



# **Red Hat JBoss Web Framework Kit 2.7 RichFaces Guide**

---

RichFaces framework for use with Red Hat JBoss Enterprise Application Platform

Red Hat Customer Content Services



## RichFaces framework for use with Red Hat JBoss Enterprise Application Platform

Red Hat Customer Content Services

## Legal Notice

Copyright © 2015 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

## Abstract

This guide presents information about the RichFaces framework shipped with Red Hat JBoss Web Framework Kit.

## Table of Contents

<b>Chapter 1. RichFaces Deprecation</b>	<b>2</b>
<b>Chapter 2. Introduction to Red Hat JBoss Web Framework Kit</b>	<b>3</b>
2.1. About Red Hat JBoss Web Framework Kit	3
2.2. About the JBoss Web Framework Kit Tiers	3
2.3. About the JBoss Web Framework Kit Distribution	4
<b>Chapter 3. Introduction to RichFaces</b>	<b>5</b>
3.1. About JSF	5
3.2. About Ajax	5
3.3. About RichFaces	5
3.4. Advantages of Using RichFaces	6
3.5. Advantages of RichFaces for Mobile Applications	7
3.6. Application Development With RichFaces	7
<b>Chapter 4. Setting up RichFaces</b>	<b>9</b>
4.1. System Requirements	9
4.2. Installing Options	10
4.3. Setting Up Maven	10
4.4. Restrictions	11
<b>Chapter 5. Exploring Examples that use RichFaces</b>	<b>12</b>
5.1. Archetypes	12
5.2. Quickstarts	12
5.3. RichFaces Showcase	12
5.4. RichFaces PhotoAlbum	13
<b>Chapter 6. Developing Applications with RichFaces</b>	<b>15</b>
6.1. Components for Web Application Layout	15
6.2. Components Using Back-End Data	18
6.3. Components using User-Input Data	22
6.4. Components for Advanced Web Application Functionality	24
6.5. Components for Skinning	27
<b>Chapter 7. Using RichFaces for Mobile Applications</b>	<b>33</b>
7.1. Configuring a RichFaces Application for Mobile Devices	33
7.2. Viewport Metadata	34
7.3. JavaScript	35
7.4. MobileESP	35
<b>Chapter 8. Migrating from RichFaces 4 to RichFaces 4.5</b>	<b>38</b>
<b>Revision History</b>	<b>40</b>

## Chapter 1. RichFaces Deprecation



### Warning

RichFaces 4.x is deprecated from the release of JBoss Web Framework Kit 2.7.0. This means that no new features will be introduced to the RichFaces 4.x framework in JBoss Web Framework Kit 2.7.0 or later. As a tier 1 framework, Red Hat continues to support RichFaces 4.x and provide fixes under our standard support terms and conditions. For more information about the Red Hat support policy, see

[https://access.redhat.com/support/policy/updates/jboss\\_notes/](https://access.redhat.com/support/policy/updates/jboss_notes/).

[Report a bug](#)

## Chapter 2. Introduction to Red Hat JBoss Web Framework Kit

### 2.1. About Red Hat JBoss Web Framework Kit

Red Hat JBoss Web Framework Kit is a set of enterprise-ready versions of popular open source frameworks. Together, these frameworks provide a solution for developing light and rich Java-based web applications.

JBoss Web Framework Kit comprises enterprise distributions of JBoss community web application frameworks and tested third-party frameworks. These leading frameworks support fast and easy client-side and server-side application development and testing, with frameworks including RichFaces, jQuery, jQuery Mobile, Hibernate Search, Spring, and Arquillian.

The breadth of JBoss Web Framework Kit frameworks provides choice for Java application development. Each framework has been tested and certified for use in applications deployed to Red Hat JBoss Enterprise Application Platform, Red Hat JBoss Web Server, or Red Hat OpenShift JBoss EAP cartridge. Inclusion in JBoss Web Framework Kit ensures stable versions of frameworks are available over long-term enterprise product life cycles, with regular releases for fixes and nonintrusive feature updates. Further, Red Hat JBoss Developer Studio (an Eclipse-based development environment) provides integrated project templates, quickstarts and tooling for many of the JBoss Web Framework Kit frameworks.

For the complete list of the frameworks composing JBoss Web Framework Kit and the certified platform and framework configurations, see <https://access.redhat.com/site/articles/112543> on the Red Hat Customer Portal.

[Report a bug](#)

### 2.2. About the JBoss Web Framework Kit Tiers

The frameworks composing JBoss Web Framework Kit are categorized in four distinct tiers. A description of each tier and the associated Red Hat support is detailed here.

#### **Tier 1 - Included Components**

These are components that are based wholly or partly on open source technologies that support broad collaboration and where Red Hat maintains a leadership role; as such Red Hat is able to support these components and provide upgrades and fixes under our standard support terms and conditions.

#### **Tier 2 - Tested Frameworks**

These are third-party frameworks where Red Hat does not have sufficient influence and does not provide upgrades and fixes under our standard support terms and conditions. Commercially reasonable support is provided by Red Hat Global Support Services for these frameworks.

#### **Tier 3 - Frameworks in Tested Examples**

These are third-party frameworks where Red Hat does not have sufficient influence and does not provide upgrades and fixes under our standard support terms and conditions. Red Hat supports the examples these frameworks are used in and the generic use cases that these examples intend to demonstrate.

#### **Tier 4 - Confirmed Frameworks**

These are third-party frameworks that do not receive any support from Red Hat, but Red Hat verifies that the frameworks run successfully on Red Hat JBoss Enterprise Application Platform. Frameworks and versions not listed here have not been explicitly tested and certified, and thus may be subject to support limitations.

For a list of JBoss Web Framework Kit frameworks by tier, see <https://access.redhat.com/site/articles/112543> on the Red Hat Customer Portal.

[Report a bug](#)

## 2.3. About the JBoss Web Framework Kit Distribution

The frameworks composing JBoss Web Framework Kit are distributed from a range of sources and in a variety of formats:

- The component frameworks are available from the Red Hat Customer Portal. They are distributed in two alternative formats: a binary for each framework or together as one Maven repository. In addition, the source code for each framework is provided for inspection.
- The third party frameworks are not distributed by Red Hat and each must be obtained from its own source.

A number of defined Maven JBoss stacks are provided as part of the JBoss Web Framework Kit distribution. All of the BOMs defining the JBoss stacks are available in the Maven repository `.zip` file available to download from the Red Hat Customer Portal or from <http://www.jboss.org/developer-materials/> on the JBoss Developer Framework website.

An extensive set of examples are also provided as part of the JBoss Web Framework Kit distribution:

- TicketMonster is a moderately complex application demonstrating a number of the JBoss Web Framework Kit frameworks working together.
- Quickstarts and Maven archetypes illustrate subsets of the JBoss Web Framework Kit frameworks used to create simple applications.
- RichFaces, Snowdrop and Seam demonstrations showcase the power of each framework in web application development.

All of these examples are available from the Red Hat Customer Portal, with TicketMonster, the quickstarts, and the Maven archetypes also available from <http://www.jboss.org/developer-materials/> on the JBoss Developer Framework website.

[Report a bug](#)



## Chapter 3. Introduction to RichFaces

### 3.1. About JSF

JSF stands for JavaServer Faces. It is a Java-based specification used to build user interfaces for web applications. Ajax components added to these interfaces utilize JSF to manage any events a user triggers. RichFaces uses JSF 2 to define the Java elements in a web application. JSF uses RichFaces' XML templates to create user interface components.

JSF provides a global **onError** handler to the client if an exception is triggered. The handler provides the relevant error code and other associated data. The RichFaces Ajax components provide the **error** attribute if extra functionalities are required.

Additional error processing is available:

- ✧ The **<a4j:status>** component has an additional error state.
- ✧ The **<a4j:queue>** component is used to process errors.
- ✧ The JSF 2 **ExceptionHandler** class handles server-side errors such as session expiration.



#### Note

The standard display technology used by JSF 1 was JavaServer Pages (JSP). With JSF 2, the standard display technology has been changed to Facelets, which is a more powerful and more efficient View Declaration Language (VLD) than JSP.

[Report a bug](#)

### 3.2. About Ajax

Ajax stands for Asynchronous JavaScript and XML. It is a method for creating dynamic webpages and applications that can be updated asynchronously. Ajax applications work independently of browsers and platforms. They gather information only from the code itself. An example of an Ajax interface is a dynamic search box which uses JavaScript to display results as the user is typing the letters.

There are a number of Ajax features that can be utilized with RichFaces:

- ✧ **AjaxOutput** is an interface that marks the JSF tree that is updated and rendered on the client for every Ajax request. It only marks the JSF tree if the component or behavior sending the request does not explicitly turn off automatic updates.
- ✧ **AjaxContainer** is an interface that marks the JSF tree that is decoded during an Ajax request. It only marks the JSF tree if the component or behavior sending the request does not explicitly specify an alternative. **AjaxRegion** is an implementation used in this interface.
- ✧ The RichFaces Ajax Extensions plug into the standard JSF 2 Ajax script facility. They extend the script facility with new features and options.

[Report a bug](#)

### 3.3. About RichFaces

The RichFaces framework is a component library for JavaServer Faces (**JSF**). The framework extends the JSF framework's Ajax capabilities with advanced features for enterprise web application development.

RichFaces leverages several parts of the JSF 2 framework including lifecycle, validation, conversion facilities, and management of static and dynamic resources. The RichFaces framework includes components with built-in Ajax support and a customizable appearance that can be incorporated into JSF applications.

The RichFaces framework is made up of two tag libraries: the **a4j** library and the **rich** library. The **a4j** tag library represents *Ajax4jsf*, which provides page-level Ajax support with core Ajax components. This allows developers to use custom Ajax behavior with existing components. The **rich** tag library provides Ajax support at the component level and includes ready-made, self-contained components. These components do not require additional configuration in order to send requests or update.

The components can be used individually or separately:

- ✦ Ajax components are used for page-wide actions. A user can input information in one place, and the resulting output can be displayed anywhere on the page. Separate components are updated. Different Ajax action tags can be inserted into HTML to allow for multiple actions.

An Ajax component always has **a4j** at the beginning followed by an indicator of the action it performs. For example:

```
a4j:status
```

- ✦ Rich components are used for single-panel actions as opposed to page-wide actions. They contain Ajax components and change part of the page layout. Information is pulled from the server rather than being inputted. For example, a user could click on a menu item and the Rich component would cause a drop-down menu to display.

A rich component always has **rich** at the beginning followed by an indicator of the action it performs. For example:

```
rich:autocomplete
```

[Report a bug](#)

## 3.4. Advantages of Using RichFaces

RichFaces provides a number of advantages for enterprise web application development:

- ✦ Create complex application views using out-of-the-box components. The RichFaces user interface (UI) library (**rich**) contains components for adding rich interactive features to JSF applications. It extends the RichFaces framework to include a large set of Ajax-enabled components that come with extensive skinning support. Additionally, the RichFaces framework is designed to be used seamlessly with other third-party libraries on the same page, so you have more options for developing applications.
- ✦ Write your own customized rich components with built-in Ajax support. The Component Development Kit (CDK), which is used for the RichFaces UI library creation, includes a code-generation facility and a templating facility using XHTML (extended hypertext markup language) syntax.

- Generate binary resources on the fly. Extensions to JSF 2 resource-handling facilities can generate images, sounds, **Microsoft Excel** spreadsheets, and more during runtime.
- Create a modern rich UI with skinning technology. RichFaces provides a skinning feature that allows you to define and manage different color schemes and other parameters of the look and feel. You can access the skin parameters from page code during runtime. RichFaces comes packaged with a number of skins to get you started, but you can also create your own customized skins too.

[Report a bug](#)

### 3.5. Advantages of RichFaces for Mobile Applications

Mobile versions of applications have become a necessity as more people use cellular devices to connect. The following are some of the issues commonly faced when using mobile JSF:

- JSF does not inherently allow pass-through attributes. Most mobile frameworks, such as jQuery Mobile, depend on this when dealing with HTML.
- It is important to maintain the native feel of the application. Back button capabilities, bookmarking and overall AJAX support are not inherited by JSF.
- Content is forced to load into a single DOM.
- Minimization of JavaScript and CSS.
- Orientation, touch, and other viewport specific settings can be diminished.

To address these issues, RichFaces has a lightweight, mobile HTML5 based approach, which supports native transitions, back button, bookmarking, and touch support. Please check the *Red Hat JBoss Web Framework Kit Release Notes* for a list of supported web browsers.

There are no alterations to the component code or renderers, only JavaScript and CSS added to help make the application mobile. This approach is backwards-compatible with any RichFaces application and is built from the ground up to support legacy installations as well as new initiatives.

[Report a bug](#)

### 3.6. Application Development With RichFaces

Many businesses have expanded their services online in the form of webshops. Customers can browse merchandise and add items to a shopping cart to purchase them. While these webshops provide a simple way to connect with consumers, the current popularity of smart phones and mobile applications means there is an increasing demand for users to be able to access services on the go at the touch of a button. In addition, the amount of coding required to create a user interface for such a website can prove cumbersome.

Creating web applications has traditionally required a prior knowledge of JavaScript. This made the bar for getting into the mobile market very high for developers who were either not fluent in JavaScript or deterred by the amount of coding involved. With RichFaces, a developer can build rich, AJAX-enabled web applications without writing a single line of JavaScript.

Creating a user interfaces with RichFaces is fast and efficient. Unlike other user interface frameworks where an application is contained to the client, RichFaces uses a server-side rendering model. The HTML is generated on the server according to the state of the application, and is then sent to the client. This means users can access the applications to quickly obtain data, provided their device is online.

RichFaces provides a straightforward way to create easily accessible applications without the need for endless coding. No JavaScript expertise is required. User-friendly layouts can be created efficiently resulting in a functional, easily accessible interface.

[Report a bug](#)

## Chapter 4. Setting up RichFaces

### 4.1. System Requirements

#### Full technical requirements

RichFaces has been developed with an open architecture to be compatible with a wide variety of environments.

A full list of software supported with this release of Red Hat JBoss Web Framework Kit is available on the [Red Hat Customer Portal](#).

#### Server requirements

Red Hat JBoss Enterprise Application Platform 6.3.2 is tested and supported for use of RichFaces 4.5.2 by this release of JBoss Web Framework Kit.



#### Important

Red Hat JBoss Enterprise Application Platform 6.0.x contains jsf-impl and jsf-api jar files as static modules that are not compatible with RichFaces 4.5. As a result, RichFaces 4.5 must be used with JBoss EAP 6.1 and later. This applies to use of the RichFaces framework in your own applications and use of the distributed quickstarts and demos that use RichFaces.



#### Note

RichFaces is designed to be functional on any application server compliant with Java Platform, Enterprise Edition 6 (JEE6), and also in major servlet containers like Jetty 8 or Apache Tomcat 7. However, using RichFaces in these environments is neither tested nor supported by Red Hat.

#### Client requirements

Both desktop and mobile clients access RichFaces applications through a web browser. The following browsers are supported:

- ✧ Mozilla Firefox
- ✧ Google Chrome
- ✧ Microsoft Internet Explorer
- ✧ Stock browser in Android OS
- ✧ Stock Safari in iOS



### Important

Other web browsers may be only partially functional, and are not subject to testing and support by Red Hat.

## Development requirements

Developing applications with the RichFaces framework requires the Java Development Kit (JDK), an implementation of JavaServer Faces (JSF), and a development environment.

### Java Development Kit (JDK)

RichFaces supports the following JDK versions:

- ✧ JDK 1.6 and higher

### JavaServer Faces (JSF)

RichFaces supports the following JSF implementations and frameworks:

- ✧ MyFaces
- ✧ Seam
- ✧ Mojarra

### Development environment

RichFaces applications can be developed using a range of tools, including the following integrated development environments (IDEs):

- ✧ Red Hat JBoss Developer Studio or Eclipse with JBDS
- ✧ Maven

Other environments, such as Idea or NetBeans, could also be used for RichFaces development, but are not detailed in this book.

[Report a bug](#)

## 4.2. Installing Options

There are two methods of installing RichFaces:

- ✧ Download the RichFaces binary from the [Red Hat Customer Portal](#)
- ✧ Obtain using Maven

For detailed instructions on installation, see the Red Hat JBoss Web Framework Kit Installation Guide.

[Report a bug](#)

## 4.3. Setting Up Maven

Maven can be downloaded and installed from Apache's website at <http://maven.apache.org/download.html>. Due to the use of dependency importing, Maven version 3.0.3 or above is required.

Once Maven has been installed, no further configuration is required to begin building Maven projects.

[Report a bug](#)

## 4.4. Restrictions

The following restrictions apply to applications implementing the RichFaces framework:

- » Elements can not be appended on a page using RichFaces Ajax, but can be replaced. Elements that are rendered conditionally should not be targeted in the **render** attributes for Ajax controls. For successful updates, an element with the same identifier as that of the response must exist on the page. To work around this, include a placeholder (an empty element) during the initial creation of the page.
- » JSF 2 does not allow resources such as JavaScript or Cascading Style Sheets (CSS) to be added if the element requiring the resource is not initially present in the JSF tree. Components added to the tree with Ajax must have any required resources pre-loaded. In RichFaces, any components added to the JSF tree require components with corresponding resources included on the main page. To facilitate this, components can use the **rendered="false"** setting to not be rendered on the page.
- » JSF renders resource links (stylesheets, scripts) in order of occurrence. Adding **<h: outputStylesheet>** to the **<h: head>** section causes JSF to render it before the RichFaces resource links (dependencies of RichFaces components). To enable overwriting of RichFaces stylesheets and re-use RichFaces JavaScript implementation, render **<h: outputStylesheet target="head">** to the **<h: body>** on the end of the section.
- » Switching RichFaces skins with Ajax during runtime can slow your system down, as this requires reloading all the stylesheets.

[Report a bug](#)

## Chapter 5. Exploring Examples that use RichFaces

### 5.1. Archetypes

A Maven archetype is a template for creating projects. Maven uses an archetype to generate a populated directory for a project and create **pom.xml** files that contain build instructions.

The RichFaces Component Development Kit includes a Maven archetype named **richfaces-archetype-simpleapp** for generating the basic structure and requirements for a RichFaces application project. Maven can obtain the archetype from the JBoss repository at <https://repository.jboss.org/nexus/content/groups/public/>. The archetype is also included with the RichFaces source code in the **archetypes** directory.

You can generate a new RichFaces project with Maven by using the **richfaces-archetype-simpleapp** archetype. For instructions, see the JBoss Web Framework Kit Exploring the Examples Guide.

[Report a bug](#)

### 5.2. Quickstarts

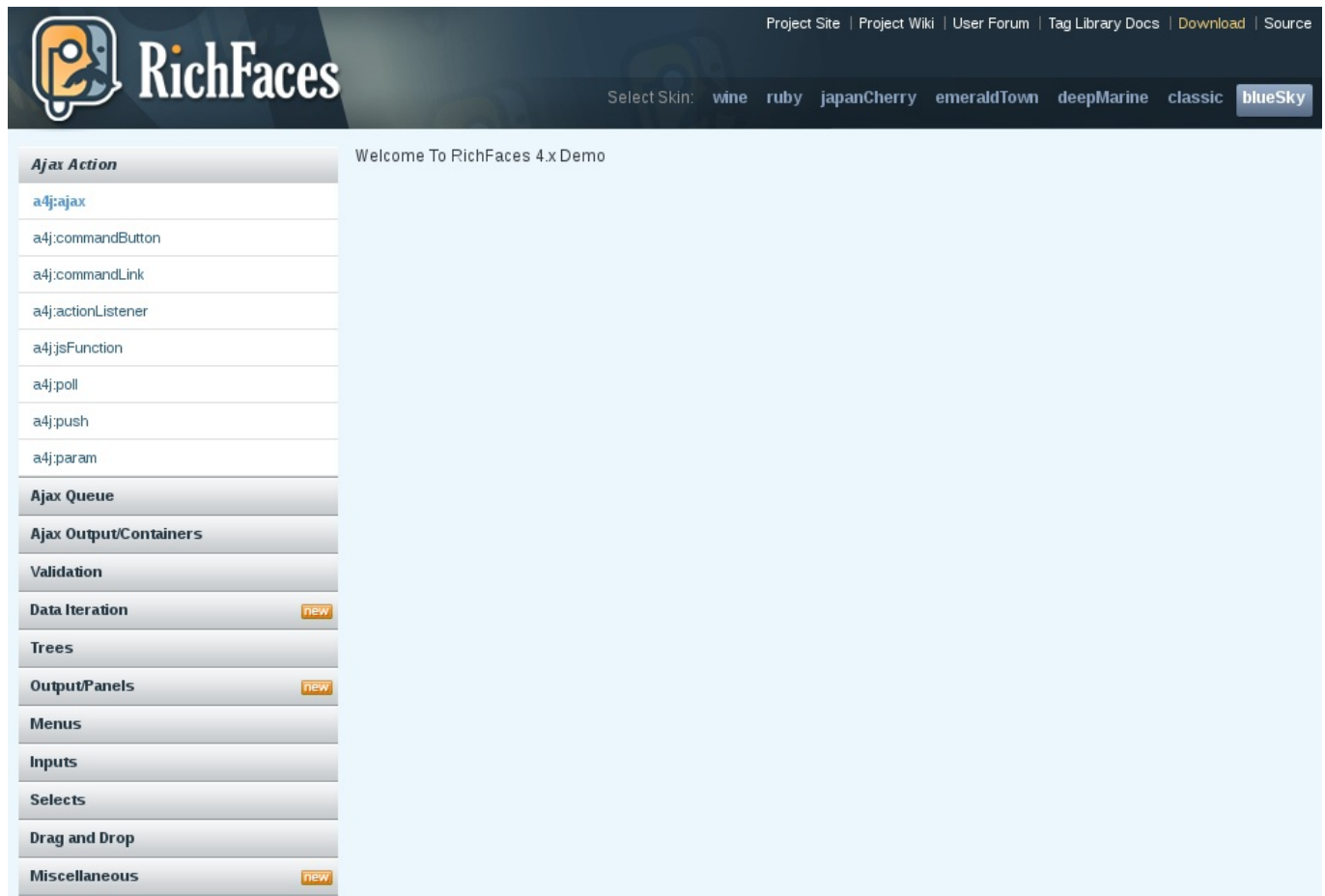
Quickstarts are pre-loaded projects that are used to demonstrate technologies and provide users with the knowledge to start building their own projects. For more information, see [Install Red Hat JBoss Web Framework Kit: Use the Examples](#).

[Report a bug](#)

### 5.3. RichFaces Showcase

Visit the [RichFaces Showcase](#) website to see different tags and online demonstrations of features. It includes a component library with a growing list of functions you can add to your project and brief descriptions of what they do.





**Figure 5.1. RichFaces Showcase homepage**

[Report a bug](#)

## 5.4. RichFaces PhotoAlbum

The RichFaces PhotoAlbum example is included as part of the Red Hat JBoss Web Framework Kit distribution. You can access the source code by downloading the richfaces-examples.zip file on the Red Hat Customer Portal. To view the PhotoAlbum example, build the source code and deploy it to the Red Hat JBoss Enterprise Application Platform.

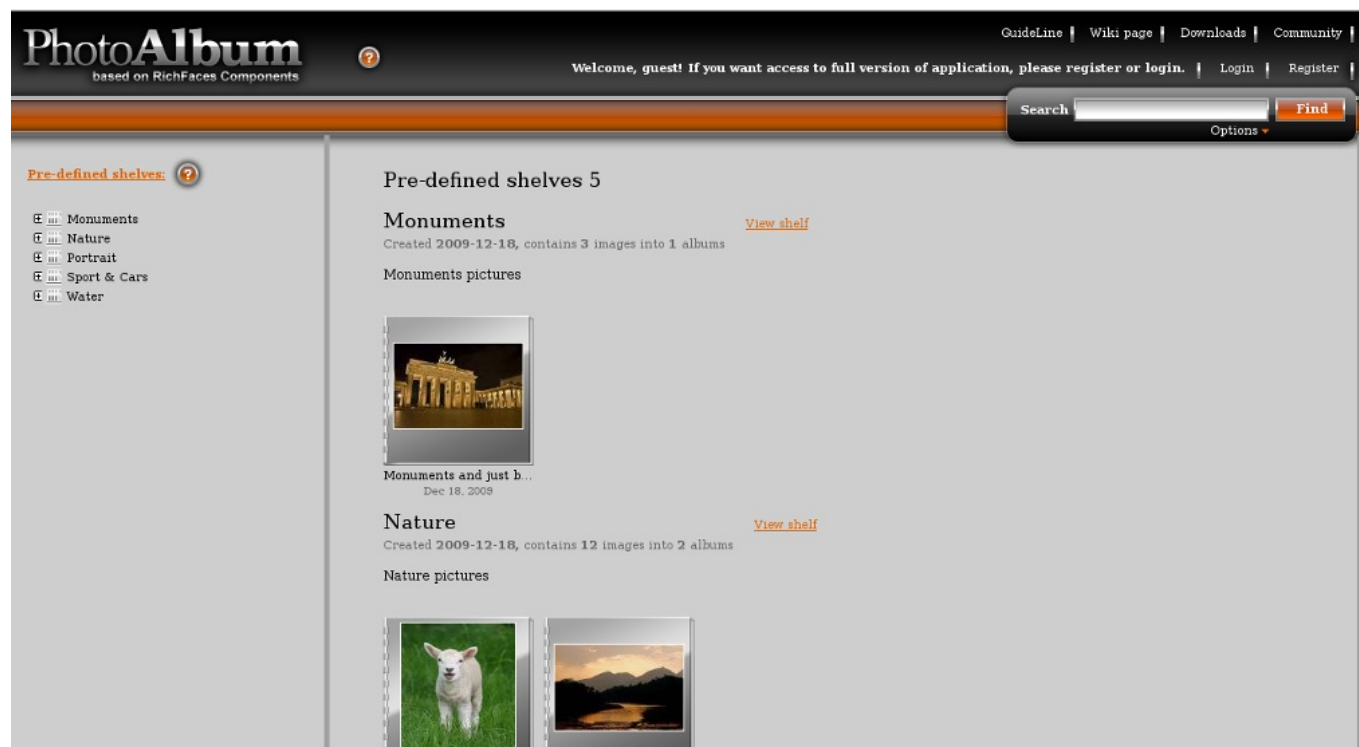


Figure 5.2. RichFaces PhotoAlbum homepage

[Report a bug](#)

## Chapter 6. Developing Applications with RichFaces

RichFaces has a range of components for developing web application interfaces. This chapter contains information on a number of key components that are widely used, as well as those that are new, categorized based on functionality. The code snippets and images presented in the chapter examples are taken from the RichFaces Showcase example, available to download from the Red Hat Customer Portal as part of the JBoss Web Framework Kit distribution.

[Report a bug](#)

### 6.1. Components for Web Application Layout

RichFaces has many different components for building the basic layout of web applications. Depending on their purpose, they can be used individually or as part of a greater body of HTML code.

[Report a bug](#)

#### 6.1.1. `a4j:commandButton`

The **`a4j:commandButton`** component creates a button that can be clicked to produce output. Users can type text into an appropriate field and click the button to receive a response.

Text can be added to the button:

- ✧ The **`value`** parameter is where the text that appears on the button is defined.

##### Example 6.1. `a4j:commandButton`

In this example, a button with the text "Say Hello" is created that renders a greeting when clicked.

```
<a4j:commandButton value="Say Hello" render="out" execute="@form" />
```

Using the **`a4j:commandButton`** component as shown in the above code creates this result:

Name:

**Hello John Citizen !**

[Report a bug](#)

#### 6.1.2. `rich:accordion`

The **`rich:accordion`** component provides a compact layout for panels. A panel displays as a heading that expands when a user clicks it, while the other panels remain collapsed. This is a minimalist way to display menu items.

You can customize how the menu displays:

- ✧ The **`rich:accordionItem`** component is where each menu item is defined.

##### Example 6.2. `rich:accordion`

This example creates an expandable menu with two options.

```
<rich:accordion switchType="client">
  <rich:accordionItem header="Overview:">
    <h:graphicImage value="/images/icons/common/rf.png"
style="float:right" alt="rf" />
    RichFaces is a component library for JSF and an advanced framework
for
    easily integrating AJAX capabilities into business applications.
    <ul>
      <li>100+ AJAX enabled components in two libraries</li>
      <li>a4j: page centric AJAX controls</li>
      <li>rich: self contained, ready to use components</li>
      <li>Whole set of JSF benefits while working with AJAX</li>
      <li>Skinnability mechanism</li>
      <li>Component Development Kit (CDK)</li>
      <li>Dynamic resources handling</li>
      <li>Testing facilities for components, actions, listeners, and
pages</li>
      <li>Broad cross-browser support</li>
      <li>Large and active community</li>
    </ul>
  </rich:accordionItem>
  <rich:accordionItem header="JSF 2 and RichFaces 4:">
    <p>We are working hard on RichFaces 4.0 which will have full JSF 2
integration. That is not all though, here is a summary of updates and
features:</p>
    <ul>
      <li>Redesigned modular repository and build system.</li>
      <li>Simplified Component Development Kit with annotations, faces-
config extensions, advanced templates support and more.</li>
      <li>Ajax framework improvements extending the JSF 2
specification.</li>
      <li>Component review for consistency, usability, and redesign
following semantic HTML principles where possible.</li>
      <li>Both server-side and client-side performance optimization.</li>
      <li>Strict code clean-up and review.</li>
    </ul>
  </rich:accordionItem>
</rich:accordion>
```

Using the **rich:accordion** component as shown in the above code creates this result:

**Overview:****JSF 2 and RichFaces 4:**

We are working hard on RichFaces 4.0 which will have full JSF 2 integration. That is not all though, here is a summary of updates and features:

- Redesigned modular repository and build system.
- Simplified Component Development Kit with annotations, faces-config extensions, advanced templates support and more..
- Ajax framework improvements extending the JSF 2 specification.
- Component review for consistency, usability, and redesign following semantic HTML principles where possible.
- Both server-side and client-side performance optimization.
- Strict code clean-up and review.

[Report a bug](#)

### 6.1.3. rich:toolbar

The **rich:toolbar** component creates a horizontal panel with clickable items. It can be used to add a toolbar to a layout with menu options that allow users to create, edit, and save input.

You can choose to customize your tools:

- ✧ The **rich:toolbarGroup** component is where each menu tool is defined.

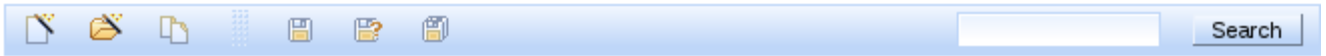
#### Example 6.3. rich:toolbar

This example creates a basic toolbar for word processing.

```
<rich:toolbar height="26" itemSeparator="grid">
  <rich:toolbarGroup>
    <h:graphicImage value="/images/icons/create_doc.gif" styleClass="pic"
alt="create_doc" />
    <h:graphicImage value="/images/icons/create_folder.gif"
styleClass="pic" alt="create_folder" />
    <h:graphicImage value="/images/icons/copy.gif" styleClass="pic"
alt="copy" />
  </rich:toolbarGroup>
  <rich:toolbarGroup>
    <h:graphicImage value="/images/icons/save.gif" styleClass="pic"
alt="save" />
    <h:graphicImage value="/images/icons/save_as.gif" styleClass="pic"
alt="save_all" />
    <h:graphicImage value="/images/icons/save_all.gif" styleClass="pic"
alt="save_all" />
  </rich:toolbarGroup>
  <rich:toolbarGroup location="right">
    <h:inputText styleClass="barsearch" />
  </rich:toolbarGroup>
</rich:toolbar>
```

```
<h:commandButton styleClass="barsearchbutton" onclick="return
false;" value="Search" />
</rich:toolbarGroup>
</rich:toolbar>
```

✧ Using the **rich:toolbar** component as shown in the above code creates this result:



[Report a bug](#)

#### 6.1.4. a4j:ajax

The **a4j:ajax** component triggers an Ajax request when an event is triggered. For example, when a user inputs a term into a text box, it can be replicated elsewhere on the page to display a rendered version of the text on the webpage.

The component can be configured to respond to user actions:

- ✧ The **event** field indicates the action that is performed to trigger the output.

##### Example 6.4. a4j:ajax

This example takes the information a user types in the text field and renders it on the webpage.

```
<a4j:ajax event="keyup" render="out" />
```

Using the **a4j:ajax** component as shown in the above code creates this result:

[Report a bug](#)

## 6.2. Components Using Back-End Data

Components using back-end data pull information from a server, instead of that information being implicit in the source. This reduced coding and allows webpages to load faster, because they do not have load that information for every session.

[Report a bug](#)

#### 6.2.1. rich:tree

The **rich:tree** component creates a panel that displays interactive file trees. Users can click on the files to select and expand them to view their contents.

The folders in the tree component can be configured:

- ✧ The **<rich:tree id="">** field is where the name of the file tree is specified.
- ✧ The **<rich:treeNode type="">** field is where a description of the file itself is specified.

### Example 6.5. rich:tree

This example creates a file tree organised by country, with each country's folder containing a list of local music producers.

```
<rich:tree id="tree" nodeType="#{node.type}" var="node" value="#{treeBean.rootNodes}" toggleType="client" selectionType="ajax" selectionChangeListener="#{treeBean.selectionChanged}">
  <rich:treeNode type="country">
    #{node.name}
  </rich:treeNode>
  <rich:treeNode type="company" iconExpanded="/images/tree/disc.gif" iconCollapsed="/images/tree/disc.gif">
    #{node.name}
  </rich:treeNode>
  <rich:treeNode type="cd" iconLeaf="/images/tree/song.gif">
    #{node.artist} - #{node.name} - #{node.year}
  </rich:treeNode>
</rich:tree>
```

Using the **rich:tree** component as shown in the above code creates this result:



[Report a bug](#)

### 6.2.2. rich:autocomplete

The **rich:autocomplete** component responds to user input by pulling data from the server. When a user starts typing a term into a search field, the component responds by auto-completing the word. A drop-down list of search terms starting with the same letter(s) is displayed as a result.

There are a number of options for customizing the autocomplete component:

- ✱ The **rich:autocomplete** component indicates the action.





```

        </f:facet>
        <h:outputText value="#{car.vendor}"/>
    </rich:column>
    <rich:column>
        <f:facet name="header">
            <h:outputText value="Model"/>
        </f:facet>
        <h:outputText value="#{car.model}"/>
    </rich:column>
    <rich:column>
        <f:facet name="header">
            <h:outputText value="Price"/>
        </f:facet>
        <h:outputText value="#{car.price}"/>
    </rich:column>
    <rich:column>
        <f:facet name="header">
            <h:outputText value="Mileage"/>
        </f:facet>
        <h:outputText value="#{car.mileage}"/>
    </rich:column>
    <rich:column>
        <f:facet name="header">
            <h:outputText value="VIN Code"/>
        </f:facet>
        <h:outputText value="#{car.vin}"/>
    </rich:column>
    <rich:column>
        <f:facet name="header">
            <h:outputText value="Items stock"/>
        </f:facet>
        <h:outputText value="#{car.stock}"/>
    </rich:column>
    <rich:column>
        <f:facet name="header">
            <h:outputText value="Days Live"/>
        </f:facet>
        <h:outputText value="#{car.daysLive}"/>
    </rich:column>
</rich:extendedDataTable>

```

Using the **rich:extendedDataTable** component as shown in the above code creates this result:

Cars marketplace				
vendor	Model	Price	Mileage	VIN Code
Chevrolet	Corvette	44561	14835.0	GGFTSIODSTDE
Chevrolet	Corvette	26864	35132.0	RUWMRZVZZLF
Chevrolet	Corvette	43541	19499.0	MUAJSTHCGJJH
Chevrolet	Corvette	32386	65295.0	YPYODZGFQES
Chevrolet	Corvette	26618	32479.0	BOSMTYZYFBV
Chevrolet	Malibu	27641	28178.0	MTMETZWZUNH
Chevrolet	Malibu	38430	21183.0	RPJAABPOYYC
Chevrolet	Malibu	23086	43714.0	ILOERVEXVEWI
Chevrolet	Malibu	18628	70425.0	TQYKTXYQMGY
Chevrolet	Malibu	31439	53006.0	YCSMVLZEAQE
Chevrolet	Malibu	49898	5242.0	XSOEWBPURZF
Chevrolet	Malibu	51659	54237.0	HJSDCTIDJRAM

[Report a bug](#)

## 6.3. Components using User-Input Data

These components produce output based on a user's actions. The component provides a response that appears as output on the webpage.

[Report a bug](#)

### 6.3.1. rich:inputNumberSlider

The **rich:inputNumberSlider** component allows users to select a number from a pre-defined range. It is rendered as a slider which the user can move to select a number, with the option to type a specific value into an input field. This dual method of specifying a quantity is useful for web forms.

The range of the slider can be defined:

- ✧ The **value** field is defined so the slider displays a value of 50. The default range of the slider is 1 to 100.

#### Example 6.8. rich:inputNumberSlider

This example creates a scale that ranges from 1 to 100 with the arrow automatically set to 50.

```
<rich:inputNumberSlider value="50" />
```

Using the **rich:inputNumberSlider** component as shown in the above code creates this result:



[Report a bug](#)

### 6.3.2. `a4j:mediaOutput`

The `a4j:mediaOutput` component displays multimedia objects as defined by the user. It can be used to add an image to a layout and give users the option to resize it to a predefined resolution.

The acceptable file types can be defined in the code:

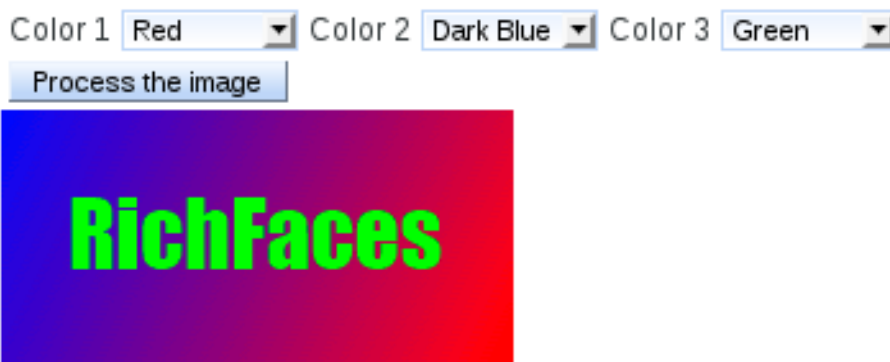
- ✧ The `element` field specifies the file type. (In this case, an image file.)
- ✧ The `mimeType` field specifies the file format. (In this case, a JPEG file.)

#### Example 6.9. `a4j:mediaOutput`

This example creates an embedded image that users can modify the colours of.

```
<a4j:mediaOutput element="img" cacheable="false" session="true"
id="img" createContent="#{mediaBean.process}"
value="#{mediaData}" mimeType="image/jpeg" alt="img" />
```

Using the `a4j:mediaOutput` component as shown in the above code creates this result:



[Report a bug](#)

### 6.3.3. `rich:focus`

The `rich:focus` component alerts users to an item in the panel they are using. For example, if a user types an invalid term in a web form, a message displays next to the text field stating that alternate text must be entered before submitting the form.

The `rich:focus` component has various uses:

- ✧ Provides a visual alert to users.
- ✧ Is applied to every field that is triggered.
- ✧ Does not allow validation until the correct input is added.

#### Example 6.10. `rich:focus`

This example can be added at the beginning of a body of HTML to display an alert to users if they have used an invalid term in a text field.

```
<rich:focus />
```

Using the **rich: focus** component shown above creates this result:

[Report a bug](#)

## 6.4. Components for Advanced Web Application Functionality

Rich components can be used to create an interactive webpage with added options. They provide a visual interface where users can upload files, use a rich editor to create content, and undertake other editing functionalities without needing to input code.

[Report a bug](#)

### 6.4.1. rich:fileUpload

The **rich:fileUpload** component enables upload of images from a user's machine to the server.

There are a number of options for customizing the fileUpload component:

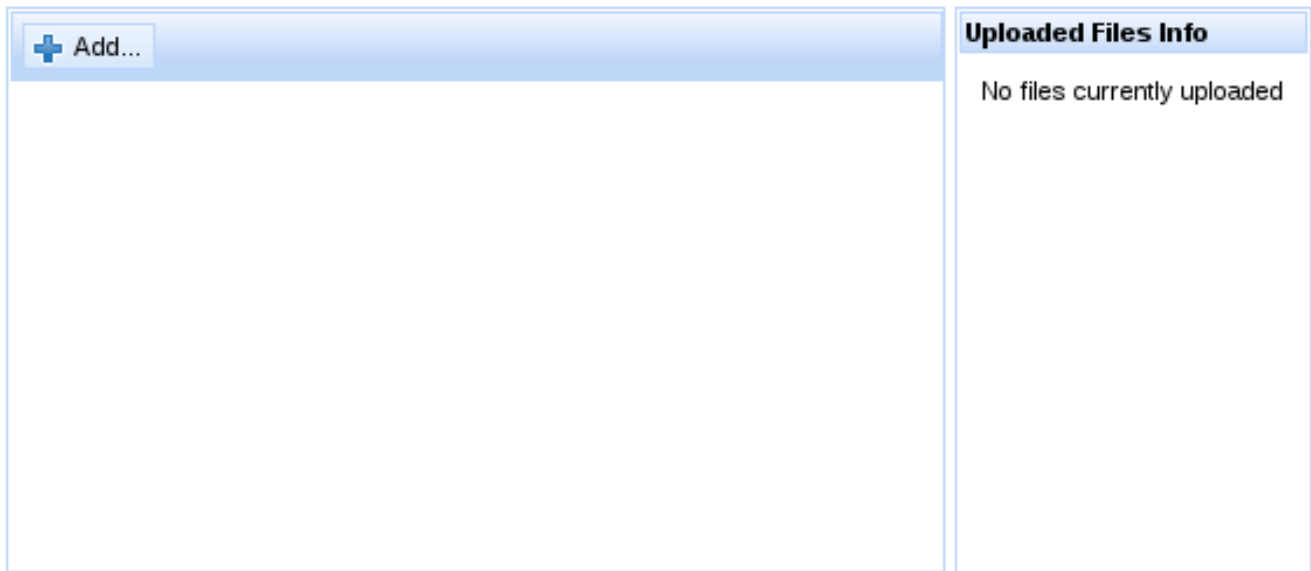
- ✧ The **id** parameter defines the name of the action.
- ✧ The **acceptedTypes** parameter defines which file formats are able to be uploaded. (In this case, only images are allowed.)
- ✧ The **maxFilesQuantity** attribute defines a maximum number of files that can be uploaded in one attempt.
- ✧ The **a4j:ajax** is an Ajax component that displays a message indicating when the upload is complete.

#### Example 6.11. rich:fileUpload

This example demonstrates usage of the **rich:fileUpload** component.

```
<rich:fileUpload fileUploadListener="#{fileUploadBean.listener}"
  id="upload" acceptedTypes="jpg, gif, png, bmp"
  ontyperejected="alert('Only JPG, GIF, PNG and BMP files are
  accepted');" maxFilesQuantity="5">
  <a4j:ajax event="uploadcomplete" execute="@none" render="info" />
</rich:fileUpload>
```

The **rich:fileUpload** component can be used in conjunction with a file upload table such as that illustrated here.



[Report a bug](#)

### 6.4.2. rich:editor

The **rich:editor** component creates a text editor where users can create, edit, and save documents for the web. It provides a toolbar, menu items, and a space for the body of the document to be written. It negates the need for a user to know HTML, because what is entered in the editor is what is displayed on the webpage.

There are a number of options for customizing the editor component:

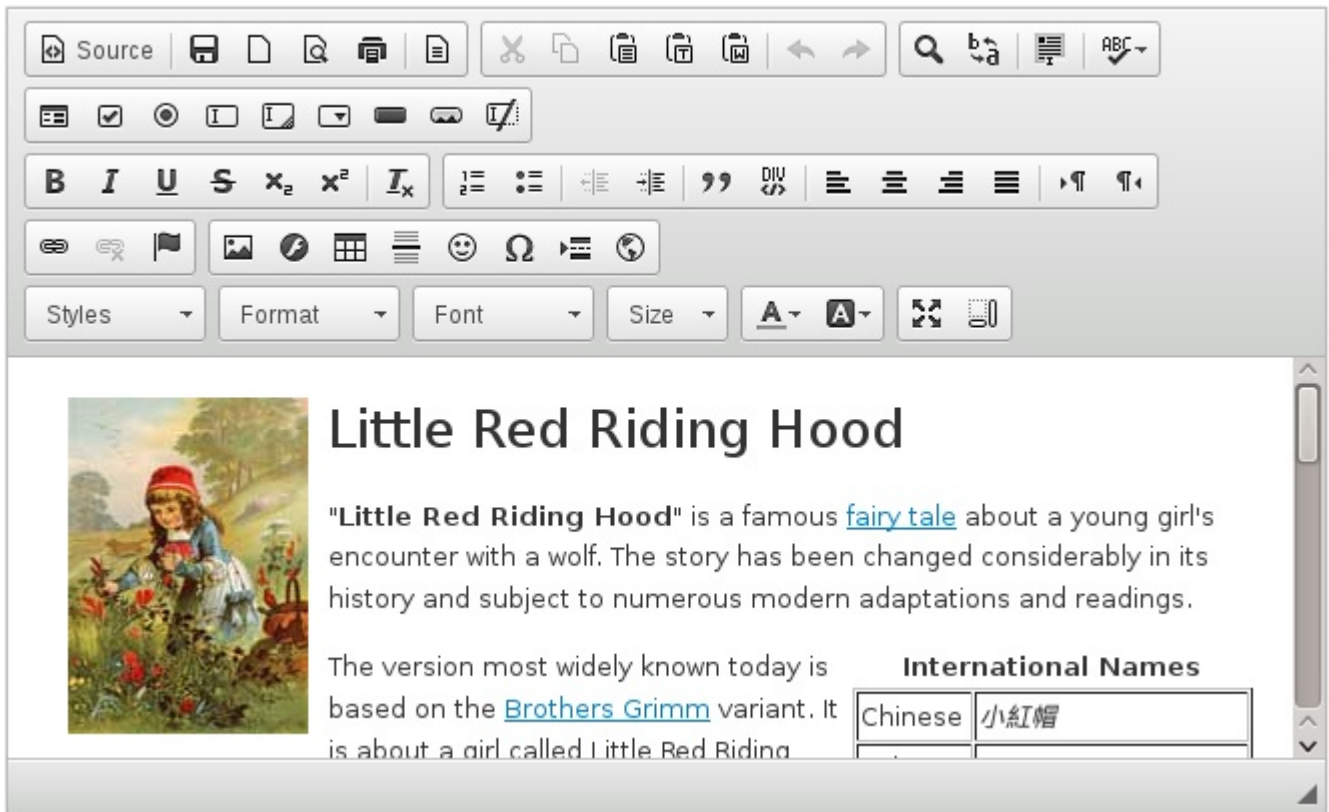
- » The `<a4j:attachQueue requestDelay="1000" />` value can be changed to wait a certain number of milliseconds (in this case 1000) before an Ajax update of output panel.
- » The **toolbar** field can be set as either **basic** (normal editing features) or **full** (advanced editing features).

#### Example 6.12. rich:editor

This example creates an editor capable of accepting a variety of content including images, hyperlinks, and tables.

```
<rich:editor id="editor" toolbar="full" value="#{editorBean.value}"
style="margin-bottom: 1em">
    <a4j:ajax event="change" render="panel"
status="panelUpdateStatus" />
    <a4j:ajax event="dirty" render="panel"
status="panelUpdateStatus">
        <a4j:attachQueue requestDelay="1000" />
    </a4j:ajax>
</rich:editor>
```

Using the **rich:editor** component as shown in the above code creates this result:



[Report a bug](#)

### 6.4.3. rich:tabPanel

The **rich:togglePanel** component creates a panel with clickable tabs to switch between views. Each view can be used for separate information. It is a compact method of allowing users to access different sections of information in one place by clicking on the tab they wish to view.

Different views can be added to the panel:

- ✧ The **<rich:togglePanelItem>** component is where the information for that particular view is specified.
- ✧ Add and configure a new **<rich:togglePanelItem>** component for each view.

#### Example 6.13. rich:togglePanel

This example creates a panel with two tabs, providing users with the ability to switch between two views.

```
<rich:togglePanel id="panel1" activeItem="item1" render="tabs"
itemChangeListener="#{panelMenuBean.updateCurrent}">
  <rich:togglePanelItem name="item1">
    <p>This toggle panel switches in Ajax mode. So only one active item
loaded to the client.</p>

    <p>For now you are at Panel 1</p>
  </rich:togglePanelItem>
  <rich:togglePanelItem name="item2">
    <p>After the second link click - panel changed active item to the
```

```
second one according to name specified in the togglePanelBehavior</p>
    <p>For now you are at Panel 2</p>
</rich:togglePanelItem>
</rich:togglePanel>
```

Using the **rich:tabPanel** component as shown in the above code creates this result:

This toggle panel switches in Ajax mode. So only one active item loaded to the client.

For now you are at Panel 1

**Toggle Panel Item 1** Toggle Panel Item 2

[Report a bug](#)

## 6.5. Components for Skinning

### 6.5.1. About Skins

Application skins are used with the RichFaces framework to change the appearance of an application by setting the visual elements of controls and components. Typically the appearance of web applications is handled through CSS files associated with the application, but skinning allows CSS settings to be easily edited. Skins consist of a small, generalized set of font and color parameters that can be applied to multiple different styles. This avoids repetitive coding and duplication in CSS files. CSS files are not completely replaced: skins work as a high-level extension to standard CSS.

Each skin has a set of **skin-parameters** for defining theme palette and other elements of the user interface. These parameters work together with regular CSS declarations and can be referred to from within CSS using JavaServer Faces Expression Language (EL).

Skins can be changed at runtime so users can personalize an application's appearance on the fly.

[Report a bug](#)

### 6.5.2. Using Predefined Skins

RichFaces includes the following predefined skins:

- ✧ **DEFAULT**
- ✧ **plain**, which contains no skin parameters and is intended for embedding RichFaces components into existing projects with their own styles.
- ✧ **emeraldTown**
- ✧ **blueSky**
- ✧ **wine**
- ✧ **japanCherry**
- ✧ **ruby**
- ✧ **classic**
- ✧ **deepMarine**

To add one of these skins to your application, add the **org.richfaces.SKIN** context parameter to the **web.xml** configuration file:

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

[Report a bug](#)

### 6.5.3. Using Custom Skins

There are many ways to modify skins to suit the style of your layout. Custom skins can be used to attain total control over how the layout appears.

RichFaces skins are implemented using the following three-level scheme:

#### Component stylesheets

Stylesheets are provided for each component. CSS style parameters map to skin parameters defined in the skin property file. This mapping is implemented using ECSS files.

#### Skin property files

Skin property files map skin parameters to constant styles. Skin properties are defined in **skin.properties** files.

#### Custom style classes

Individual components can use the **styleClass** attribute to redefine specific elements. These components then use the styles defined in a CSS file instead of the standard look for components as defined by the ECSS stylesheets.

[Report a bug](#)

#### 6.5.3.1. Modifying Predefined Skins

Skins in RichFaces can be customized in several ways:

##### Skin property files

Application interfaces can be modified by altering the values of skin parameters in the skin itself. Edit the constant values defined in the **skin.properties** file to change the style of every component mapped to that skin property.

##### Component stylesheets

Mappings and other style attributes listed in a component's ECSS file can be edited. Edit the ECSS file to change the styles of all components of that type.

##### Custom components style classes

Individual components can use the **styleClass** attribute to use a unique style class. Add the new style class to the application CSS and reference it from an individual component with the **styleClass** attribute.

##### Overwriting stylesheets in application



Load custom stylesheets using `<h:outputStylesheet>`, which rewrites and extends styles defined for style classes of components.



## Note

To extend or overwrite style sheet definitions with your own stylesheets, ensure their definitions are placed to be rendered in the correct order of occurrence.

[Report a bug](#)

### 6.5.3.2. Creating a New Skin

#### Procedure 6.1. Creating a New Skin

1. Create a new text file and name it following the format **new\_skin\_name.skin.properties**. Save it to either your project's **/WEB-INF/classes** directory.

2. **Define the skin constants**

- a. Add skin parameter constants and values to the file.

#### Example 6.14. blueSky.skin.properties file

Open the **/META-INF/skins/blueSky.skin.properties** file in the *richfaces-impl-version.jar* package. This file lists the constants and values of the skin.

You can use the **blueSky.skin.properties** file as a template for your new skin.

- b. Instead of redefining an entire new skin, your skin can use an existing skin as a base on which to build new parameters. Specify a base skin by using the **baseSkin** parameter in the skin file.

#### Example 6.15. Using a base skin

This example takes the **blueSky** skin as a base and only changes the **generalSizeFont** parameter.

```
baseSkin=blueSky
generalSizeFont=12pt
```

3. To add the skin, navigate to the project's home directory and open the **web.xml** settings file. Add a **<context-param>** property and the new skin's filename to the settings file:

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

4. Save the **web.xml** settings file and exit. The new skin loads when you build the project.

[Report a bug](#)

### 6.5.3.3. Using Round Corners

Support for adding round borders to skins is available with the **panelBorderRadius** skin parameter. The value of this parameter maps to the CSS 3 **border-radius** property. This CSS 3 property is ignored in older browsers, and the skin reverts to square corners.

Units of the **panelBorderRadius** skin parameter must be either **px** (pixels) or **%** (a percentage). The greater the pixels or percentage, the more rounded the edges are.

[Report a bug](#)

### 6.5.3.4. Adding Extra Skinning Functionality with ECSS

RichFaces uses ECSS files to add extra functionality to the skinning process. ECSS files are CSS files which use Expression Language (EL) to connect styles with skin properties.

#### Example 6.16. ECSS style mappings

The ECSS code for the **<rich:panel>** component contains styles for the panel and its body:

```
.rf-p{
    background-color: '#{richSkin.generalBackgroundColor}';
    color: '#{richSkin.panelBorderColor}';
    border-width: 1px;
    border-style: solid;
    padding: 1px;
}

.rf-p-b{
    font-size: '#{richSkin.generalSizeFont}';
    color: '#{richSkin.generalTextColor}';
    font-family: '#{richSkin.generalFamilyFont}';
    padding: 10px;
}
```

#### . rf-p defines the panel styles

- ✦ The background-color CSS property maps to the **generalBackgroundColor** skin parameter.
- ✦ The color CSS property maps to the **panelBorderColor** skin parameter.

#### . rf-p-b defines the panel body styles

- ✦ The font-family CSS property maps to the **generalFamilyFont** skin parameter.
- ✦ The font-size CSS property maps to the **generalSizeFont** skin parameter.
- ✦ The color CSS property maps to the **generalTextColor** skin parameter.

[Report a bug](#)

### 6.5.4. Enabling Skin Changes when Applications are Running

## Procedure 6.2. Enabling Skin Changes During Runtime

1. To enable changing skins during runtime, a new skin bean is required. Add the following to your skin:

```
public class SkinBean {

    private String skin;

    public String getSkin() {
        return skin;
    }

    public void setSkin(String skin) {
        this.skin = skin;
    }

}
```

### 2. Reference the skin bean

Add the **@ManagedBean** and **@SessionScoped** references to the class.

- A. Alternatively, use EL to reference the skin bean from the **web.xml** settings file.

```
<context-param>
    <param-name>org.richfaces.skin</param-name>
    <param-value>#{skinBean.skin}</param-value>
</context-param>
```

### 3. Set initial skin

The application needs an initial skin to display before the user chooses an alternative skin. Specify the skin in your class with **@ManagedProperty**.

```
@ManagedProperty(value="blueSky")
private String skin;
```

- A. Alternatively, specify the initial skin in the **web.xml** settings file.

```
<managed-bean>
    <managed-bean-name>skinBean</managed-bean-name>
    <managed-bean-class>SkinBean</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
    <managed-property>
        <property-name>skin</property-name>
        <value>blueSky</value>
    </managed-property>
</managed-bean>
```

[Report a bug](#)

## 6.5.5. Applying Skins to Standard HTML Controls

Standard HTML controls used alongside RichFaces components are themed to create a cohesive user interface.

## Automatic skinning

Skinning style properties are automatically applied to controls based on their element names and attribute types. If the HTML elements are referenced in the standard skin stylesheets, the controls are styled according to the mapped skin properties.

Standard HTML controls are skinned in this way by default. To override this behavior and prevent the RichFaces skins from being applied to the standard HTML controls, set the **org.richfaces.enableControlSkinning** context parameter in the **web.xml** configuration file to **false**:

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

## Skinning with the rfs-ctn class

The skinning style properties are determined through a separate CSS class. This method is not available by default, but is enabled through the **org.richfaces.enableControlSkinningClasses** context parameter in the **web.xml** settings file:

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

When enabled, a stylesheet with predefined classes generates a CSS class named **rfs-ctn**. Reference the **rfs-ctn** class from any container element (such as a **<div>** element) to skin all the standard HTML controls in the container.

Standard HTML controls can also be specifically defined in the CSS. Refer to the **/core/impl/src/main/resources/META-INF/resources/skinning\_both.ecss** file in the *richfaces-ui.jar* package for examples of specially-defined CSS classes with skin parameters for HTML controls.

[Report a bug](#)

## Chapter 7. Using RichFaces for Mobile Applications

### 7.1. Configuring a RichFaces Application for Mobile Devices

#### Procedure 7.1. Configuring a RichFaces Application for Mobile Devices

1.



#### Important

Ensure you copy the physical .js and .css files locally before developing your application.

To combine the out-of-the-box JavaScript and CSS code into a few files, add the following code to your project's **web.xml** file:

```
<context-param>
  <param-name>org.richfaces.resourceOptimization.enabled</param-
name>
  <param-value>true</param-value>
</context-param>
```

- To initialize the JavaScript and CSS which allows RichFaces to function on a mobile device, add the following code to your template:

```
<link rel="stylesheet"
href="https://raw.githubusercontent.com/richfaces/components/develop/mobile-
compatibility/slidfast.css" />

<script type="text/javascript"
src="https://raw.githubusercontent.com/richfaces/components/develop/mobile-
compatibility/slidfast.js"></script>

<script type="text/javascript">
slidfast({
    defaultPageID: 'home-page',
    defaultPageHash: 'home',
    backButtonID: 'back-button'
});
</script>
```

The snippet sets up the mandatory default parameters to get your application running.

#### **defaultPageID**

The actual ID of the HTML element (div) surrounding your code for the home page.

#### **defaultPageHash**

This can be any valid URL character set and is the bookmark and identifier for your home page.

**backButtonID**

The ID attribute of the application's back button. This parameter is optional since the mobile JavaScript (slidfast) already accounts for use of the browser back and forward buttons.

3. After initializing the mobile front end, you must ensure your markup is in proper order. At a minimum, you are required to adhere to the following markup structure. This provides the framework used for testing and debugging the RichFaces mobile showcase.

```
<h:body>
  <!--Include the CSS code after the body tag to ensure you
  overwrite dynamic styles-->
  <link rel="stylesheet"
href="https://raw.githubusercontent.com/richfaces/components/develop/mobile-
compatibility/rf-mobile-skin.css" />
  <div id="browser">
    <header>...</header>
    <h:form>
      <div id="page-container">
        <div id="home-page">
          ... JSF and HTML markup
        </div>
        <div id="component-page">
          ... JSF and HTML markup
        </div>
      </div>
    </h:form>
  </div>
</h:body>
```

**Note**

For a complete working example and to see how the page is setup, see the source of the RichFaces Mobile Showcase application template [here](#). Alternatively, run the RichFaces simpleapp archetype with the mobile-optimized switch set to true:

```
mvn archetype:generate \
  -DarchetypeGroupId=org.jboss.archetype.wfk \
  -DarchetypeArtifactId=richfaces-archetype-simpleapp \
  -DarchetypeVersion=2.7.0.Final \
  -Dmobile-optimized=true
```

[Report a bug](#)

## 7.2. Viewport Metadata

The **viewport** is the area in which your webpage is drawn. To define the properties of the viewport for your webpage:

1. Use the "viewport" property in an HTML <meta> tag. Place this in the document <head>.

2. Define multiple viewport properties in the content attributes of the <meta> tag.

You can define the following options:

- » The height and width of the viewport
- » The initial scale of the page
- » The target screen density

Here is the viewport setting used in the RichFaces Mobile Showcase:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0,
minimum-scale=1.0, maximum-scale=1.0"/>
```

This resizes the application when rotated to landscape or portrait mode and zooms the content appropriately.

Other device browsers may support additional metadata. For example, Mobile Safari supports Apple specific Meta-Tag keys which allow you to set full screen mode and adjustments to other visual elements.

```
<!-- enable full-screen mode -->
<meta name="apple-mobile-web-app-capable" content="yes"/>
<!-- controls the appearance of the status bar in full-screen mode -->
<meta name="apple-mobile-web-app-status-bar-style" content="black"/>
```

[Report a bug](#)

## 7.3. JavaScript

The RichFaces Mobile Showcase uses as much screen real estate as possible when run in the device browser. To automatically hide the browser URL bar, the code uses the following JavaScript onload:

```
function hideURLbar()
{
    setTimeout(scrollTo, 0, 0, 1);
}
```

[Report a bug](#)

## 7.4. MobileESP

MobileESP is an API detects if a user is accessing a website from a mobile device. This results in loading mobile versions of webpages as opposed to loading the (often bulkier) desktop versions. The RichFaces Mobile Showcase uses server-side MobileESP. The following is a custom MobileESP processor with CDI integration:

```
@ManagedBean(name="userAgent")
@SessionScoped
public class UserAgentProcessor implements Serializable {

    private static final long serialVersionUID = 1L;
```

```

private UAgentInfo uAgentInfo;

@PostConstruct
public void init() {
    FacesContext context = FacesContext.getCurrentInstance();
    HttpServletRequest request = (HttpServletRequest)
context.getExternalContext().getRequest();
    String userAgentStr = request.getHeader("user-agent");
    String httpAccept = request.getHeader("Accept");
    uAgentInfo = new UAgentInfo(userAgentStr, httpAccept);
}

public boolean isPhone() {
    //Detects a whole tier of phones that support similar
functionality as the iphone
    return uAgentInfo.detectTierIphone();
}

public boolean isTablet() {
    // Will detect iPads, Xooms, Blackberry tablets, but not Galaxy -
they use a strange user-agent
    return uAgentInfo.detectTierTablet();
}

public boolean isMobile() {
    return isPhone() || isTablet();
}
}

```

RichFace's Facelets template then determines which page should be shown through Expression Language:

```

<f:view
    xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:c="http://java.sun.com/jsp/jstl/core"
    xmlns:ui="http://java.sun.com/jsf/facelets">

    <c:choose>
        <c:when test="#{userAgent.phone}">
            <ui:include src="phoneHome.xhtml">
                [phone]
            </ui:include>
        </c:when>
        <c:when test="#{userAgent.tablet}">
            <ui:include src="phoneHome.xhtml">
                [tablet]
            </ui:include>
        </c:when>
        <c:otherwise>
            <ui:include src="desktopHome.xhtml">
                [desktop]
            </ui:include>
        </c:otherwise>
    </c:choose>

```



```
</c:choose>  
</f:view>
```

[Report a bug](#)

## Chapter 8. Migrating from RichFaces 4 to RichFaces 4.5

RichFaces 4.5.2 is distributed with this release of JBoss Web Framework Kit. Information is provided here for migrating applications from RichFaces 4 to RichFaces 4.5.

- ✦ A bug in Mojarra affects the **ExtendedPartialViewContext** in RichFaces 4.5. A check has been implemented to verify that you are running a patched version of Mojarra  $\geq 2.1.28$  or  $\geq 2.2.6$ . Red Hat JBoss Enterprise Application Platform 6.1 and later are not affected by this bug.
- ✦ RichFaces 4.5 introduces a new syntax to access the JavaScript API of components to avoid confusion with the jQuery **\$** global, as shown in the table below. A function alias is provided to ease the migration process, however using the deprecated **RichFaces.\$** syntax will generate a warning in the javascript console.

**Table 8.1.**

Before RichFaces 4.5	After RichFaces 4.5
<code>RichFaces.\$('componentIdInHtmlPage')</code>	<code>RichFaces.component('componentIdInHtmlPage')</code>

- ✦ RichFaces 4.5 extends the built-in sorting and filtering capabilities of the **ExtendedDataTable** to the simpler **DataTable** and **CollapsibleSubTable**. If you have existing applications using custom controls, you must either remove the custom controls in favor of the built-in controls or disable the built-in controls. For more information, see [RichFaces Component Reference Guide: Built-in filter controls](#).
- ✦ The **rich:fileUpload** component is no longer compatible with desktop browser Internet Explorer 9.
- ✦ If you use the RichFaces CDK to build custom components, you must change your maven configuration. The CDK maven build now requires a two step process:

```
<plugins>
  <plugin>
    <artifactId>maven-compiler-plugin</artifactId>
    <executions>
      <execution>
        <id>precompile-sources-for-cdk</id>
        <phase>generate-sources</phase>
        <goals>
          <goal>compile</goal>
        </goals>
      </execution>
    </executions>
  </plugin>
  <plugin>
    <groupId>org.richfaces.cdk</groupId>
    <artifactId>richfaces-cdk-maven-plugin</artifactId>
    <version>${org.richfaces.cdk.version}</version>
    <executions>
      <execution>
        <id>cdk-generate-sources</id>
        <phase>process-sources</phase>
```

```
    <goals>
      <goal>generate</goal>
    </goals>
  </execution>
</executions>
</plugin>
</plugins>
```

[Report a bug](#)

## Revision History

<b>Revision 2.7.0-1</b>	<b>Tues Jan 06 2015</b>	<b>Michelle Murray</b>
Generated for WFK 2.7 release		