panelMenu items except ones where this attribute was redefined. By default all items send traditional request.

The *"expandMode"* attribute defines the submission modes for all collapsing/expanding panelMenu groups except ones where this attribute was redefined.

The *"mode"* and *"expandMode"* attributes could be used with three possible parameters.

- Server (default)

Regular form submission request is used.

- Ajax

Ajax submission is used for switching.

- None

"Action" and "ActionListener" item's attributes are ignored. Items don't fire any submits itself. Behavior is fully defined by the components nested into items.

Example:

```
...
<rich:panelMenu event="onclick" submitMode="none">
    < rich:panelMenuItem label="Link to external page">
        <h:outputLink ... >
    </rich:panelMenuItem>
</rich:panelMenu>
...
```

Note:

As the **<rich:panelMenu>** component doesn't provide its own form, use it between **<h:form>** and **</h:form>** tags.

The *"expandSingle"* attribute is defined for expanding more than one submenu on the same level. The default value is *"false"*. If it's true the previously opened group on the top level closes before opening another one. See the picture below.

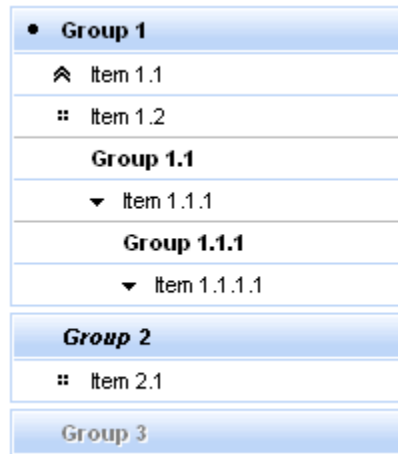


Figure 6.100. Using the *"expandSingle"* attribute

The *"selectedChild"* attribute is used for defining the name of the selected group or item. An example for group is placed below:

Here is an example:

Example:

```
...
<rich:panelMenu selectedChild="thisChild">
    <rich:panelMenuGroup label="Group1" name="thisChild">
        <!--Nested panelMenu components-->
    </rich:panelMenuGroup>
...
```

```
</rich:panelMenu>
...
```

6.63.6. JavaScript API

In Java Script code for expanding/collapsing group element creation it's necessary to use `Expand()`/`Collapse()` function.

Table 6.276. JavaScript API

| Function | Description |
|-------------------------|-------------------------|
| <code>Expand()</code> | Expands group element |
| <code>Collapse()</code> | Collapses group element |

6.63.7. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*.

There are no skin parameters. To redefine the appearance of all `<rich:panelMenu>` components at once, you should add to your style sheets the *style class* used by a `<rich:panelMenu>` component.

6.63.8. Definition of Custom Style Classes

Table 6.277. Classes names that define a component appearance

| Class name | Class description |
|------------------------------|--|
| <code>rich-panel-menu</code> | Defines styles for a wrapper <code><div></code> element of a component |

In order to redefine styles for all `<rich:panelMenu>` components on a page using CSS, it's enough to create a class with the same name and define necessary properties in it.

To change styles of particular `<rich:panelMenu>` components, define your own style class in the corresponding `<rich:panelMenu>` attributes.

6.63.9. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/panelMenu.jsf?c=panelMenu>] you can see the example of `<rich:panelMenu>` usage and sources for the given example.

6.64. < rich:panelMenuGroup >

6.64.1. Description

The `<rich:panelMenuGroup>` component is used to define an expandable group of items inside the panel menu or other group.

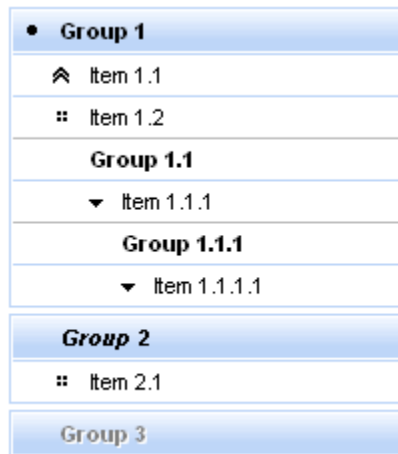


Figure 6.101. <rich:panelMenuGroup> component

6.64.2. Key Features

- Highly customizable look-and-feel
- Different submission modes inside every group
- Optional submissions on expand collapse groups
- Custom and predefined icons supported
- Support for disabling

Table 6.278. rich : panelMenuGroup attributes

| Attribute Name | Description |
|----------------|---|
| accesskey | This attribute assigns an access key to an element. An access key is a single character from the document character set. Note: Authors should consider the input method of the expected reader when specifying an accesskey |
| action | MethodBinding pointing at the application action to be invoked, if this UIComponent is activated by the user, during the Apply Request Values or Invoke Application phase of the request processing lifecycle, depending on the value of the immediate property |
| actionListener | MethodBinding pointing at method accepting an ActionEvent with return type void |
| ajaxSingle | if "true", submits ONLY one field/link, instead of all form controls |
| align | left center right justify [CI] Deprecated. This attribute specifies the horizontal alignment of its element with |

| Attribute Name | Description |
|------------------|--|
| | respect to the surrounding context. Possible values: * left: text lines are rendered flush left. * center: text lines are centered. * right: text lines are rendered flush right. * justify: text lines are justified to both margins. The default depends on the base text direction. For left to right text, the default is align=left, while for right to left text, the default is align=right |
| alt | For a user agents that cannot display images, forms, or applets, this attribute specifies alternate text. The language of the alternate text is specified by the lang attribute |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| bypassUpdates | If "true", after process validations phase it skips updates of model beans on a force render response. It can be used for validating components input |
| converter | Id of Converter to be used or reference to a Converter |
| converterMessage | A ValueExpression enabled attribute that, if present, will be used as the text of the converter message, replacing any message that comes from the converter |
| data | Serialized (on default with JSON) data passed on the client by a developer on AJAX request. It's accessible via "data.foo" syntax |
| disabled | When set for a form control, this boolean attribute disables the control for user input |
| disabledClass | Class to be applied to disabled items. |
| disabledStyle | CSS style rules to be applied to disabled items. |
| eventsQueue | Name of requests queue to avoid send next request before complete other from same event. Can be used to reduce number of requests of frequently events (key press, mouse move etc.) |
| expanded | If true group will be displayed expanded initially. |
| expandMode | Set the submission mode for all panel menu groups after expand/collapse except ones where this attribute redefined. (ajax, server, none(Default)) |
| focus | id of element to set focus after request completed on client side |

| Attribute Name | Description |
|--------------------|--|
| hoverClass | Class to be applied to hovered items. |
| hoverStyle | CSS style rules to be applied to hovered items. |
| iconClass | Class to be applied to icon element. |
| iconCollapsed | Path to the icon to be displayed for the collapsed item state |
| iconDisabled | Path to the icon to be displayed for the disabled item state |
| iconExpanded | Path to the icon to be displayed for the expanded item state |
| iconStyle | CSS style rules to be applied |
| id | Every component may have a unique id that is automatically created if omitted |
| ignoreDupResponses | Attribute allows to ignore an Ajax Response produced by a request if the newest 'similar' request is in a queue already. ignoreDupResponses="true" does not cancel the request while it is processed on the server, but just allows to avoid unnecessary updates on the client side if the response isn't actual now |
| immediate | A flag indicating that this component value must be converted and validated immediately (that is, during Apply Request Values phase), rather than waiting until a Process Validations phase |
| label | Displayed node's text |
| limitToList | If "true", updates on client side ONLY elements from this 'reRender' property. If "false" (default) updates all rendered by ajax region components |
| maxlength | When the type attribute has the value "text" or "password", this attribute specifies the maximum number of characters the user may enter. This number may exceed the specified size, in which case the user agent should offer a scrolling mechanism. The default value for this attribute is an unlimited number |
| name | 'selectedChild' attribute of PanelMenu refers to group/item with the same name |
| onbeforedomupdate | JavaScript code for call before DOM has been updated on client side |

| Attribute Name | Description |
|----------------|---|
| onblur | HTML: script expression; the element lost the focus |
| onchange | HTML: script expression; the element value was changed |
| onclick | HTML: a script expression; a pointer button is clicked |
| oncollapse | HTML: script expression; group was closed |
| oncomplete | JavaScript code for call after request completed on client side |
| ondblclick | HTML: a script expression; a pointer button is double-clicked |
| onexpand | HTML: script expression; group was opened |
| onfocus | HTML: script expression; the element got the focus |
| onkeydown | HTML: a script expression; a key is pressed down |
| onkeypress | HTML: a script expression; a key is pressed and released |
| onkeyup | HTML: a script expression; a key is released |
| onmousedown | HTML: script expression; a pointer button is pressed down |
| onmousemove | HTML: a script expression; a pointer is moved within |
| onmouseout | HTML: a script expression; a pointer is moved away |
| onmouseover | HTML: a script expression; a pointer is moved onto |
| onmouseup | HTML: script expression; a pointer button is released |
| onselect | HTML: script expression; The onselect event occurs when a user selects some text in a text field. This attribute may be used with the INPUT and TEXTAREA elements |
| rendered | If "false", this component is not rendered |
| requestDelay | Attribute defines the time (in ms.) that the request will be wait in the queue before it is ready to send. When the delay time is over, the request will be sent to the server or removed if the newest 'similar' request is in a queue already |
| required | If "true", this component is checked for non-empty input |

| Attribute Name | Description |
|---------------------|--|
| requiredMessage | A ValueExpression enabled attribute that, if present, will be used as the text of the validation message for the "required" facility, if the "required" facility is used |
| reRender | Id[s] (in format of call <code>UIComponent.findComponent()</code>) of components, rendered in case of AjaxRequest caused by this component. Can be single id, comma-separated list of Id's, or EL Expression with array or Collection |
| size | This attribute tells the user agent the initial width of the control. The width is given in pixels except when type attribute has the value "text" or "password". In that case, its value refers to the (integer) number of characters |
| status | ID (in format of call <code>UIComponent.findComponent()</code>) of Request status component |
| style | CSS style(s) to be applied when this component is rendered. |
| styleClass | Corresponds to the HTML class attribute. |
| tabindex | This attribute specifies the position of the current element in the tabbing order for the current document. This value must be a number between 0 and 32767. User agents should ignore leading zeros |
| target | Target frame for action to execute. |
| timeout | Response waiting time on a particular request. If a response is not received during this time, the request is aborted |
| validator | MethodBinding pointing at a method that is called during Process Validations phase of the request processing lifecycle, to validate the current value of this component |
| validatorMessage | A ValueExpression enabled attribute that, if present, will be used as the text of the validator message, replacing any message that comes from the validator |
| value | The initial value to set when rendered for the first time |
| valueChangeListener | Listener for value changes |

Table 6.279. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | org.richfaces.PanelMenuGroup |
| component-class | org.richfaces.component.html.HtmlPanelMenuGroup |
| component-family | org.richfaces.PanelMenuGroup |
| renderer-type | org.richfaces.PanelMenuGroupRenderer |
| tag-class | org.richfaces.taglib.PanelMenuGroupTag |

6.64.3. Creating the Component with a Page Tag

To create the simplest variant on a page use the following syntax:

Example:

```
...
<rich:panelMenu>
    <rich:panelMenuGroup label="Group1">
        <!--Nested panelMenu components-->
    </rich:panelMenuGroup>
</rich:panelMenu>
...
```

6.64.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlPanelMenuGroup;
...
HtmlPanelMenuGroup myPanelMenuGroup = new HtmlPanelMenuGroup();
...
```

6.64.5. Details of Usage

All attributes except *"label"* are optional. The *"label"* attribute defines text to be represented.

Switching mode could be chosen with the *"expandMode"* attribute for the concrete panelMenu group.

The *"expandMode"* attribute could be used with three possible parameters:

- Server (default)

Regular form submission request is used.

- Ajax

Ajax submission is used for switching.

- None

"*Action*" and "*actionListener*" attributes are ignored. Items don't fire any submits itself. Behavior is fully defined by the components nested into items.

There are three icon-related attributes. The "*iconExpanded*" attribute defines an icon for an expanded state. The "*iconCollapsed*" attribute defines an icon for a collapsed state. The "*iconDisabled*" attribute defines an icon for a disabled state.

Default icons are shown on the picture below:



Figure 6.102. Default icons

Here is an example:

Example:

```
...
    <rich:panelMenu>
        <rich:panelMenuGroup label="Group1" iconExpanded="disc"
iconCollapsed="chevron">
            <!--Nested panelMenu components-->
        </rich:panelMenuGroup>
    </rich:panelMenu>
...
```

As the result the pictures are shown below. The first one represents the collapsed state, the second one - expanded state:

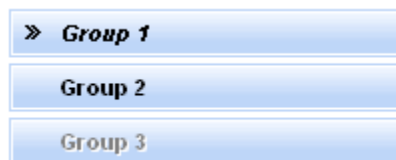


Figure 6.103. Collapsed state

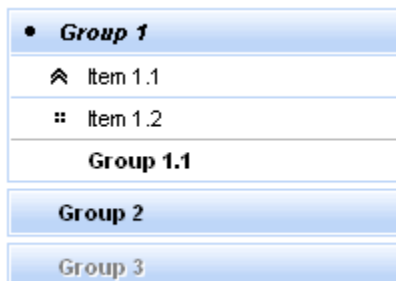


Figure 6.104. Expanded state

It's also possible to define a path to the icon. Simple code is placed below.

```
...
    <rich:panelMenu>
        <rich:panelMenuGroup label="Group1" iconExpanded="\images\img1.png"
        iconCollapsed="\images\img2.png">
            <!--Nested menu components-->
        </rich:panelMenuGroup>
    </rich:panelMenu>
...
```

6.64.6. JavaScript API

In Java Script code for expanding/collapsing group element creation it's necessary to use `Expand()`/`Collapse()` function.

Table 6.280. JavaScript API

| Function | Description |
|-------------------------|------------------------|
| <code>Expand()</code> | Expand group element |
| <code>Collapse()</code> | Collapse group element |

6.64.7. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

There are two ways to redefine the appearance of all `<rich:panelMenuGroup>` components at once:

- Redefine the corresponding skin parameters
- Add to your style sheets *style classes* used by a `<rich:panelMenuGroup>` component

6.64.8. Skin Parameters Redefinition

Table 6.281. Skin parameters redefinition for a table element of the first level group

| Skin parameters | CSS properties |
|------------------------------------|-------------------------------|
| <code>headerWeightFont</code> | <code>font-weight</code> |
| <code>generalFamilyFont</code> | <code>font-family</code> |
| <code>headerSizeFont</code> | <code>font-size</code> |
| <code>headerTextColor</code> | <code>color</code> |
| <code>headerBackgroundColor</code> | <code>background-color</code> |

Table 6.282. Skin parameters redefinition for a table element of second and next level groups

| Skin parameters | CSS properties |
|------------------|------------------|
| headerWeightFont | font-weight |
| headerFamilyFont | font-family |
| headerSizeFont | font-size |
| generalTextColor | color |
| tableBorderColor | border-top-color |

Table 6.283. Skin parameters redefinition for wrapper div element of the first level group

| Skin parameters | CSS properties |
|------------------|----------------|
| panelBorderColor | border-color |

Table 6.284. Skin parameters redefinition for a hovered group element

| Skin parameters | CSS properties |
|---------------------------|------------------|
| additionalBackgroundColor | background-color |

Table 6.285. Skin parameters redefinition for a disabled group element

| Skin parameters | CSS properties |
|----------------------|----------------|
| tabDisabledTextColor | color |

6.64.9. Definition of Custom Style Classes

On the screenshot there are classes names that define styles for component elements.

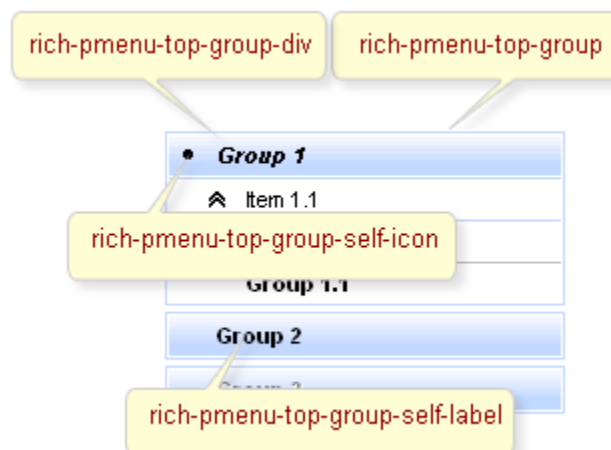
**Figure 6.105. Classes names**

Table 6.286. Classes names that define an upper level groups

| Class name | Description |
|---------------------------------|---|
| rich-pmenu-top-group-div | Defines styles for a wrapper <div> element of a top group |
| rich-pmenu-top-group | Defines styles for a top group |
| rich-pmenu-top-group-self-icon | Defines styles for a top group icon |
| rich-pmenu-top-group-self-label | Defines styles for a top group label |

Table 6.287. Classes names that define a second and lower level groups

| Class name | Description |
|-----------------------------|---|
| rich-pmenu-group-div | Defines styles for a wrapper <div> element of a group |
| rich-pmenu-group | Defines styles for a group |
| rich-pmenu-group-self-icon | Defines styles for a group icon |
| rich-pmenu-group-self-label | Defines styles for a group label |

Table 6.288. Classes names that define a group state

| Class name | Description |
|-----------------------------|---|
| rich-pmenu-hovered-element | Defines styles for a hovered group element |
| rich-pmenu-disabled-element | Defines styles for a disabled group element |

In order to redefine styles for all **<rich:panelMenuGroup>** components on a page using CSS, it's enough to create classes with the same names and define necessary properties in them.

To change styles of particular **<rich:panelMenuGroup>** components, define your own style classes in the corresponding **<rich:panelMenuGroup>** attributes.

6.64.10. Relevant resources links

Some additional information about usage of component can be found here. [<http://livedemo.exadel.com/richfaces-demo/richfaces/dropDownMenu.jsf?c=menuGroup>]

6.65. < rich:panelMenuItem >

6.65.1. Description

The **<rich:panelMenuItem>** component is used to define a single item inside popup list.

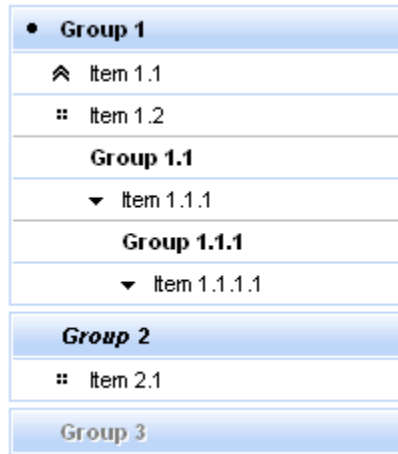


Figure 6.106. <rich:panelMenuItem> component

6.65.2. Key Features

- Highly customizable look-and-feel
- Different submission modes
- Optionally supports any content inside
- Custom and predefined icons supported
- Support for disabling

Table 6.289. rich : panelMenuItem attributes

| Attribute Name | Description |
|------------------|---|
| action | MethodBinding pointing at the application action to be invoked, if this UIComponent is activated by the user, during the Apply Request Values or Invoke Application phase of the request processing lifecycle, depending on the value of the immediate property |
| actionExpression | actionExpression |
| actionListener | MethodBinding pointing at method accepting an ActionEvent with return type void |
| ajaxSingle | if "true", submits ONLY one field/link, instead of all form controls |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| bypassUpdates | If "true", after process validations phase it skips updates of model beans on a force render response. It can be used for validating components input |

| Attribute Name | Description |
|--------------------|--|
| data | Serialized (on default with JSON) data passed on the client by a developer on AJAX request. It's accessible via "data.foo" syntax |
| disabled | If true sets state of the item to disabled state. false is default. |
| disabledClass | Class to be applied to disabled items. |
| disabledStyle | CSS style rules to be applied to disabled items. |
| eventsQueue | Name of requests queue to avoid send next request before complete other from same event. Can be used to reduce number of requests of frequently events (key press, mouse move etc.) |
| focus | id of element to set focus after request completed on client side |
| hoverClass | Class to be applied to hovered items. |
| hoverStyle | CSS style rules to be applied to hovered items. |
| icon | Path to the icon or the default one name to be displayed for the enabled item state |
| iconClass | Class to be applied to icon element. |
| iconDisabled | Path to the icon to be displayed for the disabled item state |
| iconStyle | CSS style rules to be applied |
| id | Every component may have a unique id that is automatically created if omitted |
| ignoreDupResponses | Attribute allows to ignore an Ajax Response produced by a request if the newest 'similar' request is in a queue already. ignoreDupResponses="true" does not cancel the request while it is processed on the server, but just allows to avoid unnecessary updates on the client side if the response isn't actual now |
| immediate | True means, that the default ActionListener should be executed immediately (i.e. during Apply Request Values phase of the request processing lifecycle), rather than waiting until the Invoke Application phase |
| label | Defines representation text for menuItem. |

| Attribute Name | Description |
|-------------------|---|
| limitToList | If "true", updates on client side ONLY elements from this 'reRender' property. If "false" (default) updates all rendered by ajax region components |
| mode | Set the submission mode (ajax,server(Default),none) |
| name | 'selectedChild' attribute of PanelMenu refers to group/item with the same name |
| onbeforedomupdate | JavaScript code for call before DOM has been updated on client side |
| onclick | HTML: a script expression; a pointer button is clicked |
| oncomplete | JavaScript code for call after request completed on client side |
| ondblclick | HTML: a script expression; a pointer button is double-clicked |
| onkeydown | HTML: a script expression; a key is pressed down |
| onkeypress | HTML: a script expression; a key is pressed and released |
| onkeyup | HTML: a script expression; a key is released |
| onmousedown | HTML: script expression; a pointer button is pressed down |
| onmousemove | HTML: a script expression; a pointer is moved within |
| onmouseout | HTML: a script expression; a pointer is moved away |
| onmouseover | HTML: a script expression; a pointer is moved onto |
| onmouseup | HTML: script expression; a pointer button is released |
| rendered | If "false", this component is not rendered |
| requestDelay | Attribute defines the time (in ms.) that the request will be wait in the queue before it is ready to send. When the delay time is over, the request will be sent to the server or removed if the newest 'similar' request is in a queue already |
| reRender | Id[s] (in format of call UIComponent.findComponent()) of components, rendered in case of AjaxRequest caused by this component. Can be single id, comma-separated list of Id's, or EL Expression with array or Collection |

| Attribute Name | Description |
|----------------|---|
| status | ID (in format of call <code>UIComponent.findComponent()</code>) of Request status component |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| target | Target frame for action to execute. |
| timeout | Response waiting time on a particular request. If a response is not received during this time, the request is aborted |
| value | The current value for this component |

Table 6.290. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | <code>org.richfaces.PanelMenuItem</code> |
| component-class | <code>org.richfaces.component.html.HtmlPanelMenuItem</code> |
| component-family | <code>org.richfaces.PanelMenuItem</code> |
| renderer-type | <code>org.richfaces.PanelMenuItemRenderer</code> |
| tag-class | <code>org.richfaces.taglib.PanelMenuItemTag</code> |

6.65.3. Creating the Component with a Page Tag

To create the simplest variant on a page use the following syntax:

Example:

```
...
    <rich:panelMenu>
        ...
        <rich:panelMenuItem value="Item1"/>
        ...
    </rich:panelMenu>
...
```

6.65.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlPanelMenuItem;
...
HtmlPanelMenuItem myPanelMenuItem = new HtmlPanelMenuItem();
...
```

6.65.5. Details of Usage

All attributes except *"label"* are optional. The *"label"* attribute defines text to be represented.

The *"mode"* attribute could be used with three possible parameters:

- Server (default)

Regular form submission request is used.

- Ajax

Ajax submission is used for switching.

- None

"Action" and *"actionListener"* attributes are ignored. Items don't fire any submits itself. Behavior is fully defined by the components nested into items.

Here is an example for value *"none"*:

Example:

```
...
    <rich:panelMenu>
        ...
        <rich:panelMenuItem submitMode="none"
onclick="document.location.href='http://labs.jboss.com/jbossrichfaces/'>
            <h:outputLink value="http://labs.jboss.com/jbossrichfaces/">
                <h:outputText value="RichFaces Home Page"></h:outputText>
            </h:outputLink>
        </rich:panelMenuItem>
        ...
    </rich:panelMenu>
...

```

There are two icon-related attributes. The *"icon"* attribute defines an icon. The *"iconDisabled"* attribute defines an icon for a disabled item.

Default icons are shown on the picture below:



Figure 6.107. Default icons

Here is an example:

Example:

```
...
```

```

<rich:panelMenu>
    ...
    <rich:panelMenuItem = "Item 1.1" icon="chevronUp" />
    ...
</rich:panelMenu>
...

```

As the result the picture is shown below:

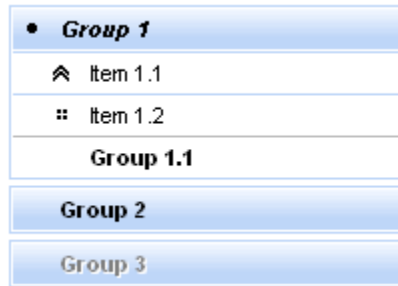


Figure 6.108. Using an *"icon"* attribute

It's also possible to define a path to the icon. Simple code is placed below.

```

...
<rich:panelMenu>
    ...
    <rich:panelMenuItem = "Item 1.1" icon="\images\img1.png" />
    ...
</rich:panelMenu>
...

```

6.65.6. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

There are two ways to redefine the appearance of all `<rich:panelMenuItem>` components at once:

- Redefine the corresponding skin parameters
- Add to your style sheets *style classes* used by a `<rich:panelMenuItem>` component

6.65.7. Skin Parameters Redefinition

Table 6.291. Skin parameters redefinition for a table element of the first level item

| Skin parameters | CSS properties |
|-------------------|----------------|
| generalFamilyFont | font-family |
| generalWeightFont | font-weight |
| generalSizeFont | font-size |
| generalTextColor | color |

| Skin parameters | CSS properties |
|------------------|------------------|
| panelBorderColor | border-top-color |

Table 6.292. Skin parameter redefinition for a disabled item

| Parameter for disabled item | CSS properties |
|-----------------------------|----------------|
| tabDisabledTextColor | color |

6.65.8. Definition of Custom Style Classes

On the screenshot there are classes names that define styles for component elements.

**Figure 6.109. Classes names****Table 6.293. Classes names that define the first level items**

| Class name | Description |
|---------------------------|--|
| rich-pmenu-top-item | Defines styles for a top panel menu item |
| rich-pmenu-top-item-icon | Defines styles for a top panel menu item icon |
| rich-pmenu-top-item-label | Defines styles for a top panel menu item label |

Table 6.294. Classes names that define the second and lower level items

| Class name | Description |
|-----------------------|--|
| rich-pmenu-item | Defines styles for a panel menu item |
| rich-pmenu-item-icon | Defines styles for a panel menu item icon |
| rich-pmenu-item-label | Defines styles for a panel menu item label |

Table 6.295. Classes names that define items state

| Class name | Description |
|-----------------------------|---|
| rich-pmenu-item-selected | Defines styles for a panel menu selected item |
| rich-pmenu-disabled-element | Defines styles for a disabled panel menu item |

| Class name | Description |
|----------------------------|--|
| rich-pmenu-hovered-element | Defines styles for a hovered panel menu item |

In order to redefine styles for all `<rich:panelMenuItem>` components on a page using CSS, it's enough to create classes with the same names and define necessary properties in them.

To change styles of particular `<rich:panelMenuItem>` components, define your own style classes in the corresponding `<rich:panelMenuItem>` attributes.

6.65.9. Relevant resources links

Some additional information about usage of component can be found here. [<http://livedemo.exadel.com/richfaces-demo/richfaces/dropDownMenu.jsf?c=menuItem>]

6.66. < rich:scrollableDataTable >

6.66.1. Description

The `<rich:scrollableDataTable>` component is used for the table-like component creation. The component just adds the set of additional features described below in comparison with the standard table.



| State | Flag | Capital |
|-------------|---|-------------|
| Alabama |  | Montgomery |
| Alaska |  | Juneau |
| Arizona |  | Phoenix |
| Arkansas |  | Little Rock |
| California |  | Sacramento |
| Colorado |  | Denver |
| Connecticut |  | Hartford |
| Delaware |  | Dover |
| Florida |  | Tallahassee |
| Georgia |  | Atlanta |
| Hawaii |  | Honolulu |
| Idaho |  | Boise |
| Illinois |  | Springfield |
| Iowa |  | Des Moines |
| Kansas |  | Topeka |
| Kentucky |  | Frankfort |

Figure 6.110. `<rich:scrollableDataTable>` component

6.66.2. Key Features

- Highly customizable look and feel

- Variable content of the table cells
- Dynamically fetching the rows from the server when the table is scrolled up and down
- Resizing columns by mouse dragging the column bar
- Sorting column by clicking the header
- Fixed one or more left columns when table is scrolled horizontally
- One and multi-selection rows mode
- Built-in drag-n-drop support

Table 6.296. rich : scrollableDataTable attributes

| Attribute Name | Description |
|----------------|--|
| activeClass | A CSS class to be applied to an active row |
| ajaxKeys | This attribute defines rows that are updated after an AJAX request |
| ajaxSingle | if "true", submits ONLY one field/link, instead of all form controls |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| bypassUpdates | If "true", after process validations phase it skips updates of model beans on a force render response. It can be used for validating components input |
| captionClass | Space-separated list of CSS style class(es) that are be applied to caption for this component |
| columnClasses | Comma-delimited list of CSS style classes that are be applied to the columns of this table. A space separated list of classes may also be specified for any individual column. If the number of elements in this list is less than the number of columns specified in the "columns" attribute, no "class" attribute is output for each column greater than the number of elements in the list. If the number of elements in the list is greater than the number of columns specified in the "columns" attribute, the elements at the position in the list after the value of the "columns" attribute are ignored |
| componentState | It defines EL-binding for a component state for saving or redefinition |
| data | |

| Attribute Name | Description |
|--------------------|--|
| | Serialized (on default with JSON) data passed on the client by a developer on AJAX request. It's accessible via "data.foo" syntax |
| eventsQueue | Name of requests queue to avoid send next request before complete other from same event. Can be used to reduce number of requests of frequently events (key press, mouse move etc.) |
| first | A zero-relative row number of the first row to display |
| focus | id of element to set focus after request completed on client side |
| footerClass | Space-separated list of CSS style class(es) that are be applied to any footer generated for this table |
| frozenColCount | Defines the number of the fixed columns from the left side that will not be scrolled via horizontal scroll. Default value is '0' |
| headerClass | Space-separated list of CSS style class(es) that are be applied to any header generated for this table |
| height | Defines a height of the component. Default value is 500px |
| hideWhenScrolling | If 'true' data will be hidden during scrolling. Can be used for increase performance. Default value is 'false' |
| id | Every component may have a unique id that is automatically created if omitted |
| ignoreDupResponses | Attribute allows to ignore an Ajax Response produced by a request if the newest 'similar' request is in a queue already. ignoreDupResponses="true" does not cancel the request while it is processed on the server, but just allows to avoid unnecessary updates on the client side if the response isn't actual now |
| limitToList | If "true", updates on client side ONLY elements from this 'reRender' property. If "false" (default) updates all rendered by ajax region components |
| onbeforedomupdate | JavaScript code for call before DOM has been updated on client side |
| oncomplete | JavaScript code for call after request completed on client side |

| Attribute Name | Description |
|-------------------|--|
| onRowClick | HTML: a script expression; a pointer button is clicked on row |
| onRowDbClick | HTML: a script expression; a pointer button is double-clicked on row |
| onRowMouseDown | HTML: script expression; a pointer button is pressed down on row |
| onRowMouseUp | HTML: script expression; a pointer button is released on row |
| onselectionchange | HTML: script expression to invoke on changing of rows selection |
| rendered | If "false", this component is not rendered |
| requestDelay | Attribute defines the time (in ms.) that the request will be wait in the queue before it is ready to send. When the delay time is over, the request will be sent to the server or removed if the newest 'similar' request is in a queue already |
| reRender | Id[s] (in format of call <code>UIComponent.findComponent()</code>) of components, rendered in case of <code>AjaxRequest</code> caused by this component. Can be single id, comma-separated list of Id's, or EL Expression with array or Collection |
| rowClasses | A comma-delimited list of CSS style classes that is applied to popup table rows. A space separated list of classes may also be specified for any individual row. The styles are applied, in turn, to each row in the table. For example, if the list has two elements, the first style class in the list is applied to the first row, the second to the second row, the first to the third row, the second to the fourth row, etc. In other words, we keep iterating through the list until we reach the end, and then we start at the beginning again |
| rowKey | The attribute is a representation of an identifier for a specific data row |
| rowKeyVar | The attribute provides access to a row key in a Request scope |
| rows | A number of rows to display, or zero for all remaining rows in the table |

| Attribute Name | Description |
|----------------|---|
| scriptVar | Name of JavaScript variable corresponding to component |
| selectedClass | Name of the CSS class for a selected row |
| selection | Value binding representing selected rows |
| sortMode | Defines mode of sorting. Possible values are 'single' for sorting of one column and 'multi' for some. |
| sortOrder | ValueBinding pointing at a property of a class to manage rows sorting |
| stateVar | The attribute provides access to a component state on the client side |
| status | ID (in format of call <code>UIComponent.findComponent()</code>) of Request status component |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| timeout | Response waiting time on a particular request. If a response is not received during this time, the request is aborted |
| value | The current value for this component |
| var | A request-scope attribute via which the data object for the current row will be used when iterating |
| width | Defines a width of the component. Default value is 700px |

Table 6.297. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | <code>org.richfaces.component.ScrollableDataTable</code> |
| component-class | <code>org.richfaces.component.html.HtmlScrollableDataTable</code> |
| component-family | <code>org.richfaces.component.ScrollableDataTable</code> |
| renderer-type | <code>org.richfaces.renderkit.html.ScrollableDataTableRenderer</code> |
| tag-class | <code>org.richfaces.taglib.ScrollableDataTableTag</code> |

6.66.3. Creating the Component with a Page Tag

Here is a simple example as it could be used on a page:

Example:

```

...
<rich:scrollableDataTable value="#{dataTableScrollerBean.allCars}"
var="category">
  <!--...//Set of columns and header/footer facets-->
</rich:scrollableDataTable>
...

```

6.66.4. Creating the Component Dynamically Using Java**Example:**

```

import org.richfaces.component.html.HtmlScrollableDataTable;
...
HtmlScrollableDataTable myScrollableDataTable = new HtmlScrollableDataTable();
...

```

6.66.5. Details of Usage

The component represents on a page as a scrollable table with some fixed (non-scrollable) rows (with header and footer) and columns. Like other tables `<rich:scrollableDataTable>` also has optional footer and header that could be implemented using the corresponding facets. Columns of the table are optionally resizable. Resizing is available using "drag and drop" of the column vertical borders. There is possibility to expand or collapse the columns through JS API on the client side. You can define the number of the fixed columns from the left side using attribute *"frozenColCount"* that will not be scrolled via horizontal scroll.

There is possibility to increase component performance using attribute *"hideWhenScrolling"* . If attribute value is 'true' data will be hidden during scrolling.

It's possible to select the whole row with onclick on the row or some set of rows. Selection is optional and availability of such feature is defined on the component. There are two ways to select a few rows:

- Just clicking the columns one by one.
- Clicking some row with the SHIFT button hold. In this case all the rows starting from last selected up to clicked should be selected.

The columns provides the possibility of expanding/collapsing on the client side through the next JS API:

- `Collapse(columnId)` - Performs the collapse action for the column with the corresponding id

It's possible to sort the table content after clicks on the header. The feature is optional. Every column should be pointed to the comparator method that will be used for sorting the table. In case the `<rich:scrollableDataTable>` is already sorted by some column and the header of this column has been clicked again - the sorting will be reversed.

The typical variant of using:

```

...

```

```

<rich:scrollableDataTable value="#{modelBuilder.model}" var="issues"
    frozenColCount="1"
    first="0"
    rows="40"
    width="300px"
    height="396px">
  <rich:column width="100px">
    <f:facet name="header" >
      <h:outputText value="State"/>
    </f:facet>
    <h:outputText value="#{issues.cell11}"/>
    <f:facet name="footer">
      <h:outputText value="State"/>
    </f:facet>
  </rich:column>
  <!--...//Set of columns and header/footer facets-->
</rich:scrollableDataTable>
...

```

The *"selection"* attribute allows to get the row data when using one and multi-selection rows mode.

This attribute is a reference to object to the instance of `org.richfaces.model.selection.Selection` interface, containing current collection of objects selected by you.

In the following example when you submits the form current collection of objects selected by you is placed in the object's property. Then on complete action the **<rich:modalPanel>** with selected data will be shown.

Example:

```

...
<h:form>
  <rich:spacer height="30" />
  <rich:scrollableDataTable rowKeyVar="rkv" frozenColCount="1" height="200px"
    width="300px" id="carList" rows="40" columnClasses="col"
    value="#{dataTableScrollerBean.allCars}" var="category" sortMode="single"
    selection="#{dataTableScrollerBean.selection}">
    <rich:column id="make">
      <f:facet name="header"><h:outputText styleClass="headerText" value="Make"
/></f:facet>
      <h:outputText value="#{category.make}" />
    </rich:column>
    <rich:column id="model">
      <f:facet name="header"><h:outputText styleClass="headerText" value="Model"
/></f:facet>
      <h:outputText value="#{category.model}" />
    </rich:column>
    <rich:column id="price">
      <f:facet name="header"><h:outputText styleClass="headerText" value="Price"
/></f:facet>
      <h:outputText value="#{category.price}" />
    </rich:column>
  </rich:scrollableDataTable>
  <rich:spacer height="20px"/>
  <a4j:commandButton value="Show Current Selection" reRender="table"

```

```

        action="#{dataTableScrollerBean.takeSelection}"
        oncomplete="javascript:Richfaces.showModalPanel('panel');" />
    </h:form>
    <rich:modalPanel id="panel" autosized="true">
        <f:facet name="header">
            <h:outputText value="Selected Rows" />
        </f:facet>
        <f:facet name="controls">
            <span style="cursor:pointer"
onclick="javascript:Richfaces.hideModalPanel('panel');">X</span>
        </f:facet>
        <rich:dataTable value="#{dataTableScrollerBean.selectedCars}" var="sel"
id="table">
            <rich:column>
                <f:facet name="header"><h:outputText value="Make" /></f:facet>
                <h:outputText value="#{sel.make}" />
            </rich:column>
            <rich:column id="model">
                <f:facet name="header"><h:outputText value="Model" /></f:facet>
                <h:outputText value="#{sel.model}" />
            </rich:column>
            <rich:column id="price">
                <f:facet name="header"><h:outputText value="Price" /></f:facet>
                <h:outputText value="#{sel.price}" />
            </rich:column>
        </rich:dataTable>
    </rich:modalPanel>
    ...

```

This is a result:

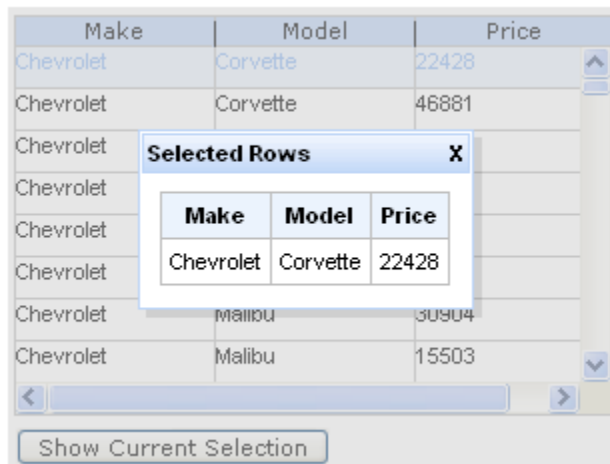


Figure 6.111. The *"selection"* attribute usage

The `<rich:scrollableDataTable>` component has the following extra attributes for event processing on the client:

- onselectionchange
- oncomplete

- onRowClick
- onRowDbClick
- onRowMouseUp
- onRowMouseDown

6.66.6. JavaScript API

Table 6.298. JavaScript API

| Function | Description |
|--------------------|---|
| Collapse(columnId) | Performs a collapse action for column with corresponding Id |

6.66.7. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

There are two ways to redefine the appearance of all `<rich:scrollableDataTable>` components at once:

- Redefine the corresponding skin parameters
- Add to your style sheets *style classes* used by a `<rich:scrollableDataTable>` component

6.66.8. Skin Parameters Redefinition

Table 6.299. Skin parameters for all table

| Skin parameters | CSS properties |
|----------------------|------------------|
| tableBackgroundColor | background-color |
| tableBorderColor | border-color |
| tableBorderWidth | border-width |

Table 6.300. Skin parameters for header rows and cells

| Skin parameters | CSS properties |
|-----------------------|------------------|
| headerBackgroundColor | background-color |
| headerTextColor | color |
| generalFamilyFont | font-family |
| generalSizeFont | font-size |

| Skin parameters | CSS properties |
|------------------|---------------------|
| tableBorderWidth | border-bottom-width |
| tableBorderColor | border-bottom-color |
| tableBorderWidth | border-right-width |
| tableBorderColor | border-right-color |

Table 6.301. Skin parameters for footer rows and cells

| Skin parameters | CSS properties |
|-------------------------------|--------------------|
| tableSubfooterBackgroundColor | background-color |
| generalFamilyFont | font-family |
| generalSizeFont | font-size |
| tableBorderColor | border-right-color |
| generalFamilyFont | font-family |
| generalSizeFont | font-size |

Table 6.302. Skin parameters for column cells

| Skin parameters | CSS properties |
|------------------|---------------------|
| tableBorderColor | border-right-color |
| tableBorderColor | border-bottom-color |

Table 6.303. Skin parameters for active rows

| Skin parameters | CSS properties |
|----------------------|----------------|
| tabDisabledTextColor | color |

Table 6.304. Skin parameters for selected rows

| Skin parameters | CSS properties |
|---------------------------|------------------|
| additionalBackgroundColor | background-color |

6.66.9. Definition of Custom Style Classes

On the screenshot there are classes names that define styles for component elements.



Figure 6.112. Classes names

Table 6.305. Classes names that define a component appearance

| Class name | Description |
|------------|---|
| rich-sdt | Defines styles for a component appearance |

Table 6.306. Classes names that define footer and header elements

| Class name | Description |
|----------------------|--------------------------------------|
| rich-sdt-header-cell | Defines styles for header cells |
| rich-sdt-header-row | Defines styles for a header row |
| rich-sdt-column-cell | Defines styles for column cells |
| rich-sdt-footer-cell | Defines styles for footer cells |
| rich-sdt-footer-row | Defines styles for a footer row |
| rich-sdt-hsep | Defines styles for header separators |

Table 6.307. Classes names that define different states

| Class name | Description |
|---------------------------|---|
| rich-sdt-row-active | Defines styles for an active row |
| rich-sdt-row-selected | Defines styles for a selected row |
| rich-sdt-column-sort-up | Defines styles for ascending sorted column |
| rich-sdt-column-sort-down | Defines styles for descending sorted column |

In order to redefine styles for all `<rich:scrollableDataTable>` components on a page using CSS, it's enough to create classes with the same names and define necessary properties in them.

To change styles of particular `<rich:scrollableDataTable>` components, define your own style classes in the corresponding `<rich:scrollableDataTable>` attributes.

6.66.10. Relevant Resources Links

Here [\[http://livedemo.exadel.com/richfaces-demo/richfaces/scrollableDataTable.jsf?c=scrollableDataTable\]](http://livedemo.exadel.com/richfaces-demo/richfaces/scrollableDataTable.jsf?c=scrollableDataTable) you can see the example of `<rich:scrollableDataTable>` usage.

6.67. < rich:separator >

6.67.1. Description

A horizontal line to use as a separator in a layout. The line type can be customized with the *"lineType"* parameter.

**Figure 6.113. <rich:separator> component**

6.67.2. Key Features

- Highly customizable look and feel
- Leveraging layout elements creation

Table 6.308. rich : separator attributes

| Attribute Name | Description |
|----------------|--|
| align | left center right [CI] This attribute specifies a position of the separator according to the document. Permitted values: * left: The separator is to the left of the |

| Attribute Name | Description |
|----------------|--|
| | document. * center: The separator is to the center of the document. * right: The separator is to the right of the document |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| height | The separator height. Default value is 6 pixels |
| id | Every component may have a unique id that is automatically created if omitted |
| lineType | A line type. The possible values are beveled (default), dotted, dashed, double and solid |
| onclick | HTML: a script expression; a pointer button is clicked |
| ondblclick | HTML: a script expression; a pointer button is double-clicked |
| onkeydown | HTML: a script expression; a key is pressed down |
| onkeypress | HTML: a script expression; a key is pressed and released |
| onkeyup | HTML: a script expression; a key is released |
| onmousedown | HTML: script expression; a pointer button is pressed down |
| onmousemove | HTML: a script expression; a pointer is moved within |
| onmouseout | HTML: a script expression; a pointer is moved away |
| onmouseover | HTML: a script expression; a pointer is moved onto |
| onmouseup | HTML: script expression; a pointer button is released |
| rendered | If "false", this component is not rendered |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| title | HTML: An advisory title for this element. Often displayed as a tooltip |
| width | The separator width that can be defined in pixels or in percents. The default value is 100% |

Table 6.309. Component identification parameters

| Name | Value |
|------------------|--|
| component-type | org.richfaces.separator |
| component-class | org.richfaces.component.html.HtmlSeparator |
| component-family | org.richfaces.separator |
| renderer-type | org.richfaces.SeparatorRenderer |
| tag-class | org.richfaces.taglib.SeparatorTag |

6.67.3. Creating the Component with a Page Tag

Here is a simple example as it could be used on a page:

Example:

```
...
    <rich:separator/>
...
```

6.67.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlSeparator;
...
HtmlSeparator mySeparator = new HtmlSeparator();
...
```

6.67.5. Details of Usage

<rich:separator> is a simple layout component which represents a separator stylized as a skin. Thus, the main attributes that define its style are *"style"* and *"styleClass"*. In addition there are *width* and *height* attributes that should be specified in pixels.

The line type can be customized with the *"lineType"* parameter. For example, different line types are shown after rendering with the following initial settings *lineType="double"* and *"lineType="solid"*.

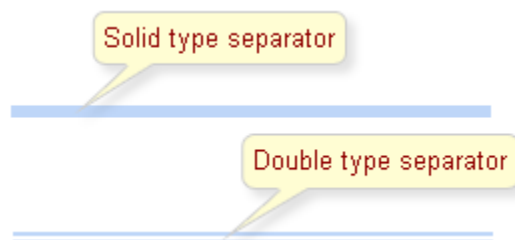


Figure 6.114. Different line types of <rich:separator>

Except style attributes, there are also event definition attributes.

- onmouseover
- onclick
- onmouseout
- etc.

6.67.6. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*.

To redefine the appearance of all `<rich:separator>` components at once, you should add to your style sheets the *style class* used by a `<rich:separator>` component.

6.67.7. Definition of Custom Style Classes

Table 6.310. Classes names that define a component appearance

| Class name | Description |
|----------------|---|
| rich-separator | Defines styles for a component appearance |

In order to redefine styles for all `<rich:separator>` components on a page using CSS, it's enough to create a class with the same name and define necessary properties in it.

To change styles of particular `<rich:separator>` components, define your own style class in the corresponding `<rich:separator>` attributes.

6.67.8. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/separator.jsf?c=separator>] you can see the example of `<rich:separator>` usage and sources for the given example.

6.68. < rich:simpleTogglePanel >

6.68.1. Description

A collapsible panel, which content shows/hides after activating a header control.

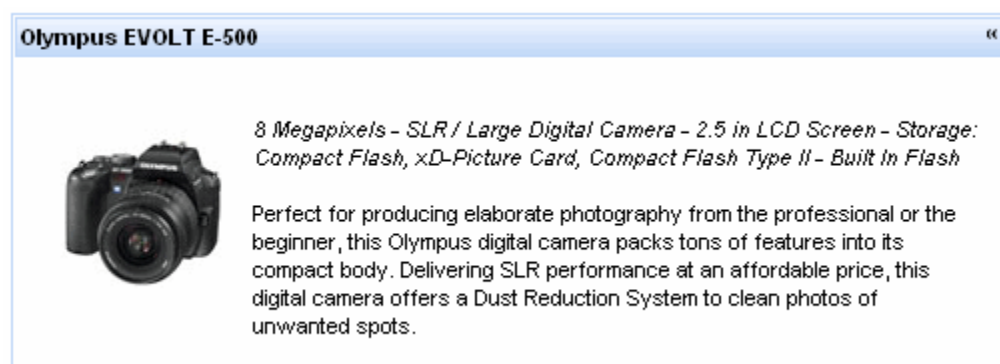


Figure 6.115. <rich:simpleTogglePanel> component

6.68.2. Key Features

- Highly customizable look and feel
- Support for any content inside
- Collapsing expanding content
- Three modes of collapsing/expanding
 - Server
 - Client
 - Ajax

Table 6.311. rich : simpleTogglePanel attributes

| Attribute Name | Description |
|------------------|---|
| action | MethodBinding pointing at the application action to be invoked, if this UIComponent is activated by the user, during the Apply Request Values or Invoke Application phase of the request processing lifecycle, depending on the value of the immediate property |
| actionExpression | The action method binding expression |
| actionListener | MethodBinding pointing at method accepting an ActionEvent with return type void |
| ajaxSingle | if "true", submits ONLY one field/link, instead of all form controls |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| bodyClass | A class that defines a style for a panel content |
| bypassUpdates | If "true", after process validations phase it skips updates of model beans on a force render response. It can be used for validating components input |
| data | Serialized (on default with JSON) data passed on the client by a developer on AJAX request. It's accessible via "data.foo" syntax |
| eventsQueue | Name of requests queue to avoid send next request before complete other from same event. Can be used to reduce number of requests of frequently events (key press, mouse move etc.) |
| focus | |

| Attribute Name | Description |
|--------------------|--|
| | id of element to set focus after request completed on client side |
| headerClass | Class that defines the style for panel header |
| height | Height of a simple toggle panel content area might be defined as pixels or in percents. By default height is not defined |
| id | Every component may have a unique id that is automatically created if omitted |
| ignoreDupResponses | Attribute allows to ignore an Ajax Response produced by a request if the newest 'similar' request is in a queue already. ignoreDupResponses="true" does not cancel the request while it is processed on the server, but just allows to avoid unnecessary updates on the client side if the response isn't actual now |
| immediate | True means, that the default ActionListener should be executed immediately (i.e. during Apply Request Values phase of the request processing lifecycle), rather than waiting until the Invoke Application phase |
| label | Marker to be rendered on a panel header |
| limitToList | If "true", updates on client side ONLY elements from this 'reRender' property. If "false" (default) updates all rendered by ajax region components |
| onbeforedomupdate | JavaScript code for call before DOM has been updated on client side |
| onclick | HTML: a script expression; a pointer button is clicked |
| oncomplete | JavaScript code for call after request completed on client side |
| ondblclick | HTML: a script expression; a pointer button is double-clicked |
| onkeydown | HTML: a script expression; a key is pressed down |
| onkeypress | HTML: a script expression; a key is pressed and released |
| onkeyup | HTML: a script expression; a key is released |
| onmousedown | HTML: script expression; a pointer button is pressed down |

| Attribute Name | Description |
|----------------|---|
| onmousemove | HTML: a script expression; a pointer is moved within |
| onmouseout | HTML: a script expression; a pointer is moved away |
| onmouseover | HTML: a script expression; a pointer is moved onto |
| onmouseup | HTML: script expression; a pointer button is released |
| opened | A false value for this attribute makes a panel closed as default |
| rendered | If "false", this component is not rendered |
| requestDelay | Attribute defines the time (in ms.) that the request will be wait in the queue before it is ready to send. When the delay time is over, the request will be sent to the server or removed if the newest 'similar' request is in a queue already |
| reRender | Id[s] (in format of call <code>UIComponent.findComponent()</code>) of components, rendered in case of AjaxRequest caused by this component. Can be single id, comma-separated list of Id's, or EL Expression with array or Collection |
| status | ID (in format of call <code>UIComponent.findComponent()</code>) of Request status component |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| switchType | Facets switch algorithm: "client", "server"(default), "ajax" |
| timeout | Response waiting time on a particular request. If a response is not received during this time, the request is aborted |
| value | The current value for this component |
| width | Width of a simple toggle panel might be defined as pixels or in percents. By default width is not defined |

Table 6.312. Component identification parameters

| Name | Value |
|-----------------|--|
| component-type | org.richfaces.SimpleTogglePanel |
| component-class | org.richfaces.component.html.HtmlSimpleTogglePanel |

| Name | Value |
|------------------|---|
| component-family | org.richfaces.SimpleTogglePanel |
| renderer-type | org.richfaces.SimpleTogglePanelRenderer |
| tag-class | org.richfaces.taglib.SimpleTogglePanelTag |

6.68.3. Creating the Component with a Page Tag

Here is a simple example as it could be used on a page:

Example:

```
...
<rich:simpleTogglePanel>
    ...
</rich:simpleTogglePanel>
...
```

6.68.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlSimpleTogglePanel;
...
HtmlSimpleTogglePanel myPanel = new HtmlSimpleTogglePanel();
...
```

6.68.5. Details of Usage

The component is a simplified version of toggle panel that initially has a defined layout as a panel with a header playing a role of a mode switching control. On a component header element, it's possible to define a label using an attribute with the same name.

Switching mode could be defined with the *"switchType"* attribute with three possible parameters.

- **Server (DEFAULT)**
The common submission is performed around simpleTogglePanel and a page is completely rendered on a called panel. Only one at a time panel is uploaded onto the client side.
- **Ajax**
AJAX form submission is performed around the panel, content of the called panel is uploaded on Ajax request and additionally specified elements in the *"reRender"* attribute are rendered. Only one at a time panel is uploaded on the client side.
- **Client**
All panels are uploaded on the client side. Switching from the active to the hidden panel is performed with client JavaScript.

The component could also have an *"expanded"* (true/false) attribute responsible for keeping a panel state. It gives an opportunity to manage a simpleTogglePanel state from a model.

- onmouseover
- onclick
- onmouseout
- etc.

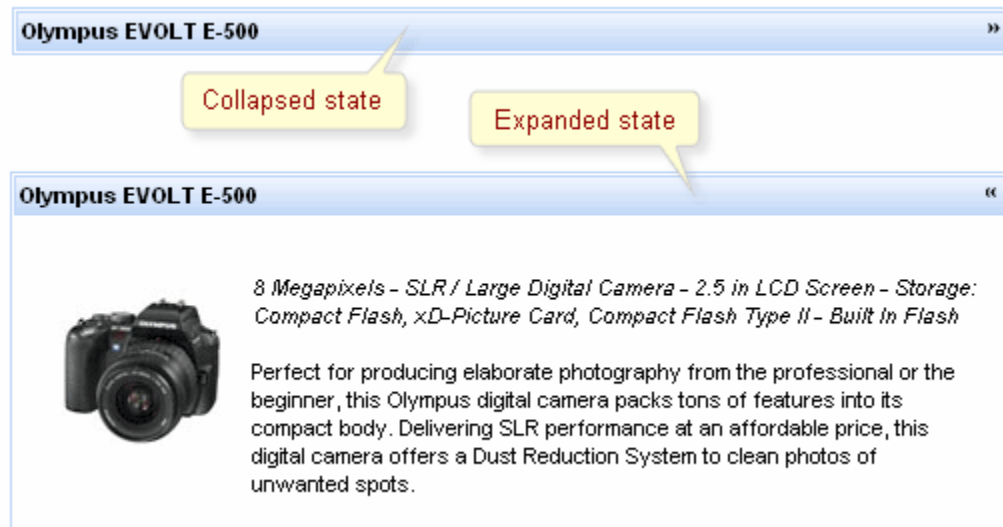


Figure 6.116. `<rich:simpleTogglePanel>` states

6.68.6. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

There are two ways to redefine the appearance of all `<rich:simpleTogglePanel>` components at once:

- Redefine the corresponding skin parameters
- Add to your style sheets *style classes* used by a `<rich:simpleTogglePanel>` component

6.68.7. Skin Parameters Redefinition

Table 6.313. Skin parameters for a whole component

| Skin parameters | CSS properties |
|------------------------|------------------|
| generalBackgroundColor | background-color |
| panelBorderColor | border-color |

Table 6.314. Skin parameters for a header element

| Skin parameters | CSS properties |
|-----------------------|------------------|
| headerBackgroundColor | background-color |

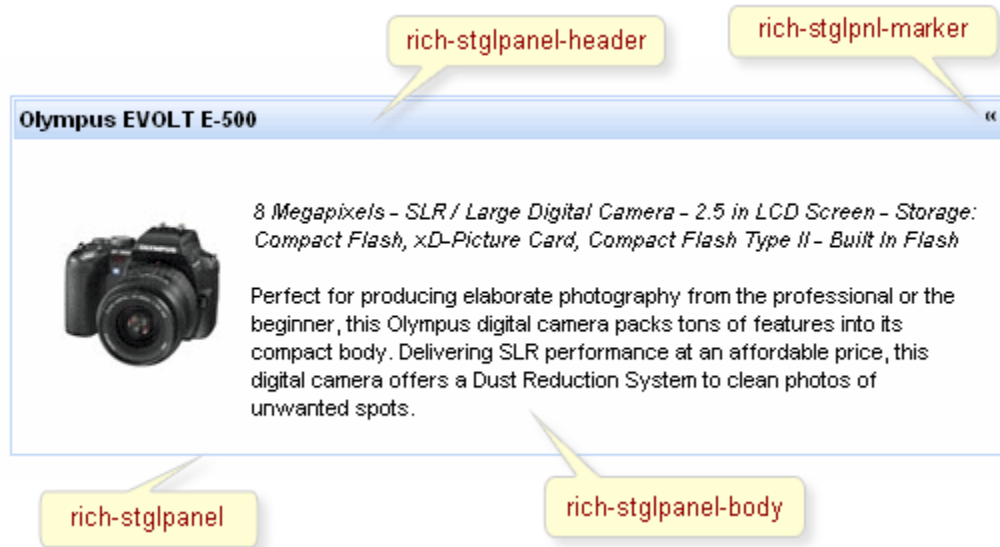
| Skin parameters | CSS properties |
|-----------------------|----------------|
| headerBackgroundColor | border-color |
| headerSizeFont | font-size |
| headTextColor | color |
| headerWeightFont | font-weight |
| headerFamilyFont | font-family |

Table 6.315. Skin parameters for a body element

| Skin parameters | CSS properties |
|------------------------|------------------|
| generalBackgroundColor | background-color |
| generalSizeFont | font-size |
| panelTextColor | color |
| generalFamilyFont | font-family |

6.68.8. Definition of Custom Style Classes

On the screenshot there are classes names that define styles for component elements.

**Figure 6.117. Style classes****Table 6.316. Classes names that define a component appearance**

| Class name | Class description |
|-----------------------|---|
| rich-stglpanel | Defines styles for a wrapper <div> element of a component |
| rich-stglpanel-header | Defines styles for header element of a component |

| Class name | Class description |
|---------------------|--|
| rich-stglpnl-marker | Defines styles for a wrapper <div> element of a marker |
| rich-stglpanel-body | Defines styles for a component content |

Table 6.317. Style component classes

| Class name | Class description |
|-------------|--|
| styleClass | The class defines panel common style. It's used in the outside <div> element |
| bodyClass | applicable to panels body elements |
| headerClass | applicable to header elements |

In order to redefine styles for all `<rich:simpleTogglePanel>` components on a page using CSS, it's enough to create classes with the same names and define necessary properties in them.

To change styles of particular `<rich:simpleTogglePanel>` components, define your own style classes in the corresponding `<rich:simpleTogglePanel>` attributes.

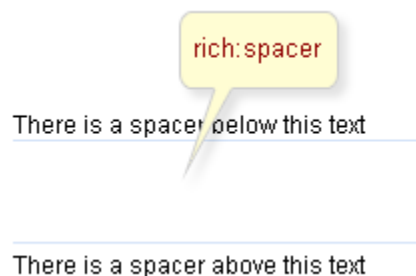
6.68.9. Relevant Resources Links

Here [\[http://livedemo.exadel.com/richfaces-demo/richfaces/simpleTogglePanel.jsf?c=simpleTogglePanel\]](http://livedemo.exadel.com/richfaces-demo/richfaces/simpleTogglePanel.jsf?c=simpleTogglePanel) you can see the example of `<rich:simpleTogglePanel>`s usage and sources for the given example.

6.69. < rich:spacer >

6.69.1. Description

A spacer that is used in layout and rendered as a transparent image.

**Figure 6.118. <rich:spacer> component**

6.69.2. Key Features

- Easily used as a transparent layout spacer
- Horizontal or vertical spacing is managed by an attribute

- Easily customizable sizes parameters

Table 6.318. rich : spacer attributes

| Attribute Name | Description |
|----------------|---|
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| height | The height of the spacer defined in pixels. The default value is 1 pixel |
| id | Every component may have a unique id that is automatically created if omitted |
| onclick | HTML: a script expression; a pointer button is clicked |
| ondblclick | HTML: a script expression; a pointer button is double-clicked |
| onkeydown | HTML: a script expression; a key is pressed down |
| onkeypress | HTML: a script expression; a key is pressed and released |
| onkeyup | HTML: a script expression; a key is released |
| onmousedown | HTML: script expression; a pointer button is pressed down |
| onmousemove | HTML: a script expression; a pointer is moved within |
| onmouseout | HTML: a script expression; a pointer is moved away |
| onmouseover | HTML: a script expression; a pointer is moved onto |
| onmouseup | HTML: script expression; a pointer button is released |
| rendered | If "false", this component is not rendered |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| title | HTML: An advisory title for this element. Often used by the user agent as a tooltip |
| width | The width of the spacer defined in pixels. The default value is 1 pixel |

Table 6.319. Component identification parameters

| Name | Value |
|----------------|----------------------|
| component-type | org.richfaces.spacer |

| Name | Value |
|------------------|---|
| component-class | org.richfaces.component.html.HtmlSpacer |
| component-family | org.richfaces.spacer |
| renderer-type | org.richfaces.SpacerRenderer |
| tag-class | org.richfaces.taglib.SpacerTag |

6.69.3. Creating the Component with a Page Tag

Here is a simple example as it could be used on a page:

Example:

```
...
    <rich:spacer />
...
```

6.69.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlSpacer;
...
HtmlSpacer mySpacer = new HtmlSpacer();
...
```

6.69.5. Details of Usage

<rich:spacer> is a simple layout component which represents a transparent spacer. Thus, the main attributes that define its style are *"style"* and *"styleClass"*.

In addition, the attributes are responsible for the component size: *"width"* and *"height"*.

Moreover, to add e.g. some JavaScript effects, events defined on it are used.

- onmouseover
- onclick
- onmouseout
- etc.

6.69.6. Look-and-Feel Customization

On the component generation, the framework presents a default rich-spacer class in *styleClass* of a generated component, i.e. in order to redefine appearance of all spacers at once, it's necessary to redefine this class in your own CSS (replacing in the result properties defined in a skin with your own).

To define appearance of the particular spacer, it's possible to write your own CSS classes and properties in the component style attributes (*"style"*, *"styleClass"*) modifying component property.

6.69.7. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/spacer.jsf?c=spacer>] you can see the example of `<rich:spacer>` usage and sources for the given example.

6.70. < rich:suggestionbox >

6.70.1. Description

The component adds on-keypress suggestions capabilities to any input text component (like `<h:inputText>`). When a key is pressed in the field Ajax request is sent to the server. When the suggestion action returns a list of possible values, it pop ups them inside the `<div>` element below the input.



Figure 6.119. `<rich:suggestionBox>` component

6.70.2. Key Features

- Fully skinnable component
- Adds *"onkeypress"* suggestions capabilities to any input text component
- Performs suggestion via Ajax requests without any line of JavaScript code written by you
- Possible to render table as a popup suggestion
- Can be pointed to any Ajax request status indicator of the page
- Easily customizable size of suggestion popup
- Setting rules that appear between cells within a table of popup values
- *"Event queue"* and *"request delay"* attributes present to divide frequently requests
- Managing area of components submitted on Ajax request
- Flexible list of components to update after Ajax request managed by attributes

- Setting restriction to Ajax request generation
- Easily setting action to collect suggestion data
- Keyboard navigation support

Table 6.320. rich : suggestionbox attributes

| Attribute Name | Description |
|----------------|--|
| ajaxSingle | if "true", submit ONLY one field/link, instead of all form controls |
| bgcolor | Deprecated. This attribute sets the background color for the document body or table cells. This attribute sets the background color of the canvas for the document body (the BODY element) or for tables (the TABLE, TR, TH, and TD elements). Additional attributes for specifying text color can be used with the BODY element. This attribute has been deprecated in favor of style sheets for specifying background color information |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| border | This attributes specifies the width (in pixels only) of the frame around a table |
| bypassUpdates | If "true", after process validations phase it skips updates of model beans on a force render response. It can be used for validating components input |
| cellpadding | This attribute specifies the amount of space between the border of the cell and its contents. If the value of this attribute is a pixel length, all four margins should be this distance from the contents. If the value of the attribute is percentage length, the top and bottom margins should be equally separated from the content based on percentage of the available vertical space, and the left and right margins should be equally separated from the content based on percentage of the available horizontal space |
| cellspacing | This attribute specifies how much space the user agent should leave between the table and the column on all four sides. The attribute also specifies the amount of space to leave between cells |
| converter | Id of Converter to be used or reference to a Converter |

| Attribute Name | Description |
|----------------|--|
| data | Serialized (on default with JSON) data passed on the client by a developer on AJAX request. It's accessible via "data.foo" syntax |
| dir | Direction indication for text that does not inherit directionality. Valid values are "LTR" (left-to-right) and "RTL" (right-to-left) |
| entryClass | Name of the CSS class for a suggestion entry element (table row) |
| eventsQueue | Name of requests queue to avoid send next request before complete other from same event. Can be used to reduce number of requests of frequently events (key press, mouse move etc.) |
| fetchValue | A value to set in the target input element on a choice suggestion that isn't shown in the suggestion table. It can be used for descriptive output comments or suggestions. If not set, all text in the suggestion row is set as a value |
| first | A zero-relative row number of the first row to display |
| focus | id of element to set focus after request completed on client side |
| for | id (or full path of id's) of target components, for which this element must provide support. If a target component inside of the same <code>NamingContainer</code> (UIForm, UIData in base implementations), can be simple value of the "id" attribute. For other cases must include id's of <code>NamingContainer</code> components, separated by ':'. For search from the root of components, must be started with ': |
| frame | void above below hsides lhs rhs vsides box border [CI] This attribute specifies which sides of the frame surrounding a table will be visible. Possible values: * void: No sides. This is the default value. * above: The top side only. * below: The bottom side only. * hsides: The top and bottom sides only. * vsides: The right and left sides only. * lhs: The left-hand side only. * rhs: The right-hand side only. * box: All four sides. * border: All four sides |

| Attribute Name | Description |
|--------------------|--|
| frequency | Delay (in seconds) before activating the suggestion pop-up |
| height | Height of the pop-up window in pixels |
| id | Every component may have a unique id that is automatically created if omitted |
| ignoreDupResponses | Attribute allows to ignore an Ajax Response produced by a request if the newest 'similar' request is in a queue already. ignoreDupResponses="true" does not cancel the request while it is processed on the server, but just allows to avoid unnecessary updates on the client side if the response isn't actual now |
| immediate | A flag indicating that this component value must be converted and validated immediately (that is, during Apply Request Values phase), rather than waiting until a Process Validations phase. |
| lang | Code describing the language used in the generated markup for this component |
| limitToList | If "true", updates on client side ONLY elements from this 'reRender' property. If "false" (default) updates all rendered by ajax region components |
| minChars | Minimal number of chars in input to activate suggestion pop-up |
| nothingLabel | "nothingLabel" is inserted to popup list if the autocomplete returns empty list. It isn't selectable and list is closed as always after click on it and nothing is put to input. |
| onbeforedomupdate | JavaScript code for call before DOM has been updated on client side |
| oncomplete | JavaScript code for call after request completed on client side |
| onselect | JavaScript code for call on select suggestion, after update value of target element |
| onsubmit | JavaScript code for call before submission of ajax event |
| param | Name the HTTP request parameter with the value of input element token. If not set, it be will sent as an |

| Attribute Name | Description |
|----------------|--|
| | input element name. In this case, input will perform validation and update the value |
| popupClass | HTML CSS class attribute of element for pop-up suggestion content |
| popupStyle | HTML CSS style attribute of element for pop-up suggestion content |
| rendered | If "false", this component is not rendered |
| requestDelay | Attribute defines the time (in ms.) that the request will be wait in the queue before it is ready to send. When the delay time is over, the request will be sent to the server or removed if the newest 'similar' request is in a queue already |
| reRender | Id[s] (in format of call <code>UIComponent.findComponent()</code>) of components, rendered in case of <code>AjaxRequest</code> caused by this component. Can be single id, comma-separated list of Id's, or EL Expression with array or Collection |
| rowClasses | A comma-delimited list of CSS style classes that is applied to popup table rows. A space separated list of classes may also be specified for any individual row. The styles are applied, in turn, to each row in the table. For example, if the list has two elements, the first style class in the list is applied to the first row, the second to the second row, the first to the third row, the second to the fourth row, etc. In other words, we keep iterating through the list until we reach the end, and then we start at the beginning again |
| rowNumber | rowNumber |
| rules | This attribute specifies which rules will appear between cells within a table. The rendering of rules is user agent dependent. Possible values: * none: No rules. This is the default value. * groups: Rules will appear between row groups (see <code>THEAD</code> , <code>TFOOT</code> , and <code>TBODY</code>) and column groups (see <code>COLGROUP</code> and <code>COL</code>) only. * rows: Rules will appear between rows only. * cols: Rules will appear between columns only. * all: Rules will appear between all rows and columns |
| selectedClass | Name of the CSS class for a selected suggestion entry element (table row) |

| Attribute Name | Description |
|------------------|--|
| selectValueClass | Name of the CSS class for a selected suggestion entry element (table cell) |
| selfRendered | If "true", forces active Ajax region render response directly from stored components tree, bypasses page processing. Can be used for increase performance. Also, must be set to 'true' inside iteration components, such as dataTable. |
| shadowDepth | Pop-up shadow depth for suggestion content |
| shadowOpacity | Attribute defines shadow opacity for suggestion content |
| status | ID (in format of call <code>UIComponent.findComponent()</code>) of Request status component |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| suggestionAction | Method calls an expression to get a collection of suggestion data on request. It must have one parameter with a type of Object with content of input component and must return any type allowed for <code><h:dataTable></code> |
| summary | This attribute provides a summary of the table's purpose and structure for user agents rendering to non-visual media such as speech and Braille |
| timeout | Response waiting time on a particular request. If a response is not received during this time, the request is aborted |
| title | Advisory title information about markup elements generated for this component |
| tokens | The list (or single value) of symbols which can be used for division chosen of suggestion pop-up values in a target element. After input of a symbol from the list suggestion pop-up it is caused again |
| upValue | upValue |
| value | The initial value to set when rendered for the first time |
| var | A request-scope attribute via which the data object for the current row will be used when iterating |

| Attribute Name | Description |
|----------------|--|
| width | Width of the pop-up window in pixels |
| zindex | Attribute is similar to the standard HTML attribute and can specify window placement relative to the content |

Table 6.321. Component identification parameters

| Name | Value |
|------------------|--|
| component-type | org.richfaces.SuggestionBox |
| component-class | org.richfaces.component.html.HtmlSuggestionBox |
| component-family | org.richfaces.SuggestionBox |
| renderer-type | org.richfaces.SuggestionBoxRenderer |
| tag-class | org.richfaces.taglib.SuggestionBoxTag |

6.70.3. Creating the Component with a Page Tag

Here is a simple example as it could be used on a page:

Example:

```
...
<h:inputText value="#{bean.property}" id="suggest"/>
<rich:suggestionBox for="suggest" suggestionAction="#{bean.autocomplete}"
    var="suggest">
    <h:column>
        <h:outputText value="#{suggest.text}"/>
    </h:column>
</rich:suggestionbBox>
...
```

Here is the *bean.autocomplete* method that returns the collection to pop up:

Example:

```
public List autocomplete(Object event) {
    String pref = event.toString();
    //collecting some data that begins with "pref" letters.
    ...
    return result;
}
```

6.70.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlSuggestionBox;
...
```

```
HtmlSuggestionBox myList = new HtmlSuggestionBox();
...
```

6.70.5. Details of Usage

As it is shown in the example above, the main component attribute are:

- *"for"*

The attribute where there is an input component which activation causes a suggestion activation

- *"suggestionAction"*

is an accepting parameter of a suggestionEvent type that returns as a result a collection for rendering in a tool tip window.

- *"var"*

a collection name that provides access for inputting into a table in a popup

There are also two size attributes (*"width"* and *"height"*) that are obligatory for the suggestion component. The attributes have initial Defaults but should be specified manually in order to be changed.

The suggestionBox component, as it is shown on the screenshot, could get any collection for an output and outputs it in a ToolTip window the same as a custom dataTable (in several columns)

```
...
<rich:suggestionBox for="test"
                    suggestionAction="#{bean.autocomplete}" var="cit"
fetchValue="#{cit.text}">
  <h:column>
    <h:outputText value="#{cit.label}" />
  </h:column>
  <h:column>
    <h:outputText value="#{cit.text}" />
  </h:column>
</rich:suggestionBox>
...
```

It looks on a page in the following way:



Figure 6.120. <rich:suggestionBox> with ToolTip window

When some string is chosen input receives the corresponding value from the second column containing `#{cit.text}`

There is also one more important attribute named *"tokens"* that specifies separators after which a set of some characters sequence is defined as a new prefix beginning from this separator and not from the string beginning.

Example:

```
...
<rich:suggestionBox for="test" suggestionAction="#{bean.autocomplete}" var="cit"
                    selfRendered="true" tokens=",">
    <h:column>
        <h:outputText value="#{cit.text}"/>
    </h:column>
</rich:suggestionBox>
...
```

This example shows that when a city is chosen and a comma and first letter character are input, Ajax request is called again, but it submits a value starting from the last token:

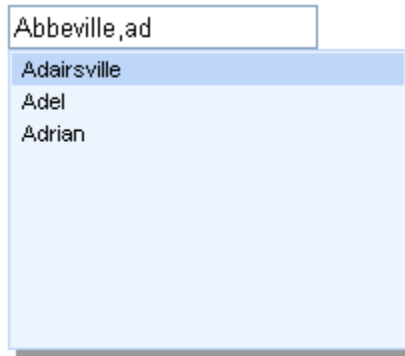


Figure 6.121. <rich:suggestionBox> with chosen word

For a multiple definition use either *".,;"* syntax as a value for tokens or link a parameter to some bean property transmitting separators collection.

The component also encompasses *"layout"* and *"style"* attributes corresponding to dataTable ones for a table appearing in popup (for additional information, read JSF Reference) and custom attribute managing AJAX requests sending (for additional information, see Ajax4JSF Project [<http://www.jboss.org/projects/jbossajax4jsf>]).

In addition to these attributes common for Ajax action components and limiting requests quantity and frequency, suggestionBox has one more its own attribute limiting requests (the *"minChars"* attribute). The attribute defines characters quantity inputted into a field after which Ajax requests are called to perform suggestion.

There is possibility to define what be shown if the autocomplete returns empty list. Attribute *"nothingLabel"* or facet with the same name could be used for it.

Example:

```
...
<rich:suggestionBox nothingLabel="Empty" for="test"
suggestionAction="#{bean.autocomplete}" var="cit">
  <h:column>
    <h:outputText value="#{cit.text}"/>
  </h:column>
</rich:suggestionBox>
...
```

Example:

```
...
<rich:suggestioBox for="test" suggestionAction="#{bean.autocomplete}" var="cit">
  <facet name="nothingLabel">
    <h:outputText value="Empty"/>
  </facet>
  <h:column>
    <h:outputText value="#{cit.text}"/>
  </h:column>
</rich:suggestionBox>
...
```

It looks on a page in the following way:

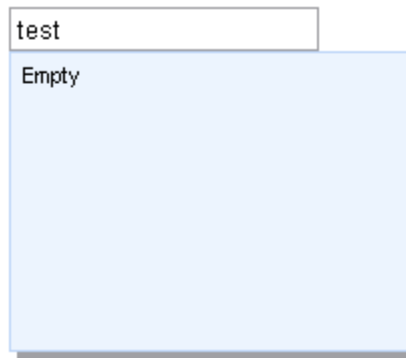


Figure 6.122. `<rich:suggestionBox>` with empty list

6.70.7. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

There are two ways to redefine the appearance of all `<rich:suggestionBox>` components at once:

- Redefine the corresponding skin parameters
- Add to your style sheets *style classes* used by a `<rich:suggestionBox>` component

6.70.8. Skin Parameters Redefinition

Table 6.322. General skin parameters redefinition for popup list

| Parameters for popup list | CSS properties |
|---------------------------|------------------|
| additionalBackgroundColor | background-color |
| panelBorderColor | border-color |

Table 6.323. Skin parameters redefinition for shadow element of the list

| Parameters for shadow element of the list | CSS properties |
|---|------------------|
| shadowBackgroundColor | background-color |
| shadowBorderColor | border-color |
| shadowOpacity | opacity |

Table 6.324. Skin parameters redefinition for popup table rows

| Parameters for popup table rows | CSS properties |
|---------------------------------|----------------|
| generalSizeFont | font-size |
| generalTextColor | color |
| generalFamilyFont | font-family |

Table 6.325. Skin parameters redefinition for selected row

| Parameters for selected row | CSS properties |
|-----------------------------|------------------|
| headerBackgroundColor | background-color |
| generalSizeFont | font-size |
| generalFamilyFont | font-family |
| headerTextColor | color |

6.70.9. Definition of Custom Style Classes

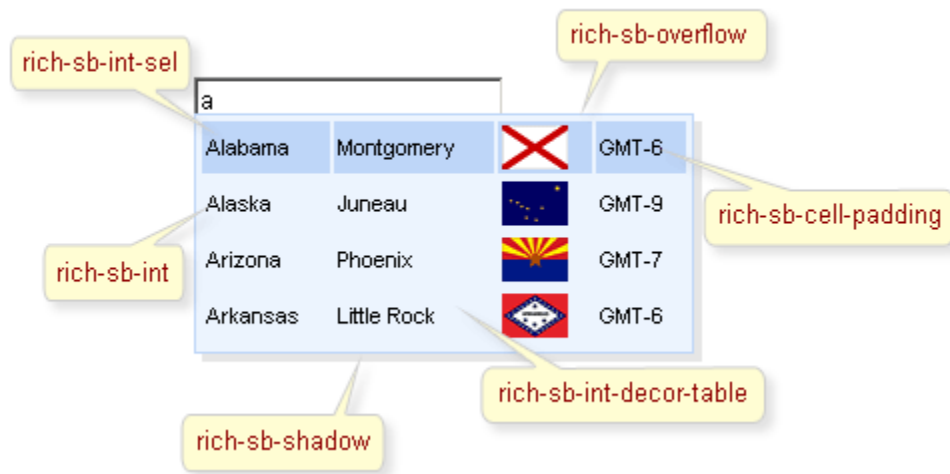


Figure 6.123. Classes names

On the screenshot, there are classes names defining specified elements.

Table 6.326. Classes names that define a suggestionBox

| Class name | Description |
|--------------------------|--|
| rich-sb-common-container | Defines styles for a wrapper <div> element of a suggestion container |
| rich-sb-ext-decor-1 | Defines styles for the first wrapper <div> element of a suggestion box exterior |
| rich-sb-ext-decor-2 | Defines styles for the second wrapper <div> element of a suggestion box exterior |
| rich-sb-ext-decor-3 | Defines styles for the third wrapper <div> element of a suggestion box exterior |
| rich-sb-overflow | Defines styles for a wrapper <div> element |
| rich-sb-int-decor-table | Defines styles for a suggestion box table |
| rich-sb-int | Defines the styles for a suggestion box table rows (tr) |
| rich-sb-cell-padding | Defines the styles for suggestion box table cells (td) |
| rich-sb-int-sel | Defines styles for a selected row |
| rich-sb-shadow | Defines styles for a suggestion box shadow |

In order to redefine styles for all **<rich:suggestionBox>** components on a page using CSS, it's enough to create classes with the same names and define necessary properties in them.

To change styles of particular **<rich:suggestionBox>** components, define your own style classes in the corresponding **<rich:suggestionBox>** attributes.

6.70.10. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/suggestionBox.jsf?c=suggestionBox>] you can see the example of `<rich:suggestionBox>` usage and sources for the given example.

6.71. < rich:tabPanel >

6.71.1. Description

A tab panel displaying tabs for grouping content of the panel.

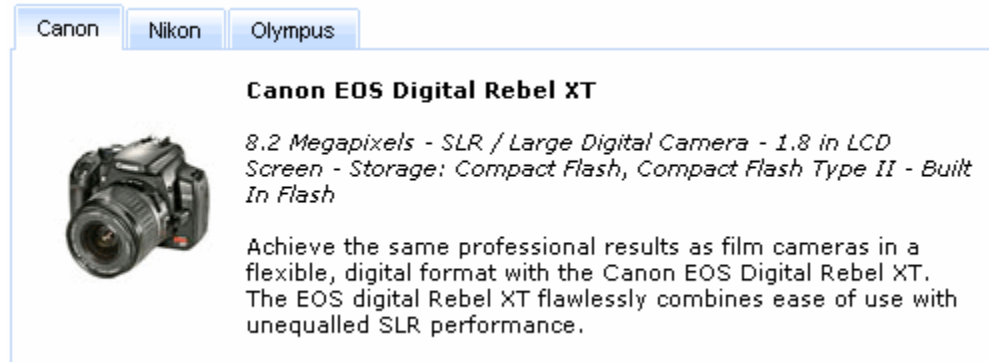


Figure 6.124. `<rich:tabPanel>` component

6.71.2. Key Features

- Skinnable tab panel and child items
- Disabled/enabled tab options
- Customizable headers
- Group any content inside a tab
- Each tab has a unique name for direct access (e.g. for switching between tabs)
- Switch methods can be easily customized with attribute to:
 - Server
 - Client
 - AJAX
- Switch methods can be selected for the whole tab panel and for the each tab separately

Table 6.327. `rich : tabPanel` attributes

| Attribute Name | Description |
|-----------------------------|--|
| <code>activeTabClass</code> | A CSS class to be applied to an active tab |

| Attribute Name | Description |
|------------------|---|
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| contentClass | A CSS class for content of a tab panel |
| contentStyle | A CSS style is for the content of a tab panel |
| converter | Id of Converter to be used or reference to a Converter |
| converterMessage | A ValueExpression enabled attribute that, if present, will be used as the text of the converter message, replacing any message that comes from the converter |
| dir | Direction indication for text that does not inherit directionality. Valid values are "LTR" (left-to-right) and "RTL" (right-to-left) |
| disabledTabClass | A CSS class to be applied to a disabled tab |
| headerAlignment | Sets tab headers alignment. It can be "left" or "right". "left" is used by default |
| headerClass | A CSS style is for the header of a tab panel. |
| headerSpacing | Sets tab headers spacing. It should be a valid size unit expression |
| height | Height of a tab panel defined in pixels or in percents |
| id | Every component may have a unique id that is automatically created if omitted |
| immediate | A flag indicating that this component value must be converted and validated immediately (that is, during Apply Request Values phase), rather than waiting until a Process Validations phase |
| inactiveTabClass | CSS class to be applied to an inactive (but not disabled) tab |
| label | A localized user presentable name for this component. |
| lang | Code describing the language used in the generated markup for this component |
| onclick | HTML: a script expression; a pointer button is clicked |
| ondblclick | HTML: a script expression; a pointer button is double-clicked |
| onkeydown | HTML: a script expression; a key is pressed down |

| Attribute Name | Description |
|---------------------|--|
| onkeypress | HTML: a script expression; a key is pressed and released |
| onkeyup | HTML: a script expression; a key is released |
| onmousedown | HTML: script expression; a pointer button is pressed down |
| onmousemove | HTML: a script expression; a pointer is moved within |
| onmouseout | HTML: a script expression; a pointer is moved away |
| onmouseover | HTML: a script expression; a pointer is moved onto |
| onmouseup | HTML: script expression; a pointer button is released |
| rendered | If "false", this component is not rendered |
| required | If "true", this component is checked for non-empty input |
| requiredMessage | A ValueExpression enabled attribute that, if present, will be used as the text of the validation message for the "required" facility, if the "required" facility is used |
| selectedTab | Attribute defines name of selected tab |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| switchType | Tab switch algorithm: "client", "server"(default), "ajax" |
| tabClass | A CSS class to be applied to all tabs |
| title | Advisory title information about markup elements generated for this component |
| validator | MethodBinding pointing at a method that is called during Process Validations phase of the request processing lifecycle, to validate the current value of this component |
| validatorMessage | A ValueExpression enabled attribute that, if present, will be used as the text of the validator message, replacing any message that comes from the validator |
| value | The initial value to set when rendered for the first time |
| valueChangeListener | Listener for value changes |

| Attribute Name | Description |
|----------------|--|
| width | Width of a tab panel defined in pixels or in percents. The default value is 100% |

Table 6.328. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | org.richfaces.tabPanel |
| component-class | org.richfaces.component.html.HtmltabPanel |
| component-family | org.richfaces.tabPanel |
| renderer-type | org.richfaces.tabPanelRenderer |
| tag-class | org.richfaces.taglib.tabPanelTag |

6.71.3. Creating the Component with a Page Tag

Here is a simple example as it could be used on a page:

Example:

```
...
    <rich:tabPanel>
        <!--//Set of Tabs inside-->
        <rich:tab>
            ...
        </rich:tab>
    </rich:tabPanel>
...
```

6.71.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmltabPanel;
...
HtmltabPanel mytabPanel = new HtmltabPanel();
...
```

6.71.5. Details of Usage

As it was mentioned above, tabPanel groups content on panels and performs switching from one to another. Hence, modes of switching between panels are described first of all.

Note:

All tabPanels should be wrapped into a form element so as content is correctly submitted inside. If a form is placed into each tab, the Action elements of Tab controls appear to be out of the form and content submission inside the panels could be performed only for Action components inside tabs.

Switching mode could be chosen with the `tabPanel` attribute *"mode"* with three possible parameters.

- Server (DEFAULT)

The common submission is performed around `tabPanel` and a page is completely rendered on a called panel. Only one at a time `tabPanel` is uploaded onto the client side.

- Ajax

AJAX form submission is performed around the `tabPanel`, content of the called `tabPanel` is uploaded on Ajax request and additionally specified elements in the *"reRender"* attribute are rendered. Only one at a time `tabPanel` is uploaded on the client.

- Client

All `tabPanels` are uploaded on the client side. The switching from the active to the hidden panel is performed with client JavaScript.

As a result, the `tabPanel` is switched to the second tab according to the action returning outcome for moving onto another page and switching from the second to the first tab is performed.

There is also the *"selectedTab"* attribute. The attribute keeps an active tab name; therefore, an active `tabPanel` could be changed with setting a name of the necessary tab to this attribute.

There is also the *"headerAlignment"* attribute responsible for rendering of `tabPanel` components. The attribute has several values: left (Default), right, center, which specify Tabs components location on the top of the `tabPanel`.

Example:

```
...
    <rich:tabPanel width="40%" headerAlignment="right">
        <rich:tab label="Canon">
            ...
        </rich:tab>
        <rich:tab label="Nikon">
            ...
        </rich:tab>
        <rich:tab label="Olympus">
            ...
        </rich:tab>
    </rich:tabPanel>
...
```

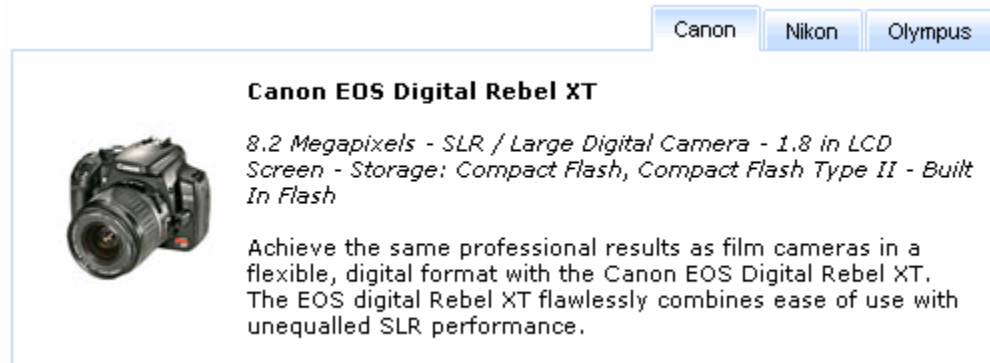


Figure 6.125. <rich:tabPanel> with right aligned tabs

Except the specific attributes, the component has all necessary attributes for JavaScript events definition.

- onmouseover
- onmouseout
- etc.

6.71.6. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

There are two ways to redefine the appearance of all **<rich:tabPanel>** components at once:

- Redefine the corresponding skin parameters
- Add to your style sheets *style classes* used by a **<rich:tabPanel>** component

6.71.7. Skin Parameters Redefinition

Table 6.329. Skin parameters redefinition for a header

| Skin parameters | CSS properties |
|------------------|------------------|
| panelBorderColor | border-top-color |

Table 6.330. Skin parameters redefinition for an internal content

| Skin parameters | CSS properties |
|------------------------|---------------------|
| generalBackgroundColor | background-color |
| generalTextColor | color |
| panelBorderColor | border-bottom-color |
| panelBorderColor | border-right-color |
| panelBorderColor | border-left-color |

| Skin parameters | CSS properties |
|-------------------|----------------|
| generalSizeFont | font-size |
| generalFamilyFont | font-family |

6.71.8. Definition of Custom Style Classes

On the screenshot there are classes names that define styles for component elements.

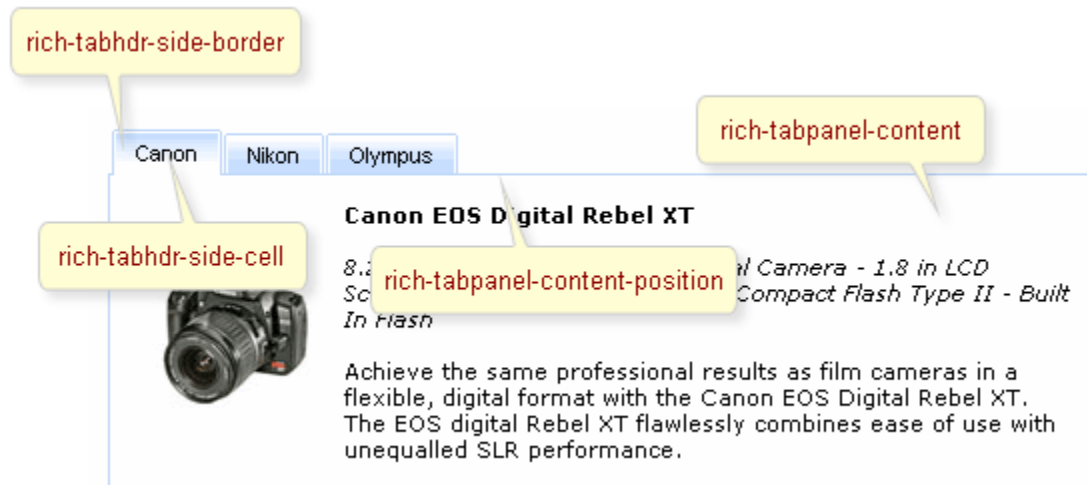


Figure 6.126. Style classes

Table 6.331. Classes names that define a component appearance

| Class name | Description |
|--------------------------------|--|
| rich-tabpanel | Defines styles for all tabPanel |
| rich-tabpanel-content | Defines styles for an internal content |
| rich-tabpanel-content-position | Defines styles for a wrapper element of a tabPanel content. It should define a shift equal to borders width in order to overlap panel tabs |
| rich-tabhdr-side-border | Defines styles for side elements of a tabPanel header |
| rich-tabhdr-side-cell | Defines styles for a header internal element |

Table 6.332. Classes names that define different tab header states (corresponds to rich-tabhdr-side-cell)

| Class name | Description |
|---------------------------|---|
| rich-tabhdr-cell-active | Defines styles for an internal element of an active header |
| rich-tabhdr-cell-inactive | Defines styles for an internal element of an inactive label |

| Class name | Description |
|---------------------------|--|
| rich-tabhdr-cell-disabled | Defines styles for an internal element of a disabled label |

In order to redefine styles for all `<rich:tabPanel>` components on a page using CSS, it's enough to create classes with the same names and define necessary properties in them.

To change styles of particular `<rich:tabPanel>` components, define your own style classes in the corresponding `<rich:tabPanel>` attributes.

6.71.9. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/tabPanel.jsf?c=tabPanel>] you can see the example of `<rich:tabPanel>` usage and sources for the given example.

6.72. < rich:tab >

6.72.1. Description

A tab section within a tab panel.

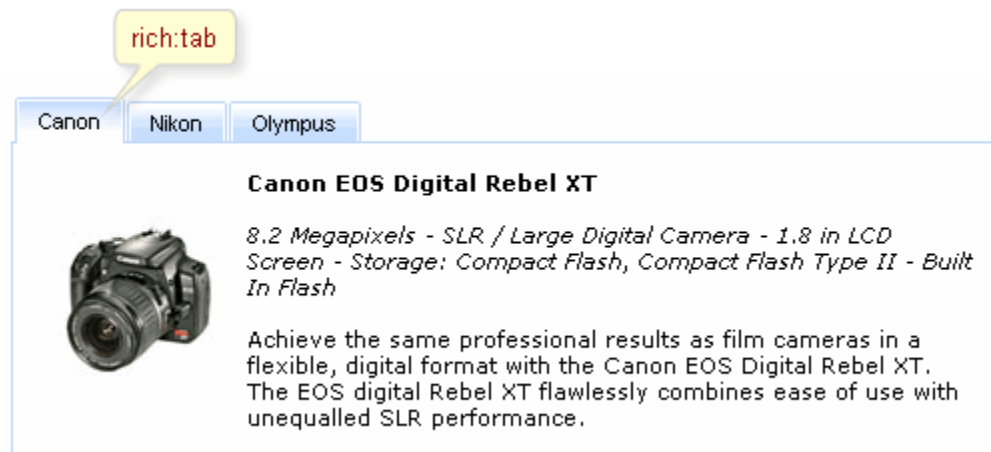


Figure 6.127. <rich:tab> component

6.72.2. Key Features

- Fully skinnable tabs content
- Disabled/enabled tab options
- Groups any content inside a tab
- Each tab has a unique name for a direct access (e.g. for switching between tabs)
- Switch methods can be easily customized for every tab separately with attribute to:
 - Server

- Client
- AJAX

Table 6.333. rich : tab attributes

| Attribute Name | Description |
|--------------------|---|
| action | MethodBinding pointing at the application action to be invoked, if this UIComponent is activated by the user, during the Apply Request Values or Invoke Application phase of the request processing lifecycle, depending on the value of the immediate property |
| actionExpression | The action method binding expression |
| actionListener | MethodBinding pointing at method accepting an ActionEvent with return type void |
| ajaxSingle | if "true", submits ONLY one field/link, instead of all form controls |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| bypassUpdates | If "true", after process validations phase it skips updates of model beans on a force render response. It can be used for validating components input |
| data | Serialized (on default with JSON) data passed on the client by a developer on AJAX request. It's accessible via "data.foo" syntax |
| disabled | Disables a tab in a tab panel |
| eventsQueue | Name of requests queue to avoid send next request before complete other from same event. Can be used to reduce number of requests of frequently events (key press, mouse move etc.) |
| focus | id of element to set focus after request completed on client side |
| id | Every component may have a unique id that is automatically created if omitted |
| ignoreDupResponses | Attribute allows to ignore an Ajax Response produced by a request if the newest 'similar' request is in a queue already. ignoreDupResponses="true" does not cancel the request while it is processed on the server, but just |

| Attribute Name | Description |
|-------------------|---|
| | allows to avoid unnecessary updates on the client side if the response isn't actual now |
| immediate | True means, that the default ActionListener should be executed immediately (i.e. during Apply Request Values phase of the request processing lifecycle), rather than waiting until the Invoke Application phase |
| label | Text for the actual "tab" in a tab section |
| labelWidth | Length for the actual "tab" in a tab section defined in pixels. If it is not defined, the length is calculated basing on a tab label text length |
| limitToList | If "true", updates on client side ONLY elements from this 'reRender' property. If "false" (default) updates all rendered by ajax region components |
| name | Attribute defines tab name |
| onbeforedomupdate | JavaScript code for call before DOM has been updated on client side |
| onclick | HTML: a script expression; a pointer button is clicked |
| oncomplete | JavaScript code for call after request completed on client side |
| ondblclick | HTML: a script expression; a pointer button is double-clicked |
| onkeydown | HTML: a script expression; a key is pressed down |
| onkeypress | HTML: a script expression; a key is pressed and released |
| onkeyup | HTML: a script expression; a key is released |
| onmousedown | HTML: script expression; a pointer button is pressed down |
| onmousemove | HTML: a script expression; a pointer is moved within |
| onmouseout | HTML: a script expression; a pointer is moved away |
| onmouseover | HTML: a script expression; a pointer is moved onto |
| onmouseup | HTML: script expression; a pointer button is released |
| ontabenter | Event must occurs on the tab which has been entered |
| ontableave | Event must occurs on the tab which has been left |

| Attribute Name | Description |
|----------------|---|
| rendered | If "false", this component is not rendered |
| requestDelay | Attribute defines the time (in ms.) that the request will be wait in the queue before it is ready to send. When the delay time is over, the request will be sent to the server or removed if the newest 'similar' request is in a queue already |
| reRender | Id[s] (in format of call <code>UIComponent.findComponent()</code>) of components, rendered in case of <code>AjaxRequest</code> caused by this component. Can be single id, comma-separated list of Id's, or EL Expression with array or Collection |
| status | ID (in format of call <code>UIComponent.findComponent()</code>) of Request status component |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| switchType | Tab switch algorithm: "client", "server", "ajax", "page" |
| timeout | Response waiting time on a particular request. If a response is not received during this time, the request is aborted |
| title | HTML: An advisory title for this element. Often displayed as a tooltip |

Table 6.334. Component identification parameters

| Name | Value |
|------------------|--------------------------------------|
| component-type | org.richfaces.Tab |
| component-class | org.richfaces.component.html.HtmlTab |
| component-family | org.richfaces.Tab |
| renderer-type | org.richfaces.TabRenderer |
| tag-class | org.richfaces.taglib.TabTag |

6.72.3. Creating the Component with a Page Tag

Here is a simple example as it could be used on a page:

Example:

```
...
```

```

<rich:tabPanel>
  <!--Set of Tabs inside-->
  <rich:tab>
    ...
  </rich:tab>
</rich:tabPanel>
...

```

6.72.4. Creating the Component Dynamically Using Java

Example:

```

import org.richfaces.component.html.HtmlTab;
...
HtmlTab myTab = new HtmlTab();
...

```

6.72.5. Details of Usage

The main component function is to define a content group that is rendered and processed when the tab is active, i.e. click on a tab causes switching onto a tab containing content corresponded to this tab.

The *"label"* attribute defines text to be represented. If you can use the *"label"* facet, you can even not use the *"label"* attribute.

Example:

```

...
<rich:tab>
  <f:facet name="label">
    <h:graphicImage value="/images/img1.png"/>
  </f:facet>
  ...
  <!--Any Content inside-->
  ...
</rich:tab>
...

```

A marker on a tab header defined with the *"label"* attribute. Moreover, each tab could be disabled (switching on this tab is impossible) with the *"disable"* attribute.

Example:

```

...
<rich:tabPanel width="20%">
  <rich:tab label="Tab">
    <h:outputText value="Active Tab content"/>
  </rich:tab>
  <rich:tab label="Disabled Tab" disabled="true">
    ...
  </rich:tab>
  <rich:tab label="Next Enabled Tab">
    ...
  </rich:tab>

```

```

</rich:tabPanel>
...

```

With this example it's possible to generate the tab panel with the second disabled and two active tabs (see the picture).

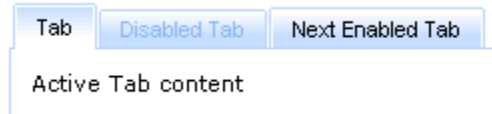


Figure 6.128. `<rich:tabPanel>` with disabled `<rich:tab>`

Switching mode could be defined not only for the whole panel tab, but also for each particular tab, i.e. switching onto one tab could be performed right on the client with the corresponding JavaScript and onto another tab with an Ajax request on the server. Tab switching modes are the same as tabPanel ones.

Each tab also has an attribute name (alias for `"id"` attribute). Using this attribute value it's possible e.g. to set an active tab on a model level specifying this name in the corresponding attribute of the whole tab.

Except the specific component attributes it has all necessary attributes for JavaScript event definition.

- onmouseover
- onmouseout
- etc.

Some event could be performed on the tab which has been entered/left using `"ontabenter"/"ontableave"` attributes. See the example below.

Example:

```

...
<rich:tabPanel>
  <rich:tab label="Tab1" ontabenter="alert()">
    ...
  </rich:tab>
  ...
</rich:tabPanel>
...

```

The following example shows how on the client side to get the names of entered/left tabs.

```
ontabenter="alert(leftTabName)"
```

6.72.6. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

Note:

A panel appearance and content is defined with a tab panel i.e. on the tab level it's possible to define only an appearance of this tab header.

There are two ways to redefine the appearance of all `<rich:tab>` components at once:

- Redefine the corresponding skin parameters
- Add to your style sheets *style classes* used by a `<rich:tab>` component

6.72.7. Skin Parameters Redefinition

Table 6.335. Skin parameters redefinition for a tab header

| Skin parameters | CSS properties |
|-------------------|----------------|
| generalTextColor | color |
| generalSizeFont | font-size |
| generalFamilyFont | font-family |

Table 6.336. Skin parameters redefinition for an active tab

| Skin parameters | CSS properties |
|------------------------|------------------|
| generalTextColor | color |
| subBorderColor | border-color |
| generalBackgroundColor | background-color |

Table 6.337. Skin parameters redefinition for an inactive tab

| Skin parameters | CSS properties |
|--------------------|------------------|
| tabBackgroundColor | background-color |
| subBorderColor | border-color |

Table 6.338. Skin parameters redefinition for a disabled tab

| Skin parameters | CSS properties |
|----------------------|------------------|
| tabBackgroundColor | background-color |
| subBorderColor | border-color |
| tabDisabledTextColor | color |

6.72.8. Definition of Custom Style Classes



Figure 6.129. Classes names

Table 6.339. Classes names that define a tab

| Class name | Description |
|-----------------|---------------------------------|
| rich-tab-header | Defines styles for a tab header |
| rich-tab-label | Defines styles for a tab label |

Table 6.340. Classes names that define a tab states

| Class name | Description |
|-------------------|------------------------------------|
| rich-tab-active | Defines styles for an active tab |
| rich-tab-inactive | Defines styles for an inactive tab |
| rich-tab-disabled | Defines styles for a disabled tab |

In order to redefine styles for all **<rich:tab>** components on a page using CSS, it's enough to create classes with the same names and define necessary properties in them.

To change styles of particular **<rich:tab>** components define your own style classes in the corresponding **<rich:tab>** attribute.

6.73. < rich:togglePanel >

6.73.1. Description

A wrapper component with named facets, where every facet is shown after activation of the corresponding toggleControl (the other is hidden).



Figure 6.130. <rich:togglePanel> component

6.73.2. Key Features

- Support for any content inside
- Three modes of facets switching
 - Server
 - Client
 - Ajax
- Controls for togglePanel can be everywhere in layout

Table 6.341. rich : togglePanel attributes

| Attribute Name | Description |
|------------------|--|
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| converter | Id of Converter to be used or reference to a Converter |
| converterMessage | A ValueExpression enabled attribute that, if present, will be used as the text of the converter message, replacing any message that comes from the converter |
| id | Every component may have a unique id that is automatically created if omitted |
| immediate | A flag indicating that this component value must be converted and validated immediately (that is, during |

| Attribute Name | Description |
|-----------------|--|
| | Apply Request Values phase), rather than waiting until a Process Validations phase |
| initialState | It contains a name of the first active facet |
| label | A localized user presentable name for this component. |
| onclick | HTML: a script expression; a pointer button is clicked |
| ondblclick | HTML: a script expression; a pointer button is double-clicked |
| onkeydown | HTML: a script expression; a key is pressed down |
| onkeypress | HTML: a script expression; a key is pressed and released |
| onkeyup | HTML: a script expression; a key is released |
| onmousedown | HTML: script expression; a pointer button is pressed down |
| onmousemove | HTML: a script expression; a pointer is moved within |
| onmouseout | HTML: a script expression; a pointer is moved away |
| onmouseover | HTML: a script expression; a pointer is moved onto |
| onmouseup | HTML: script expression; a pointer button is released |
| rendered | If "false", this component is not rendered |
| required | If "true", this component is checked for non-empty input |
| requiredMessage | A ValueExpression enabled attribute that, if present, will be used as the text of the validation message for the "required" facility, if the "required" facility is used |
| stateOrder | Names of the facets in the switching order. If ToggleControl doesn't contain information about a next facet to be shown it is switched corresponding to this attribute |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| switchType | Facets switch algorithm: "client", "server"(default), "ajax". |

| Attribute Name | Description |
|---------------------|---|
| validator | MethodBinding pointing at a method that is called during Process Validations phase of the request processing lifecycle, to validate the current value of this component |
| validatorMessage | A ValueExpression enabled attribute that, if present, will be used as the text of the validator message, replacing any message that comes from the validator |
| value | The initial value to set when rendered for the first time |
| valueChangeListener | Listener for value changes |

Table 6.342. Component identification parameters

| Name | Value |
|------------------|--|
| component-type | org.richfaces.TogglePanel |
| component-class | org.richfaces.component.html.HtmlTogglePanel |
| component-family | org.richfaces.TogglePanel |
| renderer-type | org.richfaces.TogglePanelRenderer |
| tag-class | org.richfaces.Taglib.togglePanelTag |

6.73.3. Creating the Component with a Page Tag

Here is a simple example as it could be used in a page:

Example:

```
...
<rich:togglePanel>
  <f:facet name="first">
    ...
  </f:facet>
  <f:facet name="second">
    ...
  </f:facet>
  ...
</rich:togglePanel>
...
<!--//Set of the toggleControls somewhere on a page.-->
...
```

6.73.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmltogglePanel;
```

```
...
HtmltogglePanel myPanel = new HtmltogglePanel();
...
```

6.73.5. Details of Usage

As it was mentioned above, `togglePanel` splits content into named facets that become rendered and processed when a click performed on controls linked to this `togglePanel` (either switched on the client or send requests on the server for switching).

The initial component state is defined with `"initialState"` attribute, where a facet name that is shown at first is defined.

Note:

It's also possible to define an "empty" facet to implement the functionality as drop-down panels have and make the facet active when no content is required to be rendered.

Switching mode could be defined with the `"switchType"` attribute with three possible parameters:

- **Server (DEFAULT)**
The common submission is performed around `togglePanel` and a page is completely rendered on a called panel. Only one at a time the panel is uploaded onto the client side.
- **Ajax**
AJAX form submission is performed around the panel, content of the called panel is uploaded on an Ajax request and additionally specified elements in the `"reRender"` attribute are rendered. Only one at a time the panel is uploaded on the client side.
- **Client**
All panels are uploaded on the client side. The switching from the active to the hidden panel is performed with client JavaScript.

"Facets" switching order could be defined on the side of `<rich:toggleControl>` component or on the panel. On the side of the `togglePanel` it's possible to define facets switching order with the `"stateOrder"` attribute. The facets names are enumerated in such an order that they are rendered when a control is clicked, as it's not defined where to switch beforehand.

Example:

```
...
<rich:togglePanel id="panel" initialState="panelB" switchType="client"
                  stateOrder="panelA,panelB,panelC">
  <f:facet name="panelA">
    ...
  </f:facet>
  <f:facet name="panelB">
    ...
  </f:facet>
  <f:facet name="panelC">
    ...
  </f:facet>
</rich:togglePanel>
```

```
<rich:toggleControl for="panel" value="Switch"/>
...
```

The example shows a togglePanel initial state when the second facet (panelB) is rendered and successive switching from the first to the second happens.

6.73.6. Look-and-Feel Customization

The component doesn't have its own representation rendering only content of its facets, thus all look and feel is set only for content.

6.73.7. Definition of Custom Style Classes

Table 6.343. Classes names that define a component appearance

| Class name | Description |
|-------------------|-------------------------------------|
| rich-toggle-panel | Defines styles for all component |
| rich-tglctrl | Defines styles for a toggle control |

6.73.8. Relevant Resources Links

Here [http://livedemo.exadel.com/richfaces-demo/richfaces/togglePanel.jsf?c=togglePanel] you can see the example of **<rich:togglePanel>** usage and sources for the given example.

6.74. < rich:toggleControl >

6.74.1. Description

A link type control for switching between togglePanel facets. Target Panel is specified with *"for"* attribute. It can be located inside or outside the togglePanel. As the result of switching between facets previous facet is hidden and another one (specified with *"switchToState"* or panel *"stateOrder"* attributes) is shown.



Figure 6.131. <rich:toggleControl> component

6.74.2. Key Features

- Highly customizable look and feel
- Can be located anywhere in a page layout
- Switching is provided in the three modes
 - Server
 - Client
 - Ajax

Table 6.344. rich : toggleControl attributes

| Attribute Name | Description |
|------------------|---|
| accesskey | Access key that, when pressed, transfers focus to this element |
| action | MethodBinding pointing at the application action to be invoked, if this UIComponent is activated by the user, during the Apply Request Values or Invoke Application phase of the request processing lifecycle, depending on the value of the immediate property |
| actionExpression | The action method binding expression |
| actionListener | MethodBinding pointing at method accepting an ActionEvent with return type void |
| ajaxSingle | if "true", submit ONLY one field/link, instead of all form controls |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| bypassUpdates | If "true", after process validations phase skip updates of model beans and force render response. Can be used for validate components input |
| data | Serialized (on default with JSON) data passed on the client by a developer on AJAX request. It's accessible via "data.foo" syntax |
| dir | Direction indication for text that does not inherit directionality. Valid values are "LTR" (left-to-right) and "RTL" (right-to-left) |
| eventsQueue | Name of requests queue to avoid send next request before complete other from same event. Can be used |

| Attribute Name | Description |
|--------------------|---|
| | to reduce number of requests of frequently events (key press, mouse move, etc.) |
| focus | id of element to set focus after request completed on client side |
| for | String containing comma separated ids (in the format of a <code>UIComponent.findComponent()</code> call) of the target components. |
| id | Every component may have a unique id that is automatically created if omitted |
| ignoreDupResponses | Attribute allows to ignore an Ajax Response produced by a request if the newest 'similar' request is in a queue already. <code>ignoreDupResponses="true"</code> does not cancel the request while it is processed on the server, but just allows to avoid unnecessary updates on the client side if the response isn't actual now |
| immediate | True means, that the default <code>ActionListener</code> should be executed immediately (i.e. during Apply Request Values phase of the request processing lifecycle), rather than waiting until the Invoke Application phase |
| lang | Code describing the language used in the generated markup for this component |
| limitToList | If "true", updates on client side ONLY elements from this 'reRender' property. if "false" (default) updates all rendered by ajax region components |
| onbeforedomupdate | JavaScript code for call before DOM has been updated on client side |
| onblur | JavaScript code executed when this element loses focus |
| onclick | JavaScript code executed when a pointer button is clicked over this element |
| oncomplete | JavaScript code for call after request completed on client side |
| ondblclick | JavaScript code executed when a pointer button is double clicked over this element |
| onfocus | JavaScript code executed when this element receives focus |

| Attribute Name | Description |
|----------------|---|
| onkeydown | JavaScript code executed when a key is pressed down over this element |
| onkeypress | JavaScript code executed when a key is pressed and released over this element |
| onkeyup | JavaScript code executed when a key is released over this element |
| onmousedown | JavaScript code executed when a pointer button is pressed down over this element |
| onmousemove | JavaScript code executed when a pointer button is moved within this element |
| onmouseout | JavaScript code executed when a pointer button is moved away from this element |
| onmouseover | JavaScript code executed when a pointer button is moved onto this element |
| onmouseup | JavaScript code executed when a pointer button is released over this element |
| panelId | Attribute defines Id for corresponding panel |
| rendered | If "false", this component is not rendered |
| requestDelay | Attribute defines the time (in ms.) that the request will be wait in the queue before it is ready to send. When the delay time is over, the request will be sent to the server or removed if the newest 'similar' request is in a queue already |
| status | ID (in format of call <code>UIComponent.findComponent()</code>) of Request status component |
| style | CSS style(s) to be applied when this component is rendered |
| styleClass | Space-separated list of CSS style class(es) to be applied when this element is rendered. This value must be passed through as the "class" attribute on generated markup |
| switchToState | Contains one of the facets names where target <code>togglePanel</code> is switched to |

| Attribute Name | Description |
|----------------|---|
| tabindex | Position of this element in the tabbing order for the current document. This value must be an integer between 0 and 32767 |
| timeout | Response waiting time on a particular request. If a response is not received during this time, the request is aborted |
| title | Advisory title information about markup elements generated for this component |
| value | Initial value to set when rendered for the first time |

Table 6.345. Component identification parameters

| Name | Value |
|------------------|--|
| component-type | org.richfaces.ToggleControl |
| component-class | org.richfaces.component.html.HtmlToggleControl |
| component-family | org.richfaces.ToggleControl |
| renderer-type | org.richfaces.ToggleControlRenderer |
| tag-class | org.richfaces.taglib.ToggleControlTag |

6.74.3. Creating the Component with a Page Tag

Here is a simple example as it could be used on a page:

Example:

```
...
<rich:toggleControl for="panel"/>
    ...
    <rich:togglePanel id="panel" stateOrder="[facets order to be switched]">
        <!--//Set of Facets-->
    </rich:togglePanel>
...
```

6.74.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlToggleControl;
...
HtmlToggleControl myControl = new HtmlToggleControl();
...
```

6.74.5. Details of Usage

As it was mentioned above, the control could be in any place in layout and linked to a switching panel that is managed with *"for"* attribute (in the *"for"* attribute the full component *"id"* is specified according to naming containers).

The togglePanel could be also switched from the side of the control instead of being strictly defined in *"switchOrder"* attribute of **<rich:togglePanel>**.

Example:

```
...
<rich:togglePanel id="panel" initialState="empty" switchType="client">
  <f:facet name="first">
    <h:panelGroup>
      <rich:toggleControl for="helloForm:panel" value="Empty "
switchToState="empty"/>
      <rich:toggleControl for="helloForm:panel" value=" Second"
switchToState="second"/>
      ...//Some Content
    </h:panelGroup>
  </f:facet>
<f:facet name="second">
  <h:panelGroup>
    <rich:toggleControl for="helloForm:panel" value="Empty "
switchToState="empty"/>
    <rich:toggleControl for="helloForm:panel" value=" first"
switchToState="first"/>
    ...//Some Content
  </h:panelGroup>
</f:facet>
<f:facet name="empty">
  <h:panelGroup>
    <rich:toggleControl for="helloForm:panel" value="first "
switchToState="first"/>
    <rich:toggleControl for="helloForm:panel" value=" second"
switchToState="second"/>
  </h:panelGroup>
</f:facet>
</rich:togglePanel>
...
```

In this example the switching is performed on facets specified in the *"switchToState"* attribute.

6.74.6. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*.

To redefine the appearance of all **<rich:toggleControl>** components at once, you should add to your style sheets *style class* used by a **<rich:toggleControl>** component.

6.74.7. Definition of Custom Style Classes

Table 6.346. Classes names that define a component appearance

| Class name | Description |
|--------------|-------------------------------------|
| rich-tglctrl | Defines styles for a toggle control |

In order to redefine styles for all `<rich:toggleControl>` components on a page using CSS, it's enough to create a class with the same name and define necessary properties in it.

To change styles of particular `<rich:toggleControl>` components define your own style class in the corresponding `<rich:toggleControl>` attributes.

6.75. < rich:toolBar >

6.75.1. Description

A horizontal bar with Action items on it that accepts any JSF components as children.

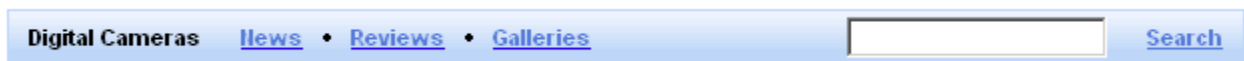


Figure 6.132. <rich:toolBar> with action items

6.75.2. Key Features

- Skinnable menu panel and child items
- Standard top menu bar that can be used in accordance with a menu component
- Grouping bar content
- Easily place content on any side of a menu bar using predefined group layout
- Predefined separators for menu items and groups
- Any content inside

Table 6.347. rich : toolBar attributes

| Attribute Name | Description |
|----------------|---|
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| contentClass | A CSS style is to be applied to each element of tool bar content. Use this style, for example, to setup parameters of the font. |
| contentStyle | A CSS style is to be applied to each element of tool bar content. |

| Attribute Name | Description |
|-----------------|---|
| height | A height of a bar in pixels. If a height is not defined, a bar height depends of the "headerFontSize" skin parameter. |
| id | Every component may have a unique id that is automatically created if omitted |
| itemSeparator | A separator between items on a bar. Possible values are none, line, square, disc and grid. |
| onitemclick | HTML: a script expression; a pointer button is clicked on an item |
| onitemdblclick | HTML: a script expression; a pointer button is double-clicked on an item |
| onitemkeydown | HTML: a script expression; a key is pressed down on an item |
| onitemkeypress | HTML: a script expression; a key is pressed and released on an item |
| onitemkeyup | HTML: a script expression; a key is released on an item |
| onitemmousedown | |
| onitemmousemove | HTML: a script expression; a pointer is moved on an item |
| onitemmouseout | HTML: a script expression; a pointer is moved away from an item |
| onitemmouseover | HTML: a script expression; a pointer is moved onto an item |
| onitemmouseup | HTML: script expression; a pointer button is released on an item |
| rendered | If "false", this component is not rendered |
| separatorClass | A CSS class to be applied to tool bar separators. |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| width | A width of a bar that can be defined in pixels or as percentage. The default value is 100%. |

Table 6.348. Component identification parameters

| Name | Value |
|------------------|--|
| component-type | org.richfaces.ToolBar |
| component-class | org.richfaces.component.html.HtmlToolBar |
| component-family | org.richfaces.ToolBar |
| renderer-type | org.richfaces.ToolBarRenderer |
| tag-class | org.richfaces.taglib.ToolBarTag |

6.75.3. Creating the Component with a Page Tag

Here is a simple example as it could be used on a page:

Example:

```
...
<rich:toolbar>
    <!--//...Set of action or other JSF components-->
</rich:toolbar>
...
```

6.75.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlToolBar;
...
HtmlToolBar myToolBar = new HtmlToolBar();
...
```

6.75.5. Details of Usage

A toolbar is a wrapper component that facilitates creation of menu and tool bars. All components defined inside are located on a stylized bar with possibility to group, arrange on the both bar sides, and place predefined separators between them.

Grouping and an input side definition is described for toolbarGroup that defines this functionality.

Separators are located between components with the help of the *"itemSeparator"* attribute with four predefined values:

- none
- line
- square
- disc

For example, when setting a separator of a disc type, the following result is produced:



Figure 6.133. `<rich:toolBar>` with a "disc" separator

Moreover, for toolbar style "width" and "height" attributes are placed above all.

6.75.6. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

There are two ways to redefine the appearance of all `<rich:toolBar>` components at once:

- Redefine the corresponding skin parameters
- Add to your style sheets *style classes* used by a `<rich:toolBar>` component

6.75.7. Skin Parameters Redefinition

Table 6.349. Skin parameters redefinition for a component exterior

| Skin parameters | CSS properties |
|-----------------------|------------------|
| panelBorderColor | border-color |
| headerBackgroundColor | background-color |

Table 6.350. Skin parameters redefinition for a component item

| Skin parameters | CSS properties |
|------------------|----------------|
| headerSizeFont | font-size |
| headerTextColor | color |
| headerWeightFont | font-weight |
| headerFamilyFont | font-family |

6.75.8. Definition of Custom Style Classes

Table 6.351. Classes names that define a component appearance

| Class name | Description |
|-------------------|--------------------------------------|
| rich-toolbar | Defines styles for a toolbar element |
| rich-toolbar-item | Defines styles for a toolbar item |

In order to redefine styles for all `<rich:toolBar>` components on a page using CSS, it's enough to create class with the same names and define necessary properties in them.

To change styles of particular `<rich:toolBar>` components define your own style classes in the corresponding `<rich:toolBar>` attributes.

The component also has the standard attributes `"style"` and `"styleClass"` that could redefine an appearance of a particular component variants.

6.75.9. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/toolBar.jsf?c=toolBar>] you can see the example of `<rich:toolBar>` usage and sources for the given example.

6.76. < rich:toolBarGroup >

6.76.1. Description

A group of items inside a tool bar.

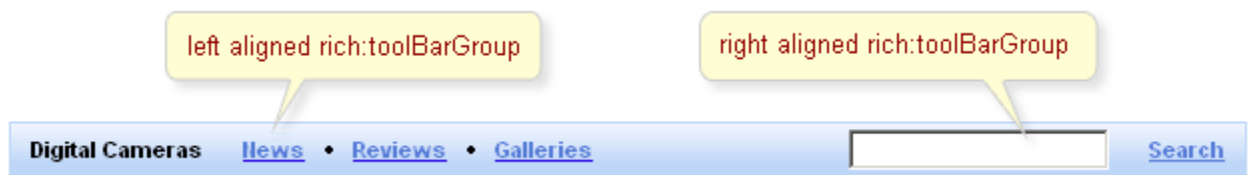


Figure 6.134. `<rich:toolBarGroup>` with items on it

6.76.2. Key Features

- Fully skinnable with its child items
- Grouping bar content
- Easily place content on either side of tool bar using a predefined group layout
- Predefined separators for menu items and groups
- Any content inside

Table 6.352. `rich : toolBarGroup` attributes

| Attribute Name | Description |
|----------------|--|
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| id | Every component may have a unique id that is automatically created if omitted |
| itemSeparator | "A separator for the items in a group. Possible values are "none", "line", "square", "disc" and "grid"." |

| Attribute Name | Description |
|----------------|--|
| location | "A location of a group on a tool bar. Possible values are "left" and "right"." |
| onclick | HTML: a script expression; a pointer button is clicked |
| ondblclick | HTML: a script expression; a pointer button is double-clicked |
| onkeydown | HTML: a script expression; a key is pressed down |
| onkeypress | HTML: a script expression; a key is pressed and released |
| onkeyup | HTML: a script expression; a key is released |
| onmousedown | HTML: script expression; a pointer button is pressed down |
| onmousemove | HTML: a script expression; a pointer is moved within |
| onmouseout | HTML: a script expression; a pointer is moved away |
| onmouseover | HTML: a script expression; a pointer is moved onto |
| onmouseup | HTML: script expression; a pointer button is released |
| rendered | If "false", this component is not rendered |
| separatorClass | "A CSS class to be applied to tool bar group separators." |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |

Table 6.353. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | org.richfaces.ToolBarGroup |
| component-class | org.richfaces.component.html.HtmlToolBarGroup |
| component-family | org.richfaces.ToolBarGroup |
| renderer-type | org.richfaces.ToolBarGroupRenderer |
| tag-class | org.richfaces.taglib.ToolBarGroupTag |

6.76.4. Creating the Component with a Page Tag

Here is a simple example as it could be used on a page:

Example:

```

...
<rich:toolBar>
    ...
    <rich:toolBarGroup>
        <!--...Set of action or other JSF components-->
    </rich:toolBarGroup>
    <rich:toolBarGroup>
        <!--...Set of action or other JSF components-->
    </rich:toolBarGroup>
    ...
</rich:toolBar>
...

```

6.76.5. Creating the Component Dynamically Using Java**Example:**

```

import org.richfaces.component.html.HtmlToolBarGroup;
...
HtmlToolBarGroup myToolBarGroup = new HtmlToolBarGroup();
...

```

6.76.6. Details of Usage

A `toolBarGroup` is a wrapper component that groups `toolBar` content and facilitates creation of menu and tool bars. All components defined inside are located on a stylized bar with a possibility to group, arrange on the both bar sides, and place predefined separators between them.

Separators are located between components with the help of the *"itemSeparator"* attribute with four predefined values:

- none
- line
- square
- disc

To control the group location inside, use the *"location"* attribute with left (DEFAULT) and right values.

Example:

```

...
<rich:toolBar itemSeparator="disc" width="500">
    <rich:toolBarGroup itemSeparator="line">
        <h:commandLink value="Command 1.1"/>
        <h:commandLink value="Command 2.1"/>
    </rich:toolBarGroup>
    <rich:toolBarGroup itemSeparator="line" location="right">
        <h:commandLink value="Command 1.2"/>
        <h:commandLink value="Command 2.2"/>
    </rich:toolBarGroup>
</rich:toolBar>

```

```

    </rich:toolBarGroup>
  </rich:toolBar>
  ...

```

The code result is the following:

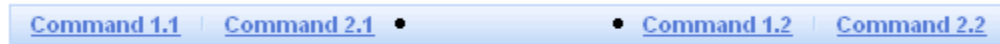


Figure 6.135. Stylized `<rich:toolBarGroup>`

6.76.7. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

There are two ways to redefine the appearance of all `<rich:toolBarGroup>` components at once:

- Redefine the corresponding skin parameters
- Add to your style sheets *style classes* used by a `<rich:toolBarGroup>` component

6.76.8. Skin Parameters Redefinition

Table 6.354. Skin parameters redefinition

| Skin parameters | CSS properties |
|------------------|----------------|
| headerSizeFont | font-size |
| headTextColor | color |
| headerFamilyFont | font-family |

6.77. `< rich:toolTip >`

6.77.1. Description

The component used for creation non-modal popup that activated on some event and display some information.



Figure 6.136. `<rich:toolTip>` component

6.77.2. Key Features

- Highly customizable look and feel

- Different ways of data loading to tooltip
- Disablement support

Table 6.355. rich : tooltip attributes

| Attribute Name | Description |
|------------------|--|
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| direction | Defines direction of the popup list to appear (top-right, top-left bottom-right, bottom-left, auto(default)) |
| disabled | If false the components is rendered on the client but Js for calling disabled. |
| event | event that triggers the tooltip appearance (default = onmouseover) |
| followMouse | If 'true' tooltip should follow the mouse while it moves over the parent element |
| hideDelay | Delay in milliseconds before tooltip will be hidden. |
| horizontalOffset | Sets the horizontal offset between popup list and mouse pointer |
| id | Every component may have a unique id that is automatically created if omitted |
| layout | Allowed values: "inline" or "block". Block/inline mode flag. Tooltip will contain div/span elements accordingly. |
| mode | controls the way of data loading to tooltip and should have following values client (default), ajax |
| onclick | HTML: a script expression; a pointer button is clicked |
| oncomplete | JavaScript code for call after the tooltip shown |
| ondblclick | HTML: a script expression; a pointer button is double-clicked |
| onhide | JavaScript code for call after the tooltip hidden |
| onkeydown | HTML: a script expression; a key is pressed down |
| onkeypress | HTML: a script expression; a key is pressed and released |
| onkeyup | HTML: a script expression; a key is released |
| onmousedown | |

| Attribute Name | Description |
|----------------|---|
| | HTML: script expression; a pointer button is pressed down |
| onmousemove | HTML: a script expression; a pointer is moved within |
| onmouseout | HTML: a script expression; a pointer is moved away |
| onmouseover | HTML: a script expression; a pointer is moved onto |
| onmouseup | HTML: script expression; a pointer button is released |
| onshow | JavaScript code for call after the tooltip called (some element overed) but before its requesting |
| rendered | If "false", this component is not rendered |
| showDelay | Delay in milliseconds before tooltip will be displayed. |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| value | Label on the tooltip |
| verticalOffset | Sets the vertical offset between popup list and mouse pointer |
| zorder | The same as CSS z-index for toolTip. |

Table 6.356. Component identification parameters

| Name | Value |
|------------------|--|
| component-type | org.richfaces.component.toolTip |
| component-class | org.richfaces.component.html.HtmltoolTip |
| component-family | org.richfaces.component.toolTip |
| renderer-type | org.richfaces.renderkit.html.toolTipRenderer |
| tag-class | org.richfaces.taglib.HtmltoolTipTag |

6.77.3. Creating the Component with a Page Tag

To create the simplest variant of toolTip on a page, use the following syntax:

Example:

```
...
<h:commandButton value="Button">
  <rich:toolTip value="toolTip content"/>
</h:commandButton>
```

...

6.77.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmltoolTip;
...
HtmltoolTip mytoolTip = new HtmltoolTip();
...
```

6.77.5. Details of Usage

toolTip main area is a simple rectangle area with any information inside. The content may be defined via "value" attribute as text or via any nested content. When both are defined, the value is displayed as text and nested content after the text. toolTip stretches if the content more than the toolTip sizes.

There is possibility to define a facet with the name "defaultContent". This facet provides the default content to display while the main content is loaded to a page in an Ajax mode. Thus when toolTip called in an Ajax mode, it appears with the content defined in the facet and when loading is completed, the content is changed to a loaded one.

Here is an example:

Example:

```
...
<h:commandLink value="Simple Link">
  <rich:toolTip followMouse="true" direction="top-right" mode="ajax"
    value="#{bean.toolTipContent}" horizontalOffset="5"
    verticalOffset="5" layout="block">
    <f:facet name="defaultContent">
      <f:verbatim>DEFAULT toolTip CONTENT</f:verbatim>
    </f:facet>
  </rich:toolTip>
</h:commandLink>
...
```

This is the result:



Figure 6.137. <rich:toolTip> component with default content

And after toolTip loaded it will be changed to next one:



Figure 6.138. <rich:toolTip> component with loaded content

By default, `toolTip` appears smart positioned. But as you can see from the previous example, you can define an appearance direction via the corresponding attribute `"direction"`. And also it's possible to define vertical and horizontal offsets relatively to a mouse position.

`toolTip` appears attached to the corner dependent on the `"direction"` attribute. By default it is positioned bottom-right. `toolTip` activation occurs after a defined event (default=`mouseover`) on the parent component takes into consideration the `"delay"` attribute (default=`0`) or after calling JS API function `Show()`. `toolTip` deactivation occurs after `mouseout` event on the parent component (excepting the situation when the mouse is hovered onto the `toolTip` itself) or after calling JS API function `Hide()`.

Note:

It is recommended to define parent component `"id"` for correction of `toolTip` work.

The attribute `"for"` is used for defining the `"id"` of an element a `toolTip` should be attached to. Look at the example:

Example:

```
...
<div id="elementId">
    <rich:toolTip for="elementId">Using a toolTip</rich:toolTip>
    <p>The first simple example</p>
</div>
...
<div id="elementId">
    <p>The second simple example</p>
</div>
<rich:toolTip for="elementId">Using a toolTip</rich:toolTip>
...
```

Here, the attribute `"for"` of a `<rich:toolTip>` component is required. Without it an example doesn't work because HTML elements aren't presented in component tree built by facelets.

The `"mode"` attribute is provided you to control the way of data loading to `toolTip`. It has following values:

- Client
- Ajax

In a client mode, `toolTip` content is rendered once on the server and could be reRendered only via external submit. In an Ajax mode, `toolTip` content is requested from server every activation.

Disabled `toolTip` is rendered to a page as usual but JS that responds for its activation is disabled until `Enable()` is called.

Moreover, to add some JavaScript effects, client events defined on it are used:

Standart:

- onclick
- ondblclick

- onmouseout
- onmousemove
- onmouseover

Special:

- onshow - Called after the toolTip is called (some element hovered) but before its request
- oncomplete - Called just after the toolTip is shown
- onhide - Called after the toolTip is hidden

6.77.6. JavaScript API

Table 6.357. JavaScript API

| Function | Description |
|-----------|------------------------------------|
| Show() | Shows the corresponding toolTip |
| Hide() | Hides the corresponding toolTip |
| Enable() | Enables the corresponding toolTip |
| Disable() | Disables the corresponding toolTip |

6.77.7. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

There are two ways to redefine the appearance of all `<rich:toolTip>` components at once:

- Redefine the corresponding skin parameters
- Add to your style sheets *style classes* used by a `<rich:toolTip>` component

6.77.8. Skin Parameters Redefinition

Table 6.358. Skin parameters redefinition for a component

| Skin parameters | CSS properties |
|--------------------|------------------|
| tipBackgroundColor | background-color |
| tipBorderColor | border-color |
| generalSizeFont | font-size |
| generalFamilyFont | font-family |
| generalFontColor | color |

6.77.9. Definition of Custom Style Classes

Table 6.359. Classes names that define a component appearance

| Class name | Description |
|---------------|---|
| rich-tool-tip | Defines styles for a wrapper <code></code> or <code><div></code> element of a toolTip |

It depends on `<rich:toolTip>` layout what a wrapper element `` or `<div>` to choose.

In order to redefine styles for all `<rich:toolTip>` components on a page using CSS, it's enough to create class with the same name and define necessary properties in it.

To change styles of particular `<rich:toolTip>` components define your own style class in the corresponding `<rich:toolTip>` attributes

6.77.10. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/toolTip.jsf?c=toolTip>] you can see the example of `<rich:toolTip>` usage and sources for the given example.

6.78. < rich:tree >

6.78.1. Description

A component for a tree-like presentation of data. It includes built-in drag and drop support for its child elements.

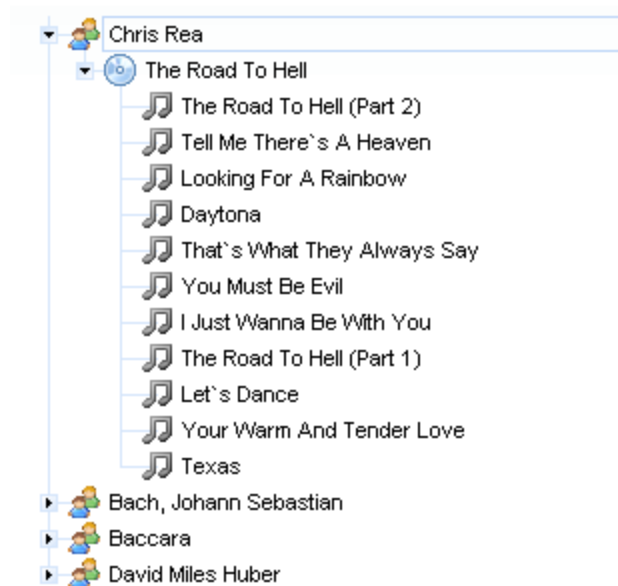


Figure 6.139. Expanded `<rich:tree>` with child elements

6.78.2. Key Features

- Highly customizable look-and-feel

This feature provides rich possibilities to change an appearance of a component into different styles.

- Built-in drag and drop support

This feature allows programming DnD possibility for the `<rich:tree>` component so that it would be possible to change a placement of nodes in a tree simply by dropping a certain node from one place to another. Look at the screenshot below:

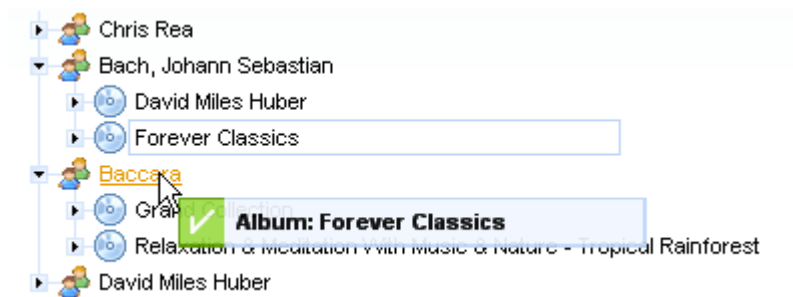


Figure 6.140. Drag and drop support

Here, an appropriate CD-disk is been moving from one album to the next one.

- Built-in Ajax processing
- Possibility to define a visual representation by node type
- Support of several root elements in a tree

Table 6.360. rich : tree attributes

| Attribute Name | Description |
|--------------------|---|
| acceptedTypes | List of drag types to be processed by the current drop zone |
| adviseNodeOpened | MethodBinding pointing at a method accepting an <code>org.richfaces.component.UITree</code> with return of <code>java.lang.Boolean</code> type. If returned value is: <code>java.lang.Boolean.TRUE</code> , a particular <code>treeNode</code> is expanded; <code>java.lang.Boolean.FALSE</code> , a particular <code>treeNode</code> is collapsed; null, a particular <code>treeNode</code> saves the current state |
| adviseNodeSelected | MethodBinding pointing at a method accepting an <code>org.richfaces.component.UITree</code> with return of <code>java.lang.Boolean</code> type. If returned value is: <code>java.lang.Boolean.TRUE</code> , a particular <code>treeNode</code> is selected; <code>java.lang.Boolean.FALSE</code> , a particular <code>treeNode</code> is unselected; null, a particular <code>treeNode</code> saves the current state |

| Attribute Name | Description |
|----------------------|---|
| ajaxSingle | if "true", submits ONLY one field/link, instead of all form controls |
| ajaxSubmitSelection | If "true", an Ajax request to be submit when selecting node |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| bypassUpdates | If "true", after process validations phase it skips updates of model beans on a force render response. It can be used for validating components input |
| changeExpandListener | Listener called on expand/collapse event on the node |
| componentState | It defines EL-binding for a component state for saving or redefinition |
| data | Serialized (on default with JSON) data passed on the client by a developer on AJAX request. It's accessible via "data.foo" syntax |
| dragIndicator | An indicator component id |
| dragListener | MethodBinding representing an action listener method that will be notified after drag operation |
| dragType | Key of a drag object. It's used to define a necessity of processing the current dragged element on the drop zone side |
| dragValue | Data to be sent to the drop zone after a drop event |
| dropListener | MethodBinding representing an action listener method that will be notified after drop operation |
| dropValue | Data to be processed after a drop event |
| eventsQueue | Name of requests queue to avoid send next request before complete other from same event. Can be used to reduce number of requests of frequently events (key press, mouse move etc.) |
| focus | id of element to set focus after request completed on client side |
| highlightedClass | Corresponds to the HTML class attribute. Applied to highlighted node |
| icon | The icon for node |

| Attribute Name | Description |
|--------------------|--|
| iconCollapsed | The icon for collapsed node |
| iconExpanded | The icon for expanded node |
| iconLeaf | An icon for component leaves |
| id | Every component may have a unique id that is automatically created if omitted |
| ignoreDupResponses | Attribute allows to ignore an Ajax Response produced by a request if the newest 'similar' request is in a queue already. ignoreDupResponses="true" does not cancel the request while it is processed on the server, but just allows to avoid unnecessary updates on the client side if the response isn't actual now |
| immediate | A flag indicating that this component value must be converted and validated immediately (during an Apply Request Values phase), rather than waiting until a Process Validations phase |
| limitToList | If "true", updates on client side ONLY elements from this 'reRender' property. If "false" (default) updates all rendered by ajax region components |
| nodeFace | Node face facet name |
| nodeSelectListener | MethodBinding representing an action listener method that will be notified after selection of node. |
| onbeforedomupdate | JavaScript code for call before DOM has been updated on client side |
| onclick | HTML: a script expression; a pointer button is clicked |
| oncollapse | HTML: script expression to invoke on node collapsing |
| oncomplete | JavaScript code for call after request completed on client side |
| ondblclick | HTML: a script expression; a pointer button is double-clicked |
| ondragend | A JavaScript event handler called after a drag operation |
| ondragenter | A JavaScript event handler called on enter draggable object to zone |
| ondragexit | A JavaScript event handler called after a drag object leaves zone |

| Attribute Name | Description |
|-----------------------|---|
| ondragstart | A JavaScript event handler called before drag object |
| ondrop | It's an event that is called when something is dropped on a drop zone |
| ondropend | A JavaScript handler for event fired on a drop even the drop for a given type is not available |
| onexpand | HTML: script expression to invoke on node expansion |
| onkeydown | HTML: a script expression; a key is pressed down |
| onkeypress | HTML: a script expression; a key is pressed and released |
| onkeyup | HTML: a script expression; a key is released |
| onmousedown | HTML: script expression; a pointer button is pressed down |
| onmousemove | HTML: a script expression; a pointer is moved within |
| onmouseout | HTML: a script expression; a pointer is moved away |
| onmouseover | HTML: a script expression; a pointer is moved onto |
| onmouseup | HTML: script expression; a pointer button is released |
| onselected | HTML: script expression to invoke on node selection |
| preserveDataInRequest | If "true", data is preserved in a request |
| preserveModel | It can be "state", "request", "none". The default is "request" |
| rendered | If "false", this component is not rendered |
| requestDelay | Attribute defines the time (in ms.) that the request will be wait in the queue before it is ready to send. When the delay time is over, the request will be sent to the server or removed if the newest 'similar' request is in a queue already |
| reRender | Id[s] (in format of call <code>UIComponent.findComponent()</code> of components, rendered in case of <code>AjaxRequest</code> caused by this component. Can be single id, comma-separated list of Id's, or EL Expression with array or Collection |
| rowKeyVar | The attribute provides access to a row key in a Request scope |

| Attribute Name | Description |
|---------------------|--|
| selectedClass | Corresponds to the HTML class attribute. Applied to selected node |
| showConnectingLines | If "true", connecting lines are show |
| stateAdvisor | ValueBinding pointing at instance of class implementing org.richfaces.component.state.TreeStateAdvisor interface. |
| stateVar | The attribute provides access to a component state on the client side |
| status | ID (in format of call UIComponent.findComponent()) of Request status component |
| style | CSS style(s) is/are to be applied when this component is rendered |
| styleClass | Corresponds to the HTML class attribute |
| switchType | Tree switch algorithm: "client", "server", "ajax" |
| timeout | Response waiting time on a particular request. If a response is not received during this time, the request is aborted |
| toggleOnClick | If "false" do not toggle node state on click. If "true", than node will be toggles on click on ether node content, or node icon. Default value is false. |
| treeNodeVar | The attribute provides access to a TreeNode instance in a Request scope |
| typeMapping | Map between a draggable type and an indicator name on zone. it's defined with the pair (drag type:indicator name)) |
| value | The current value for this component |
| var | Attribute contains a name providing an access to data defined with value |

Table 6.361. Component identification parameters

| Name | Value |
|------------------|---------------------------------------|
| component-type | org.richfaces.Tree |
| component-class | org.richfaces.component.html.HtmlTree |
| component-family | org.richfaces.Tree |

| Name | Value |
|---------------|------------------------------|
| renderer-type | org.richfaces.TreeRenderer |
| tag-class | org.richfaces.taglib.TreeTag |

6.78.3. Creating the Component with a Page Tag

There are two cases for a tree definition: together with `<rich:recursiveTreeNodesAdaptor>` or `<rich:treeNodesAdaptor>` and without them. In the first case, it's possible to define it without `"value"` and `"var"` attributes as follows:

Example:

```
...
    <rich:tree>
        <rich:recursiveTreeNodesAdaptor roots="#{fileSystemBean.sourceRoots}"
        var="item" nodes="#{item.nodes}" />
    </rich:tree>
...
```

In the second case, it's necessary to define it with these attributes as it's shown below:

Example:

```
...
    <rich:tree value="#{library.data}" var="item" >
        <rich:treeNode icon="/images/tree/singer.png" >
            <h:outputText value="#{item.name}" />
        </rich:treeNode>
        ...
    </rich:tree>
...
```

6.78.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlTree;
...
HtmlTree myTree = new HtmlTree();
...
```

6.78.5. Details of Usage

As it has been mentioned above the `<rich:tree>` component allows rendering any tree-like data model.

The component interacts with data model through "TreeNode" interface (org.richfaces.model.TreeNode [http://labs.jboss.com/file-access/default/members/jbossrichfaces/freezone/docs/apidoc_framework/org/richfaces/model/TreeNode.html]) that is used for tree nodes representation. The `"value"` attribute of the `<rich:tree>` component contains a nodes structure defined in a bean property. The property keeps a structure of objects that implement "TreeNode" interface.

"TreeNode" has a property "data" (see `org.richfaces.model.TreeNode` [http://labs.jboss.com/file-access/default/members/jbossrichfaces/freezone/docs/apidoc_framework/org/richfaces/model/TreeNode.html]). Data contained in the property are placed in a request scope variable, which name is defined with *"var"* attribute for the **<rich:tree>** component.

You can develop and use your own implementation for the "TreeNode" interface or use a default implementation, which is defined with a default class "TreeNodeImpl" (`org.richfaces.model.TreeNodeImpl` [http://labs.jboss.com/file-access/default/members/jbossrichfaces/freezone/docs/apidoc_framework/org/richfaces/model/TreeNodeImpl.html]).

There is "XmlTreeDataBuilder" class (`org.richfaces.component.xml.XmlTreeDataBuilder` [<http://labs.jboss.com/file-access/default/members/jbossrichfaces/freezone/docs/apidoc/org/richfaces/component/xml/XmlTreeDataBuilder.html>]) that allows transforming XML into structures of objects containing "XmlNodeData" (`org.richfaces.component.xml.XmlNodeData` [http://labs.jboss.com/file-access/default/members/jbossrichfaces/freezone/docs/apidoc_framework/org/richfaces/component/xml/XmlNodeData.html]) instances as data, which could be represented by the **<rich:tree>** component.

It's possible to define a visual representation of a data model node (to define a node icon) and its behavior in a compliance with data contained in this node (with a value of the *"var"* attribute). The node behavior is defined by components nested to the **<rich:treeNode>** (it could be links or buttons, for example). For these purposes you should use *"nodeFace"* attribute. For each tree node a value of *"nodeFace"* attribute is evaluated and **<rich:treeNode>** with a value of *"type"* attribute equal to a value of *"nodeFace"* is used for node representation. An example is placed below.

Example:

```
...
    <h:form>
        <rich:tree style="width:300px" value="#{library.data}" var="item"
nodeFace="#{item.type}">
            <rich:treeNode type="artist" iconLeaf="/images/tree/singer.png"
icon="/images/tree/singer.png">
                <h:outputText value="#{item.name}" />
            </rich:treeNode>
            <rich:treeNode type="album" iconLeaf="/images/tree/disc.png"
icon="/images/tree/disc.png">
                <h:outputText value="#{item.title}" />
            </rich:treeNode>
            <rich:treeNode type="song" iconLeaf="/images/tree/song.png"
icon="/images/tree/song.png">
                <h:outputText value="#{item.title}" />
            </rich:treeNode>
        </rich:tree>
    </h:form>
...
```

This is a result:

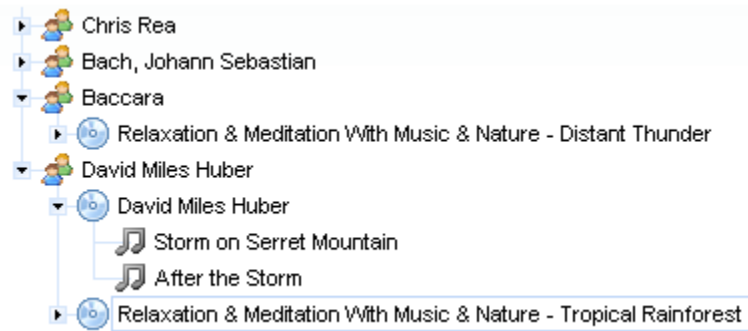


Figure 6.141. The *"nodeFace"* attribute usage

In the example when each node of data model is processed, data contained in the property "data" of "TreeNode" interface are placed in a request scope variable, which name is defined with *"var"* attribute. The value of the *"nodeFace"* attribute is evaluated in compliance with data placed in the *"var"* attribute. And for the node representation is used corresponding `<rich:treeNode>` component (with a value of *"type"* attribute equal to a value of *"nodeFace"*). For example, during processing of data model an object with name "Chris Rea" was placed in the *"var"* attribute. Then value of *"nodeFace"* attribute is evaluated as "artist". Thus, for node representation the `<rich:treeNode>` with *"type"* equal to "artist" is used.

Also you can define an EL-expression as value of the *"nodeFace"* attribute. See an example below:

Example:

```
nodeFace="#{data.name != 'param-value' ? 'artist' : 'album'}"
```

There are some essential moments in a *"nodeFace"* attribute usage. For their description it's necessary to define notions of a "typeless node" and a "default node".

The typeless node is the first `<rich:treeNode>` component (from all children nodes nested to the `<rich:tree>` component) with not defined *"type"* attribute and defined *"rendered"* attribute. The typeless node is used for representation when *"nodeFace"* attribute is null.

Default node has the following interior presentation:

Example:

```
<h: outputText value="#{varAttributeName}">
```

"varAttributeName" is a value for *"var"* attribute.

Default node is used in the following cases:

- *"nodeFace"* attribute is defined, but its value isn't equal to any *"type"* attribute value from all children nodes;
- *"nodeFace"* attribute is defined and its value is equal to a value of some *"type"* attribute from all children nodes, but the value of *"rendered"* attribute for this node is "false".

There is also one especial moment in the *"type"* and *"rendered"* attributes usage. It's possible to define several `<rich:treeNode>` components with the equal values of *"type"* attribute and different values of *"rendered"* attribute. It provides the possibility to define different representation for the same nodes type. In the example with artists and their albums (see above [454]) it's possible to represent albums that are available for sale and albums that are not available. An example is placed below:

Example:

```
...
<h:form>
  <rich:tree style="width:300px" value="#{library.data}" var="item"
nodeFace="#{item.type}">
    ...
    <rich:treeNode type="album" iconLeaf="/images/tree/album.gif"
icon="/images/tree/album.gif"
        rendered="#{item.exist}">
            <h:outputText value="#{item.name}" />
        </rich:treeNode>
    <rich:treeNode type="album" iconLeaf="/images/tree/album_absent.gif"
icon="/images/tree/album_absent.gif"
        rendered="#{not item.exist}">
            <h:outputText value="#{item.name}" />
        </rich:treeNode>
    ...
  </rich:tree>
</h:form>
...
```

This is a result:

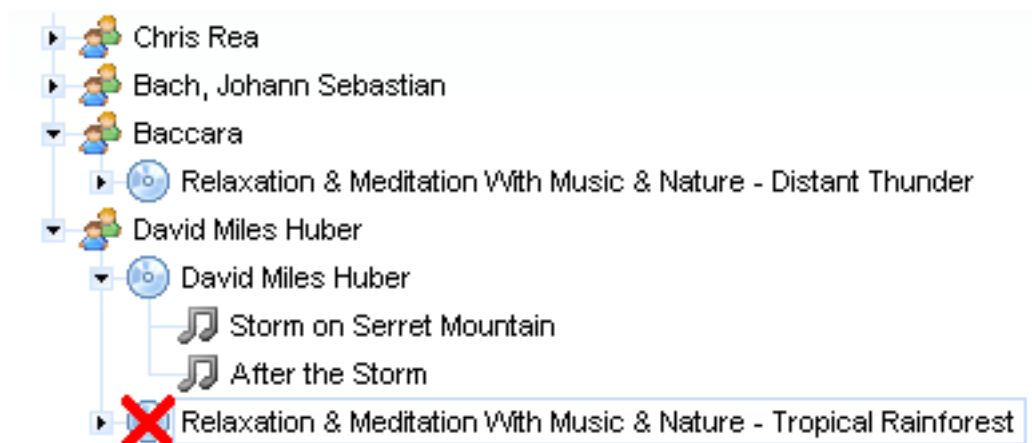


Figure 6.142. The *"type"* and the *"rendered"* attributes usage

In the example the `<rich:treeNode>` components has equal values of the *"type"* attribute. In a compliance with value of the *"rendered"* attribute corresponding `<rich:treeNode>` component is selected for node representation. If an album is available for sale the value of the *"rendered"* for the first `<rich:treeNode>` component is *"true"*, for the second one is *"false"*. Thus, the first `<rich:treeNode>` is selected for node representation.

Switching on tree nodes could be performed in three modes. Modes could be specified with *"switchType"* attribute for **<rich:tree>** component.

- Ajax (default value) - Ajax submission is used for switching. Note, that for collapse/expand operations it is sent an Ajax request to the server. It could be seen a short delay in this case.
- Server - regular form submission request is used. Note, that all nodes actually render up-front and expanding/collapsing do not require server call.
- Client - switching is done as a whole on a client, no interaction with a server presents. Full page content is reloaded after every click.

The *"icon"*, *"iconCollapsed"*, *"iconExpanded"*, *"iconLeaf"* attributes define icons for the component. Also you can define icons using facets with the same names. If the facets are defined, the corresponding attributes are ignored and facets contents are used as icons. By default the width of a rendered facet area is 16px.

Example:

```
...
    <rich:tree ....>
        ...
        <f:facet name="icon">
            <h:graphicImage value="/images/tree/singer.png " />
        </f:facet>
        <f:facet name="iconCollapsed">
            <h:graphicImage value="/images/tree/singer.png " />
        </f:facet>
        <f:facet name="iconExpanded">
            <h:graphicImage value="/images/tree/singer.png " />
        </f:facet>
        <f:facet name="iconLeaf">
            <h:graphicImage value="/images/tree/song.png " />
        </f:facet>
        ...
    </rich:tree>
...

```

The **<rich: tree>** component can be used together with **<rich: treeNodeAdaptor>**. In this case there is no need to specify the attributes *"value"* and *"var"*. Besides, visual representation shouldn't be defined right in the tree. In this case a tree tag is intended mainly for defining common attributes such as *"ajaxSubmitSelection"*, for instance.

6.78.6. Built-In Drag and Drop

The **<rich: tree>** component functionality provides a built-in support for Drag and Drop operations. The main usage principles are the same as for RichFaces DnD wrapper components. Hence, to get additional information on the topic, read the corresponding chapters: *"rich:dndParam"*, *"rich:dragSupport"*, *"rich:dragIndicator"*, *"rich:dropSupport"*. As treeNodes could be Drag, Drop or DnD elements, a tree can include the following attributes groups.

Table 6.362. Drag attributes group

| | |
|------------------|---|
| dragValue | Element value drag passing into processing after a Drop event |
| dragListener | A listener that processes a Drag event |
| dragIndicator | Id of a component that is used as drag pointer during the drag operation |
| dragType | Defines a drag zone type that is used for definition of a dragged element, which can be accepted by a drop zone |

Table 6.363. Drop attributes group

| | |
|------------------|--|
| dropValue | Element value drop passed into processing after Drop events . |
| dropListener | A listener that processes a Drop event. |
| acceptedTypes | Drag zone names are allowed to be processed with a Drop zone. |
| typeMapping | Drag zones names mapping on the corresponding drop zone parameters. |

An example of usage is placed below.

Example:

```

...
    <h:form>
        <rich:tree dragIndicator=":treeDragIndicator"
dropListener="#{libraryAjaxTree.processDrop}" style="width:300px"
value="#{libraryAjaxTree.data}" var="item" nodeFace="#{item.type}">
            <rich:treeNode type="artist" acceptedTypes="album"
iconLeaf="/images/tree/group.png" icon="/images/tree/group.png">
                <h:outputText value="#{item.name}" />
            </rich:treeNode>
            <rich:treeNode type="album" dragType="album" acceptedTypes="song"
iconLeaf="/images/tree/cd.png" icon="/images/tree/cd.png">
                <h:outputText value="#{item.title}" />
                <rich:dndParam name="label" type="drag" value="Album: #{item.title}"
/>
            </rich:treeNode>
            <rich:treeNode type="song" dragType="song"
iconLeaf="/images/tree/music.png" icon="/images/tree/music.png">
                <h:outputText value="#{item.title}" />
                <rich:dndParam name="label" type="drag" value="Song: #{item.title}"
/>
            </rich:treeNode>
        </rich:tree>
    </h:form>
...

```

In the example a song from one album can be dragged into other because for the second treeNode with `"type"="album"` was defined attribute `"acceptedTypes"="song"`. Its value is equal to the value of the `"type"` attribute defined for the third treeNode (see picture below). Also an album can be dragged into treeNode with `"type"="artist"`.

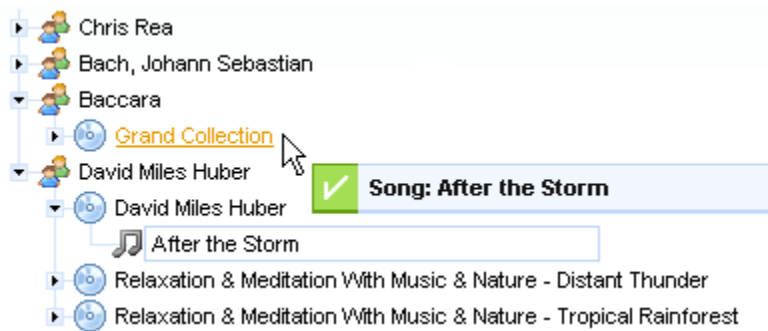


Figure 6.143. DnD operations

6.78.7. Events handling

Listeners classes that process events on the server side are defined with the help of:

- `nodeSelectListener` is called during request sending on a node selecting event (if request sending on this event is defined)
- `dropListener` processes a Drop event
- `dragListener` processes a Drag event
- `changeExpandListener` processes expand/collapse event of a `treeNode`

Listener methods can be defined using the following attributes or using nested tags.

Client event attributes are:

- `onexpand` - expands a node event
- `oncollapse` - collapses a node event
- `ondragexit` - element passing out from a tree zone event
- `ondragstart` - drags a start event
- `ondragend` - drags an end event (a drop event)
- `ondragenter` - drags an element appearing on a tree event

They can be used to add some JavaScript effects.

Also standart HTML event attributes like `"onclick"`, `"onmousedown"`, `"onmouseover"` and etc. could be used. Event handlers of a `<rich:tree>` component capture events occurred on any tree part. But event handlers of `treeNode` capture events occurred on `treeNode` only, except for children events.

6.78.8. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

There are two ways to redefine the appearance of all `<rich:tree>` components at once:

- Redefine the corresponding skin parameters
- Add to your style sheets *style classes* used by a `<rich:tree>` component

6.78.9. Skin Parameters Redefinition:

There is only one skin parameter for `<rich:tree>`. As it's a wrapper component for `<rich:treeNode>` components, look and feel customization is described in the corresponding section.

Table 6.364. Skin parameters for a wrapper element

| Skin parameters | CSS properties |
|-------------------|------------------|
| overAllBackground | background-color |

6.78.10. Definition of Custom Style Classes

Table 6.365. Classes names that define a component appearance

| Class name | Description |
|------------|---|
| rich-tree | Defines styles for a wrapper <code><div></code> element of a tree |

In order to redefine styles for all `<rich:tree>` components on a page using CSS, it's enough to create a class with the same names and define necessary properties in them.

To change styles of particular `<rich:tree>` components define your own style class in the corresponding `<rich:tree>` attribute.

6.78.11. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/tree.jsf?c=tree>] you can see the example of `<rich:tree>` usage and sources for the given example.

How to Expand/Collapse Tree Nodes from code, see here [<http://labs.jboss.com/wiki/ExpandCollapseTreeNode>].

6.79. < rich:treeNode >

6.79.1. Description

A component is used for designing templates for nodes definition.

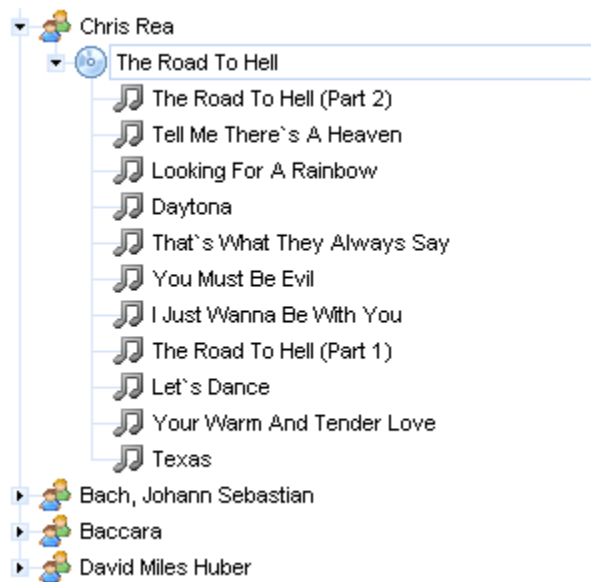


Figure 6.144. <rich:treeNode> component

6.79.2. Key Features

Table 6.366. rich : treeNode attributes

| Attribute Name | Description |
|----------------------|---|
| acceptedTypes | List of drag types to be processed by the current drop zone |
| ajaxSingle | if "true", submits ONLY one field/link, instead of all form controls |
| ajaxSubmitSelection | An algorithm of AJAX request submission. "inherit", "true", "false" values are possible |
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| bypassUpdates | If "true", after process validations phase it skips updates of model beans on a force render response. It can be used for validating components input |
| changeExpandListener | Listener called on expand/collapse event on the node |
| data | Serialized (on default with JSON) data passed on the client by a developer on AJAX request. It's accessible via "data.foo" syntax |
| dragIndicator | Id of the dragIndicator component used as drag operation cursor |
| dragListener | MethodBinding representing an action listener method that will be notified after drag operation |

| Attribute Name | Description |
|--------------------|--|
| dragType | Key of a drag object. It's used to define a necessity of processing the current dragged element on the drop zone side |
| dragValue | Data to be sent to the drop zone after a drop event |
| dropListener | MethodBinding representing an action listener method that will be notified after drop operation |
| dropValue | Data to be processed after a drop event |
| eventsQueue | Name of requests queue to avoid send next request before complete other from same event. Can be used to reduce number of requests of frequently events (key press, mouse move etc.) |
| focus | id of element to set focus after request completed on client side |
| highlightedClass | Corresponds to the HTML class attribute. Applied to highlighted node |
| icon | The icon for node |
| iconCollapsed | The icon for collapsed node |
| iconExpanded | The icon for expanded node |
| iconLeaf | An icon for component leaves |
| id | Every component may have a unique id that is automatically created if omitted |
| ignoreDupResponses | Attribute allows to ignore an Ajax Response produced by a request if the newest 'similar' request is in a queue already. ignoreDupResponses="true" does not cancel the request while it is processed on the server, but just allows to avoid unnecessary updates on the client side if the response isn't actual now |
| limitToList | If "true", updates on client side ONLY elements from this 'reRender' property. If "false" (default) updates all rendered by ajax region components |
| nodeClass | Name of node CSS class |
| nodeSelectListener | MethodBinding representing an action listener method that will be notified after selection of node. |
| onbeforedomupdate | JavaScript code for call before DOM has been updated on client side |

| Attribute Name | Description |
|----------------|--|
| onclick | HTML: a script expression; a pointer button is clicked |
| oncollapse | HTML: script expression to invoke on node collapsing |
| oncomplete | JavaScript code for call after request completed on client side |
| oncontextmenu | JavaScript handler to be called on right click. Returning false prevents default browser context menu from being displayed |
| ondblclick | HTML: a script expression; a pointer button is double-clicked |
| ondragend | A JavaScript event handler called after a drag operation |
| ondragenter | A JavaScript event handler called on enter draggable object to zone |
| ondragexit | A JavaScript event handler called after a drag object leaves zone |
| ondragstart | A JavaScript event handler called before drag object |
| ondrop | It's an event that is called when something is dropped on a drop zone |
| ondropend | A JavaScript handler for event fired on a drop even the drop for a given type is not available |
| onexpand | HTML: script expression to invoke on node expansion |
| onkeydown | HTML: a script expression; a key is pressed down |
| onkeypress | HTML: a script expression; a key is pressed and released |
| onkeyup | HTML: a script expression; a key is released |
| onmousedown | HTML: script expression; a pointer button is pressed down |
| onmousemove | HTML: a script expression; a pointer is moved within |
| onmouseout | HTML: a script expression; a pointer is moved away |
| onmouseover | HTML: a script expression; a pointer is moved onto |
| onmouseup | HTML: script expression; a pointer button is released |
| onselected | HTML: script expression to invoke on node selection |
| rendered | If "false", this component is not rendered |

| Attribute Name | Description |
|----------------|---|
| requestDelay | Attribute defines the time (in ms.) that the request will be wait in the queue before it is ready to send. When the delay time is over, the request will be sent to the server or removed if the newest 'similar' request is in a queue already |
| reRender | Id[s] (in format of call <code>UIComponent.findComponent()</code>) of components, rendered in case of <code>AjaxRequest</code> caused by this component. Can be single id, comma-separated list of Id's, or EL Expression with array or Collection |
| selectedClass | Corresponds to the HTML class attribute. Applied to selected node |
| status | ID (in format of call <code>UIComponent.findComponent()</code>) of Request status component |
| timeout | Response waiting time on a particular request. If a response is not received during this time, the request is aborted |
| type | A node type |
| typeMapping | Map between a draggable type and an indicator name on zone. it's defined with the pair (drag type:indicator name)) |

Table 6.367. Component identification parameters

| Name | Value |
|------------------|--|
| component-type | <code>org.richfaces.TreeNode</code> |
| component-class | <code>org.richfaces.component.html.HtmlTreeNode</code> |
| component-family | <code>org.richfaces.TreeNode</code> |
| renderer-type | <code>org.richfaces.TreeNodeRenderer</code> |
| tag-class | <code>org.richfaces.taglib.TreeNodeTag</code> |

6.79.3. Creating the Component with a Page Tag

Here is a simple example as it could be used on a page:

Example:

```
...
<rich:tree ... faceNode="simpleNode">
  <rich:treeNode type="simpleNode">
```

```

        <!--Tree node data displaying template-->
        </rich:treeNode>
    </rich:tree>
    ...

```

6.79.4. Creating the Component Dynamically Using Java

Example:

```

import org.richfaces.component.html.HtmlTreeNode;
...
HtmlTreeNode myPanel = new HtmlTreeNode();
...

```

6.79.5. Details of Usage

As it has been mentioned above, `treeNode` defines a template for nodes rendering in a tree. Thus, during XML document rendering (a web.xml application) as a tree, the following nodes output (passed via `var="data"` on a tree) happens:

Example:

```

...
<rich:tree ... faceNode="simpleNode" ... value="#{bean.data}" var="data">
    <rich:treeNode type="simpleNode">
        <h:outputText value="context-param:" />
        <h:inputText value="#{data.name}" />
    </rich:treeNode>
</rich:tree >
...

```

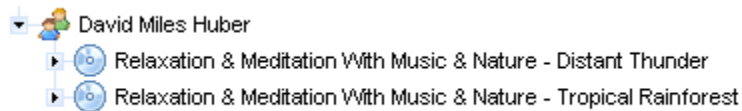


Figure 6.145. Nodes output

Hence, `outputText` outputs the "context-param" string and then the input is output for the `data.name` element of this node.

Different nodes for rendering could be defined depending on some conditions on the tree level. Each condition represents some rendering template. To get more information on various `treeNodesAdaptorAdaptor` definition for nodes, see the tree component chapter.

Switching between expanded/collapsed modes is also managed on the tree level and defined in the corresponding section.

Default nodes of the tree level as well as ones defined with the `treeNodesAdaptorAdaptor` component could send Ajax requests when selected with the mouse, it's managed with the `"ajaxSubmitSelection"` attribute (true/false).

The *"icon"*, *"iconCollapsed"*, *"iconExpanded"*, *"iconLeaf"* attributes define icons for the component. Also you can define icons using facets with the same names. If the facets are defined, the corresponding attributes are ignored and facets contents are used as icons. By default the width of a rendered facet area is 16px.

Example:

```
...
    <rich:tree ....>
        ...
        <f:facet name="icon">
            <h:graphicImage value="/images/tree/singer.png " />
        </f:facet>
        <f:facet name="iconCollapsed">
            <h:graphicImage value="/images/tree/singer.png " />
        </f:facet>
        <f:facet name="iconExpanded">
            <h:graphicImage value="/images/tree/singer.png " />
        </f:facet>
        <f:facet name="iconLeaf">
            <h:graphicImage value="/images/tree/song.png " />
        </f:facet>
        ...
    </rich:tree>
...
```

6.79.6. Built-in Drag and Drop

The main information on Drag and Drop operations is given in the corresponding paragraph of the tree component chapter. It's only necessary to mention that each node could also be a Drag element as well as a Drop container, i.e. the container and the element have all attributes, listeners and ways of behavior similar to the ones of the **<rich:draggable>** and **<rich:dropZone>** components simultaneously.

6.79.7. Events Handling

Just as Drag and Drop operations it corresponds to the one described on the tree component level for a default Node.

6.79.8. Look-and-Feel Customization

For skinnability implementation, the components use a *style class redefinition method*. Default style classes are mapped on *skin parameters*.

There are two ways to redefine the appearance of all **<rich:treeNode>** components at once:

- Redefine the corresponding skin parameters
- Add to your style sheets *style classes* used by a **<rich:treeNode>** component

6.79.9. Skin Parameters Redefinition

Table 6.368. Skin parameters for a node element

| Skin parameters | CSS properties |
|--------------------------|----------------|
| panelTextColor | color |
| preferableDataSizeFont | font-size |
| preferableDataFamilyFont | font-family |

Table 6.369. Skin parameters for a selected element

| Skin parameters | CSS properties |
|-----------------------|----------------|
| headerBackgroundColor | border-color |
| panelTextColor | color |
| selectControlColor | color |

Table 6.370. Skin parameters for a mouseovered element

| Skin parameters | CSS properties |
|--------------------|----------------|
| selectControlColor | color |

6.79.10. Definition of Custom Style Classes

On the screenshot there are classes names that define styles for component elements.

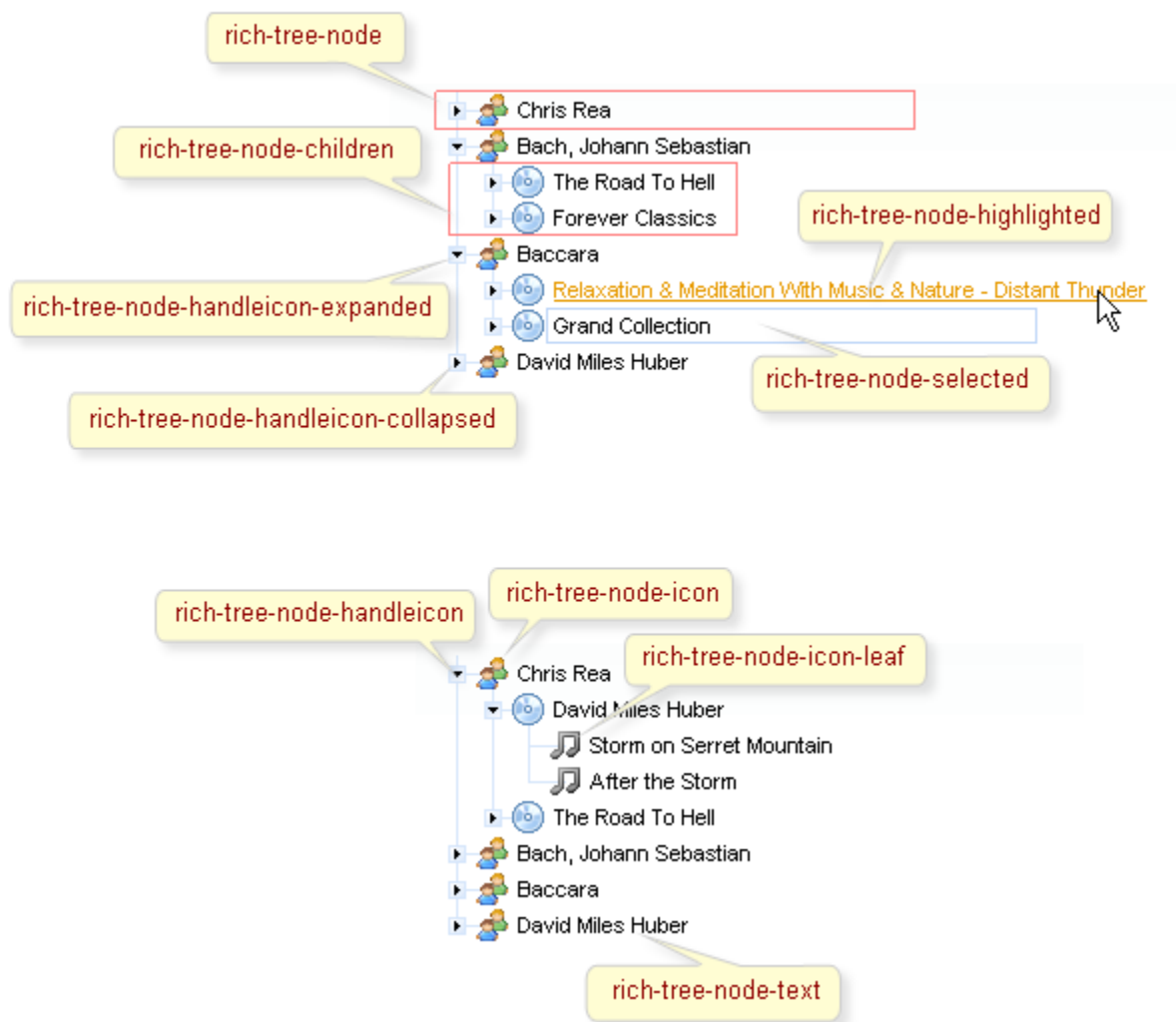


Figure 6.146. Classes names

Table 6.371. Classes names that define a node element

| Class name | Description |
|---------------------------|---|
| rich-tree-node | Defines styles for a tree node |
| rich-tree-node-handleicon | Defines styles for a tree node handleicon |
| rich-tree-node-children | Defines styles for all tree node subnodes |
| rich-tree-node-text | Defines styles for a tree node text |
| rich-tree-node-icon | Defines styles for a tree node icon |
| rich-tree-node-icon-leaf | Defines styles for a tree node icon leaf |

Table 6.372. Classes names that define states for a node element

| Class name | Description |
|-------------------------|---|
| rich-tree-node-selected | Defines styles for a selected tree node |

| Class name | Description |
|-------------------------------------|---|
| rich-tree-node-highlighted | Defines styles for a highlighted tree node |
| rich-tree-node-handleicon-collapsed | Defines styles for a collapsed tree node handleicon |
| rich-tree-node-handleicon-expanded | Defines styles for a expanded tree node handleicon |

In order to redefine the style for all `<rich:treeNode>` components on a page using CSS, it's enough to create classes with the same names and define the necessary properties in them.

To change the style of particular `<rich:treeNode>` components define your own style classes in the corresponding `<rich:treeNode>` attributes.

It is also possible to change look and feel of specific `<rich:treeNode>` with the help of defining for them *"selectedClass"* and *"highlightedClass"* attributes by their specific classes.

6.79.11. Relevant Resources Links

How to Expand/Collapse Tree Nodes from code see here [<http://labs.jboss.com/wiki/ExpandCollapseTreeNodeAdaptor>].

6.80. < rich:changeExpandListener >

6.80.1. Description

The `<rich:changeExpandListener>` represents an action listener method that is notified on an expand/collapse event on the node.

6.80.2. Key Features

- Allows to define some "changeExpand" listeners for the component

Table 6.373. rich : changeExpandListener attributes

| Attribute Name | Description |
|----------------|---|
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| type | Attribute defines the fully qualified Java class name for listener |

Table 6.374. Component identification parameters

| Name | Value |
|----------------|--|
| listener-class | org.richfaces.event.NodeExpandedListener |
| event-class | org.richfaces.event.NodeExpandedEvent |

| Name | Value |
|-----------|--|
| tag-class | org.richfaces.taglib.ChangeExpandListenerTag |

6.80.3. Creating the Component with a Page Tag

Simple Component definition on a page:

Example:

```
...
<rich:changeExpandListener type="demo.Bean"/>
...
```

6.80.4. Creating the Component Dynamically Using Java

Example:

```
package demo;

public class ImplBean implements org.richfaces.event.NodeExpandedListener{
    ...
}
```

```
import demo.ImplBean;
...
ImplBean myListener = new ImplBean();
...
```

6.80.5. Details of usage

The `<rich:changeExpandListener>` is used as a nested tag with `<rich:tree>` and `<rich:treeNode>` components.

Attribute *"type"* defines the fully qualified Java class name for the listener. This class should implement `org.richfaces.event.NodeExpandedListener` interface.

The typical variant of using:

```
...
<rich:tree switchType="server" value="#{project.data}" var="item"
  nodeFace="#{item.type}">
  <rich:changeExpandListener type="demo.ListenerBean"/>
  ...
  <!-- Tree nodes -->
  ...
</rich:tree>
...
```

Java bean source:


```

package demo;

import org.richfaces.event.NodeExpandedEvent;

public class ListenerBean implements org.richfaces.event.NodeExpandedListener{
    ...
    public void processExpansion(NodeExpandedEvent arg0){
        //Custom Developer Code
    }
    ...
}

```

6.80.6. Look-and-Feel Customization

`<rich:changeExpandListener>` has no skin parameters and custom style classes, as the component isn't visual.

6.81. < rich:nodeSelectListener >

6.81.1. Description

The `<rich:nodeSelectListener>` represents an action listener method that will be notified after selection of a node.

6.81.2. Key Features

- Allows to define some "nodeSelect" listeners for the component

Table 6.375. rich : nodeSelectListener attributes

| Attribute Name | Description |
|----------------|---|
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| type | Attribute defines the fully qualified Java class name for listener |

Table 6.376. Component identification parameters

| Name | Value |
|----------------|--|
| listener-class | org.richfaces.event.NodeSelectedListener |
| event-class | org.richfaces.event.NodeSelectedEvent |
| tag-class | org.richfaces.taglib.NodeSelectListenerTag |

6.81.3. Creating the Component with a Page Tag

Simple Component definition on a page:

Example:

```
...
<rich:nodeSelectListener type="demo.Bean"/>
...
```

6.81.4. Creating the Component Dynamically Using Java**Example:**

```
package demo;

public class ImplBean implements org.richfaces.event.NodeSelectListener{
    ...
}
```

```
import demo.ImplBean;
...
ImplBean myListener = new ImplBean();
...
```

6.81.5. Details of usage

The `<rich:nodeSelectListener>` is used as a nested tag with `<rich:tree>` and `<rich:treeNode>` components.

Attribute *"type"* defines the fully qualified Java class name for listener. This class should implement `org.richfaces.event.NodeSelectedListener` interface.

The typical variant of using:

```
...
<rich:tree switchType="server" value="#{project.data}" var="item"
  nodeFace="#{item.type}">
  <rich:nodeSelectListener type="demo.ListenerBean"/>
  ...
  <!-- Tree nodes -->
  ...
</rich:tree>
...
```

Java bean source:

```
package demo;

import org.richfaces.event.NodeSelectedEvent;

public class ListenerBean implements org.richfaces.event.NodeSelectedListener{
    ...
    public void processSelection(NodeSelectedEvent arg0){
        //Custom Developer Code
    }
    ...
}
```

6.81.6. Look-and-Feel Customization

`<rich:nodeSelectListener>` has no skin parameters and custom style classes, as the component isn't visual.

6.82. `< rich:recursiveTreeNodesAdaptor >`

6.82.1. Description

The `rich:recursiveTreeNodesAdaptor` provides possibility to define data models and process nodes recursively.



Figure 6.147. Expanded tree with recursive `<rich:treeNodesAdaptor>`

6.82.2. Key Features

- Allows to define combined data models
- Possibility to define nodes for processing via attributes
- Allows to process nodes recursively

Table 6.377. `rich : recursiveTreeNodesAdaptor` attributes

| Attribute Name | Description |
|----------------|---|
| binding | The attribute takes a value-binding expression for a component property of a backing bean |

| Attribute Name | Description |
|----------------|--|
| id | Every component may have a unique id that is automatically created if omitted |
| included | This boolean expression is used to define which elements of both collections are processed |
| includedNode | This boolean expression is used to define which elements are processed |
| includedRoot | This boolean expression is used to define which elements are processed applying to "roots" collection |
| nodes | Defines collection to use at the other (non-top) levels of iteration |
| rendered | If "false", this component is not rendered |
| roots | Defines collection to use at the top of iteration |
| var | A request-scope attribute via which the data object for the current collection element will be used when iterating |

Table 6.378. Component identification parameters

| Name | Value |
|------------------|---|
| component-type | org.richfaces.RecursiveTreeNodesAdaptor |
| component-class | org.richfaces.component.html.HtmlRecursiveTreeNodesAd |
| component-family | org.richfaces.RecursiveTreeNodesAdaptor |
| tag-class | org.richfaces.taglib.RecursiveTreeNodesAdaptorTag |

6.82.3. Creating the Component with a Page Tag

To create the simplest variant of rich:recursiveTreeNodesAdaptor on a page, use the following syntax:

Example:

```

...
<rich:recursiveTreeNodesAdaptor var="issue" root="#{project.root}"
nodes="#{model.issues}">
  ...
  <rich:treeNode>
    <!-- node content -->
  </rich:treeNode>

  <!-- Others nodes -->
  ...
</rich:recursiveTreeNodesAdaptor>

```

...

6.82.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlRecursiveTreeNodesAdaptor;
...
HtmlRecursiveTreeNodesAdaptor myRecursiveTreeNodesAdaptor = new
    HtmlRecursiveTreeNodesAdaptor();
...
```

6.82.5. Details of Usage

The typical variant of using:

```
...
<rich:tree adviseNodeOpened="#{treeModelBean.adviseNodeOpened}" switchType="client">
  <rich:treeNodesAdaptor id="project" nodes="#{loaderBean.projects}" var="project">

    <rich:treeNode>
      <h:commandLink action="#{project.click}" value="Project: #{project.name}" />
    </rich:treeNode>

    <rich:recursiveTreeNodesAdaptor id="dir" var="dir" root="#{project.dirs}"
    nodes="#{dir.directories}">

      <rich:treeNode>
        <h:commandLink action="#{dir.click}" value="Directory: #{dir.name}" />
      </rich:treeNode>

      <rich:treeNodesAdaptor id="file" var="file" nodes="#{dir.files}">
        <rich:treeNode>
          <h:commandLink action="#{file.click}" value="File: #{file.name}" />
        </rich:treeNode>
      </rich:treeNodesAdaptor>

      <rich:treeNodesAdaptor id="file1" var="file" nodes="#{dir.files}">
        <rich:treeNode>
          <h:commandLink action="#{file.click}" value="File1: #{file.name}" />
        </rich:treeNode>
      </rich:treeNodesAdaptor>

      <rich:recursiveTreeNodesAdaptor id="archiveEntry" var="archiveEntry"
      roots="#{dir.files}" nodes="#{archiveEntry.archiveEntries}"
      includedRoot="#{archiveEntry.class.simpleName == 'ArchiveFile'}"
      includedNode="#{archiveEntry.class.simpleName == 'ArchiveEntry'}">

        <rich:treeNode id="archiveEntryNode">
          <h:commandLink action="#{archiveEntry.click}" value="Archive entry:
#{archiveEntry.name}" />
        </rich:treeNode>

      </rich:recursiveTreeNodesAdaptor>
    </rich:treeNodesAdaptor>
  </rich:tree>
```

```

</rich:recursiveTreeNodesAdaptor>
</rich:treeNodesAdaptor>
</rich:tree>
...

```

6.83. < rich:treeNodesAdaptor >

6.83.1. Description

The rich:treeNodesAdaptor provides possibility to define data models and create representations for them.

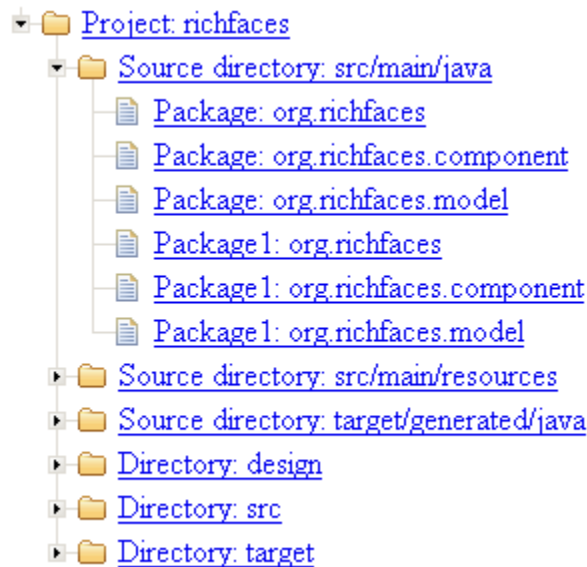


Figure 6.148. Expanded tree with <rich:treeNodesAdaptor>

6.83.2. Key Features

- Allows to define combined data models
- Possibility to define nodes for processing via attributes

Table 6.379. rich : treeNodesAdaptor attributes

| Attribute Name | Description |
|----------------|---|
| binding | The attribute takes a value-binding expression for a component property of a backing bean |
| id | Every component may have a unique id that is automatically created if omitted |
| includedNode | This boolean expression is used to define which elements are processed |
| nodes | Defines collection to use at the other (non-top) levels of iteration |

| Attribute Name | Description |
|----------------|--|
| rendered | If "false", this component is not rendered |
| var | A request-scope attribute via which the data object for the current collection element will be used when iterating |

Table 6.380. Component identification parameters

| Name | Value |
|------------------|--|
| component-type | org.richfaces.TreeNodeAdaptor |
| component-class | org.richfaces.component.html.HtmlTreeNodeAdaptor |
| component-family | org.richfaces.TreeNodeAdaptor |
| tag-class | org.richfaces.taglib.TreeNodeAdaptorTag |

6.83.3. Creating the Component with a Page Tag

To create the simplest variant of `rich:treeNodesAdaptor` on a page, use the following syntax:

Example:

```
...
<rich:treeNodesAdaptor var="issue" nodes="#{model.issues}">
  ...
  <rich:treeNode>
    <!-- node content -->
  </rich:treeNode>
  ...
  <!-- Others nodes -->
  ...
</rich:treeNodesAdaptor>
...
```

6.83.4. Creating the Component Dynamically Using Java

Example:

```
import org.richfaces.component.html.HtmlTreeNodeAdaptor;
...
HtmlTreeNodeAdaptor myTreeNodeAdaptor = new HtmlTreeNodeAdaptor();
...
```

6.83.5. Details of Usage

The typical variant of using:

```
...
<rich:tree adviseNodeOpened="#{treeModelBean.adviseNodeOpened}" switchType="client">
```

```

<rich:treeNodesAdaptor id="project" nodes="#{loaderBean.projects}" var="project">
  <rich:treeNode>
    <h:commandLink action="#{project.click}" value="Project: #{project.name}" />
  </rich:treeNode>
  <rich:treeNodesAdaptor id="srcDir" var="srcDir" nodes="#{project.srcDirs}">
    <rich:treeNode>
      <h:commandLink action="#{srcDir.click}" value="Source directory: #{srcDir.name}"
    />
    </rich:treeNode>
    <rich:treeNodesAdaptor id="pkg" var="pkg" nodes="#{srcDir.packages}">
      <rich:treeNode>
        <h:commandLink action="#{pkg.click}" value="Package: #{pkg.name}" />
      </rich:treeNode>
      <rich:treeNodesAdaptor id="class" var="class" nodes="#{pkg.classes}">
        <rich:treeNode>
          <h:commandLink action="#{class.click}" value="Class: #{class.name}" />
        </rich:treeNode>
      </rich:treeNodesAdaptor>
    </rich:treeNodesAdaptor>
  </rich:treeNodesAdaptor>
</rich:tree>
...

```

6.83.6. Relevant Resources Links

Here [<http://livedemo.exadel.com/richfaces-demo/richfaces/treeNodesAdaptor.jsf?c=treeNodesAdaptor>] you can see the example of **<rich:treeNodesAdaptor>** usage and sources for the given example.

IDE Support

Red Hat Developer Studio 1.0.0 [<http://www.redhat.com/developers/rhds/index.html>] is an IDE that provides full support for Java Server Faces, RichFaces, Facelets, Struts, and other Web technologies. In addition to this, it seamlessly combines visual and source-oriented development approaches. One of the special support feature for RichFaces is that it is available as project "capabilities". These project capabilities can be added to any existing JSF project to make the project a RichFaces JSF project by automatically adding libraries and modifying configuration files as required.

Links to information resources

Table 8.1. Web Resources

| Resources | Links |
|------------------|--|
| JBoss Rich Faces | JBoss Rich Faces [http://labs.jboss.com/portal/jbossrichfaces/] |
| JBoss Forum | JBoss Forums [http://jboss.com/index.html?module=bb&op=main&c=27] |
| Rich Faces Wiki | Rich Faces Wiki [http://labs.jboss.com/wiki/RichFaces] |
| Rich Faces Blog | Rich Faces Blog [http://jroller.com/page/a4j] |