Liferay Faces

Reference Documentation

3.1.0-ga1

Liferay Faces

Legal Notice

Chapter 1. Overview

Liferay Faces Alloy is a JAR that JSF developers can add as a dependency to their portlet WAR projects in order to utilize Alloy UI in a way that is consistent with JSF development.

The project home page can be found at http://www.liferay.com/community/liferay-projects/liferay-faces/alloy

Chapter 2. UIComponent Tags

Liferay Faces Alloy provides the following UIComponent tags as part of its component suite.

Table 2.1. UIComponent Tags

Tag	Description
`aui:button-row`	Renders a span element with CSS class name "aui-button-holder" that will be positioned by Alloy UI CSS.
`aui:column`	Renders a div element with CSS class name "aui-column" that will be positioned by Alloy UI CSS.
`aui:field`	Renders a span element with CSS class name "aui-field" that will be positioned by Alloy UI CSS.
`aui:fieldset`	Renders a fieldset element with CSS class name "aui-fieldset" that will be positioned by Alloy UI CSS.
`aui:form`	Renders a form element with CSS class name "aui-form" that will be positioned by Alloy UI CSS.
`aui:layout`	Renders a div element with CSS class name "aui-layout" that will be positioned by Alloy UI CSS.
`aui:list`	Renders a ul element with CSS class name "aui-list" that will be positioned by Alloy UI CSS.
`aui:list-item`	Renders an li element with CSS class name "aui-listitem" that will be positioned by Alloy UI CSS.
`aui:text-box-list`	Renders a div element with CSS class name "aui-textboxlist" that will be positioned by Alloy UI CSS.
`aui:text-box-list-item`	Renders an li element with CSS class name "aui-textboxlistentry" that will be positioned by Alloy UI CSS.

2.1. The aui:button-row tag

The `aui:button-row` tag renders a span element with CSS class name "aui-button-holder" that will be positioned by Alloy UI CSS.

Table 2.2. Attributes

Attribute	Type	Description	Required
id	String	The identifier of the component	false
cssClass	String	The name of a CSS class that is to be rendered within the class attribute.	false
rendered	Boolean	Boolean flag indicating whether or not this component is to be rendered during the RENDER_RESPONSE phase of the JSF lifecycle. The default value is “true”.	false
styleClass	String	The name of a CSS class that is to be rendered within the class attribute.	false

2.2. The aui:column tag

The `aui:column` tag renders a div element with CSS class name "aui-column" that will be positioned by Alloy UI CSS.

Table 2.3. Attributes

Attribute	Type	Description	Required
id	String	The identifier of the component	false
columnWidth	String	The width of the column.	false
cssClass	String	The name of a CSS class that is to be rendered within the class attribute.	false
first	Boolean	Boolean flag indicating whether or not this is the first column. The default value is “false”.	false
last	Boolean	Boolean flag indicating whether or not this is the last column. The default value is “false”.	false
rendered	Boolean	Boolean flag indicating whether or not this component is to be rendered during the RENDER_RESPONSE phase of the JSF lifecycle. The default value is “true”.	false
styleClass	String	The name of a CSS class that is to be rendered within the class attribute.	false

2.3. The aui:field tag

The `aui:field` tag renders a span element with CSS class name "aui-field" that will be positioned by Alloy UI CSS.

Table 2.4. Attributes

Attribute	Type	Description	Required
id	String	The identifier of the component	false
cssClass	String	The name of a CSS class that is to be rendered within the class attribute.	false
inlineLabel	String	The position of the label. Valid values are: left, right.	false
label	String	The text value for the rendered label.	false
rendered	Boolean	Boolean flag indicating whether or not this component is to be rendered during the RENDER_RESPONSE phase of the JSF lifecycle. The default value is “true”.	false
styleClass	String	The name of a CSS class that is to be rendered within the class attribute.	false
type	String	The type of field. Valid values are: checkbox, boolean, menu, select.	false

2.4. The aui:fieldset tag

The `aui:fieldset` tag renders a fieldset element with CSS class name "aui-fieldset" that will be positioned by Alloy UI CSS.

Table 2.5. Attributes

Attribute	Type	Description	Required
id	String	The identifier of the component	false
column	Boolean	Boolean flag indicating whether or not this is a column. The default value is “false”.	false
cssClass	String	The name of a CSS class that is to be rendered within the class attribute.	false
label	String	The text value for the rendered label.	false
rendered	Boolean	Boolean flag indicating whether or not this component is to be rendered during the RENDER_RESPONSE phase of the JSF lifecycle. The default value is “true”.	false
styleClass	String	The name of a CSS class that is to be rendered within the class attribute.	false

2.5. The aui:form tag

The `aui:form` tag renders a form element with CSS class name "aui-form" that will be positioned by Alloy UI CSS.

Table 2.6. Attributes

Attribute	Type	Description	Required
id	String	The identifier of the component	false
rendered	Boolean	Boolean flag indicating whether or not this component is to be rendered during the RENDER_RESPONSE phase of the JSF lifecycle. The default value is “true”.	false
styleClass	String	The name of a CSS class that is to be rendered within the class attribute.	false

2.6. The aui:layout tag

The `aui:layout` tag renders a div element with CSS class name "aui-layout" that will be positioned by Alloy UI CSS.

Table 2.7. Attributes

Attribute	Type	Description	Required
id	String	The identifier of the component	false
cssClass	String	The name of a CSS class that is to be rendered within the class attribute.	false
rendered	Boolean	Boolean flag indicating whether or not this component is to be rendered during the RENDER_RESPONSE phase of the JSF lifecycle. The default value is “true”.	false
styleClass	String	The name of a CSS class that is to be rendered within the class attribute.	false

2.7. The aui:list tag

The `aui:list` tag renders a ul element with CSS class name "aui-list" that will be positioned by Alloy UI CSS.

Table 2.8. Attributes

Attribute	Type	Description	Required
id	String	The identifier of the component	false
cssClass	String	The name of a CSS class that is to be rendered within the class attribute.	false
rendered	Boolean	Boolean flag indicating whether or not this component is to be rendered during the RENDER_RESPONSE phase of the JSF lifecycle. The default value is “true”.	false
styleClass	String	The name of a CSS class that is to be rendered within the class attribute.	false

2.8. The aui:list tag

The `aui:list` tag renders an li element with CSS class name "aui-listitem" that will be positioned by Alloy UI CSS.

Table 2.9. Attributes

Attribute	Type	Description	Required
id	String	The identifier of the component	false
cssClass	String	The name of a CSS class that is to be rendered within the class attribute.	false
rendered	Boolean	Boolean flag indicating whether or not this component is to be rendered during the RENDER_RESPONSE phase of the JSF lifecycle. The default value is “true”.	false
styleClass	String	The name of a CSS class that is to be rendered within the class attribute.	false

2.9. The aui:text-box-list tag

The `aui:text-box-list` tag renders a div element with CSS class name "aui-textboxlist" that will be positioned by Alloy UI CSS.

Table 2.10. Attributes

Attribute	Type	Description	Required
id	String	The identifier of the component	false
cssClass	String	The name of a CSS class that is to be rendered within the class attribute.	false
rendered	Boolean	Boolean flag indicating whether or not this component is to be rendered during the RENDER_RESPONSE phase of the JSF lifecycle. The default value is “true”.	false
styleClass	String	The name of a CSS class that is to be rendered within the class attribute.	false

2.10. The aui:text-box-list-item tag

The `aui:text-box-list-item` tag renders an li element with CSS class name "aui-textboxlistentry" that will be positioned by Alloy UI CSS.

Table 2.11. Attributes

Attribute	Type	Description	Required
id	String	The identifier of the component	false
cssClass	String	The name of a CSS class that is to be rendered within the class attribute.	false
rendered	Boolean	Boolean flag indicating whether or not this component is to be rendered during the RENDER_RESPONSE phase of the JSF lifecycle. The default value is “true”.	false
styleClass	String	The name of a CSS class that is to be rendered within the class attribute.	false

Chapter 3. Composite Component Tags

Liferay Faces Alloy provides the following Facelet Composite Component tags as part of its component suite.

Table 3.1. Facelet Composite Component Tags

Tag	Description
`aui-cc:button`	Renders a div with CSS class name "aui-button" and a child input element with CSS class name "aui-button-input".
`aui-cc:input`	Renders either an input (type=text), textarea, input (type="secret"), input (type="checkbox"), or input (type="radio") with appropriate Alloy UI CSS class names, according to the value of the type attribute.
`aui-cc:message`	Encapsulates a standard JSF h:message tag with JSR 286 (Portlet 2.0) CSS class names.
`aui-cc:messages`	Encapsulates a standard JSF h:messages tag with JSR 286 (Portlet 2.0) CSS class names.
`aui-cc:select`	Encapsulates a standard JSF h:selectOneMenu tag with CSS class name "aui-field-input aui-field-input-select aui-field-input-menu".

3.1. The aui-cc:button tag

The `aui-cc:button` renders a div with CSS class name "aui-button" and a child input element with CSS class name "aui-button-input".

Table 3.2. Attributes

Attribute	Type	Description	Required
id	String	The identifier of the component	false
cssClass	String	The name of a CSS class that is to be rendered within the class attribute.	false
immediate	Boolean	Boolean flag indicating whether or not actions and action listeners are to be executed during the APPLY_REQUEST_VALUES phase of the JSF lifecycle. The default value is “false”.	false
label	String	The text value for the rendered label of the button.	false
rendered	Boolean	Boolean flag indicating whether or not this component is to be rendered during the RENDER_RESPONSE phase of the JSF lifecycle. The default value is “true”.	false
styleClass	String	The name of a CSS class that is to be rendered within the class attribute.	false

3.2. The aui-cc:input tag

The `aui-cc:input` renders either an input (type=text), textarea, input (type="secret"), input (type="checkbox"), or input (type="radio") with appropriate Alloy UI CSS class names, according to the value of the type attribute.

onchange	String	This is an HTML pass-thru type of attribute.	false
readonly	Boolean	This is an HTML pass-thru type of attribute. The default value is “false”.	false
required	Boolean	Boolean flag indicating whether or not it is required for the user to supply a value for this component. The default value is “false”.	false
rendered	Boolean	Boolean flag indicating whether or not this component is to be rendered during the RENDER_RESPONSE phase of the JSF lifecycle. The default value is “true”.	false
rows	String	The number of rows for the rendered textarea.	false
size	String	The size/width of the rendered input text box.	false
style	String	This is an HTML pass-thru type of attribute.	false
styleClass	String	The name of a CSS class that is to be rendered within the class attribute.	false
type	String	The type of input field. Valid values are: text, textarea, password, checkbox, boolean.	false

Table 3.3. Attributes

3.3. The aui-cc:message tag

The `aui-cc:message` encapsulates a standard JSF h:message tag with JSR 286 (Portlet 2.0) CSS class names.

Table 3.4. Attributes

Attribute	Type	Description	Required
id	String	The identifier of the component	false
for	String	The id of the component for which this message is associated.	false
rendered	Boolean	Boolean flag indicating whether or not this component is to be rendered during the RENDER_RESPONSE phase of the JSF lifecycle. The default value is “true”.	false
styleClass	String	The name of a CSS class that is to be rendered within the class attribute.	false
style	String	This is an HTML pass-thru type of attribute.	false

3.4. The aui-cc:messages tag

The `aui-cc:messages` encapsulates a standard JSF h:messages tag with JSR 286 (Portlet 2.0) CSS class names.

Table 3.5. Attributes

Attribute	Type	Description	Required
id	String	The identifier of the component	false
globalOnly	Boolean	Boolean flag indicating whether or not this component is to render only the global messages, meaning, messages that are not associated with an individual component. The default value is “false”.	false
layout	String	The layout of the list of messages. Valid values are: list, table.	false
rendered	Boolean	Boolean flag indicating whether or not this component is to be rendered during the RENDER_RESPONSE phase of the JSF lifecycle. The default value is “true”.	false
styleClass	String	The name of a CSS class that is to be rendered within the class attribute.	false
style	String	This is an HTML pass-thru type of attribute.	false

3.5. The aui-cc:select tag

The `aui-cc:select` encapsulates a standard JSF h:selectOneMenu tag with CSS class name "aui-field-input aui-field-input-select aui-field-input-menu". NOTE that sometimes the JSF implementation performs more reliably if a simple h:selectOneMenu is used instead of this tag.

Table 3.6. Attributes

Attribute	Type	Description	Required
id	String	The identifier of the component	false
inlineMessage	String	Boolean flag indicating whether or not there should be an inline h:message tag included. The default value is “false”.	false
label	String	The text value for the rendered label.	false
rendered	Boolean	Boolean flag indicating whether or not this component is to be rendered during the RENDER_RESPONSE phase of the JSF lifecycle. The default value is “true”.	false
value	java.util.List	The list of items that are to appear in the drop down list.	false

Chapter 4. Overview

Liferay Faces Bridge is a JAR that JSF developers can add as a dependency to their portlet WAR projects in order to deploy JSF web applications as portlets within JSR 286 (Portlet 2.0) compliant portlet containers like Liferay Portal 5.2, 6.0, and 6.1.

The project home page can be found at
http://www.liferay.com/community/liferay-projects/liferay-faces/bridge

Chapter 5. Portlet Standard

5.1. Overview

Portlets are web applications that are designed to run inside a portlet container that implements either the Portlet 1.0 (JSR 168) or Portlet 2.0 (JSR 286) standard. Portlet containers provide a layer of abstraction over the Java EE Servlet API, and consequently require a servlet container like Apache Tomcat to function. The reference implementation for Portlet 1.0 and 2.0 is the Apache Pluto project: http://portals.apache.org/pluto.

Portals are standalone systems that use a portlet container as the runtime engine for executing portlets. When a portal is asked to deliver a portal page to the end-user's web browser, each portlet is asked to render itself as a fragment of HTML. It is the job of the portal to aggregate these HTML fragments into a complete HTML document.

5.2. Portlet Lifecycle

The Portlet 1.0 standard defines two lifecycle phases for the execution of a portlet that a compliant portlet container must support: The first is the javax.portlet.PortletRequest.RENDER_PHASE, in which the portlet container asks each portlet to render itself as a fragment of HTML. The second is the javax.portlet.PortletRequest.ACTION_PHASE, in which the portlet container invokes actions related to HTML form submission. When the portal receives an HTTP GET request for a portal page, the portlet container executes the portlet lifecycle and each of the portlets on the page undergoes the RENDER_PHASE. When the portal receives an HTTP POST request, the portlet container executes the portlet lifecycle and the portlet associated with the HTML form submission will first undergo the ACTION_PHASE before the RENDER_PHASE is invoked for all of the portlets on the page.

The Portlet 2.0 standard adds two more lifecycle phases that define the execution of a portlet. The first is the javax.portlet.PortletRequest.EVENT_PHASE, in which the portlet container broadcasts events that are the result of an HTML form submission. During this phase, the portlet container asks each portlet to process events that they are interested in. The typical use case for the EVENT_PHASE is to achieve Inter-Portlet Communication (IPC), whereby two or more portlets on a portal page share data in some way. The other new phase added by the Portlet 2.0 standard is the javax.portlet.PortletRequest.RESOURCE_PHASE, in which the portlet container asks a specific portlet to perform resource-related processing. One typical use case for the RESOURCE_PHASE is for an

individual portlet to process Ajax requests. Another typical use case for the RESOURCE_PHASE is for an individual portlet to generate non-HTML content (for download purposes) such as a PDF or spreadsheet document.

5.3. Portlet Modes

The Portlet 1.0 and 2.0 standards define three portlet modes that a compliant portlet container must support: javax.portlet.PortletMode.VIEW, javax.portlet.PortletMode.EDIT, and javax.portlet.PortletMode.HELP. Portal vendors and portlet developers may supply custom modes as well. VIEW mode refers to the rendered portlet markup that is encountered by the user under normal circumstances. Perhaps a clearer name would be "normal" mode or "typical" mode, because the word "view" is also used by developers to review to the "view" concern of the MVC design pattern. EDIT mode refers to the rendered portlet markup that is encountered by the user when selecting custom values for portlet preferences. Perhaps a clearer name would be "preferences" mode. Finally, HELP mode refers to the rendered portlet markup that is encountered by the user when seeking help regarding the usage and/or functionality of the portlet.

Figure 5.1. Portlet VIEW Mode

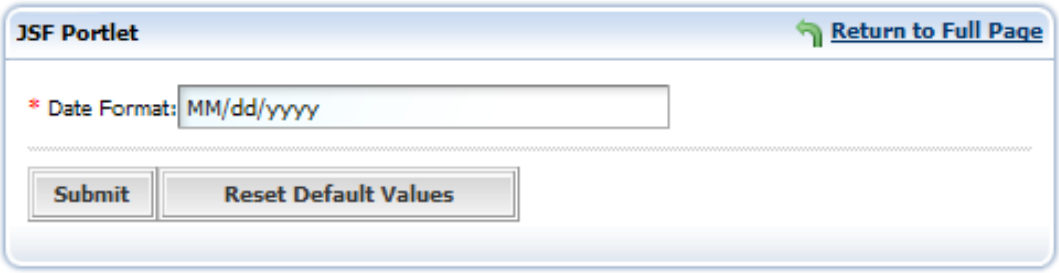

Figure 5.2. Portlet EDIT Mode

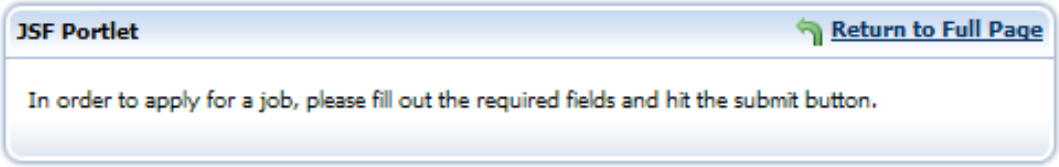

Figure 5.3. Portlet HELP Mode

5.4. Portlet Window States

Portals typically manifest the rendered markup of a portlet in a rectangular section of the browser known as a portlet window. The Portlet 1.0 and 2.0 standards define three window states that a compliant portlet container must support: javax.portlet.WindowState.NORMAL, javax.portlet.WindowState.MAXIMIZED, and javax.portlet.WindowState.MINIMIZED. The NORMAL window state refers to the way in which the portlet container displays the rendered markup of a portlet when it can appear on the same portal page as other portlets. The MAXIMIZED window state refers to the way in which the portlet container displays the rendered markup of a portlet when it is the only portlet on a page, or when the portlet is to be rendered more prominently than other portlets on a page. Finally, the MINIMIZED window state refers to the way in which the portlet container displays a portlet when the markup is not to be rendered.

5.5. Portlet Preferences

Developers often have the requirement to provide the end-user with the ability to personalize the portlet behavior in some way. To meet this requirement, the Portlet 1.0 and 2.0 standards provide the ability to define preferences for each portlet. Preference names and default values can be defined in the WEB-INF/portlet.xml configuration file. Portal end-users start out interacting with the portlet user interface in portlet VIEW mode but can switch to portlet EDIT mode in order to select custom preference values.

Example 5.1. Specifying preference names and associated default values in the WEB-INF/portlet.xml configuration file

```
<portlet-app>
 <portlet>
  ...
  <portlet-preferences>
   <preference>
    <name>datePattern</name>
    <value>MM/dd/yyyy</value>
   </preference>
   <preference>
    <name>unitedStatesPhoneFormat</name>
    <value>###-###-####</value>
   </preference>
  </portlet-preferences>
  ...
 </portlet>
</portlet-app>
```

5.6. Inter-Portlet Communication

Inter-Portlet Communication (IPC) is a technique whereby two or more portlets on a portal page share data in some way. In a typical IPC use case, user interactions with one portlet affect the rendered markup of another portlet. The Portlet 2.0 standard provides two techniques to achieve IPC: Public Render Parameters and Server-Side Events.

The Public Render Parameters technique provides a way for portlets to share data by setting public/shared parameter names in a URL controlled by the portal. While the benefit of this approach is that it is relatively easy to implement, the drawback is that only small amounts of data can be shared. Typically the kind of data that is shared is simply the value of a database primary key.

The Server-Side Events technique provides a way for portlets to share data using an event-listener design. When using this form of IPC, the portlet container acts as broker and distributes events and payload (data) to portlets. One requirement of this approach is that the payload must implement the java.io.Serializable interface since it might be sent to a portlet in another WAR running in a different classloader.

It could be argued that the Portlet 2.0 approaches for IPC have a common drawback in that they can lead to a potentially disruptive end-user experience. This is because they cause either an HTTP GET or HTTP POST which results in a full page refresh. Technologies such as ICEfaces Ajax

Push can be used to solve this problem. Refer to the Ajax Push IPC section of this document for more details.

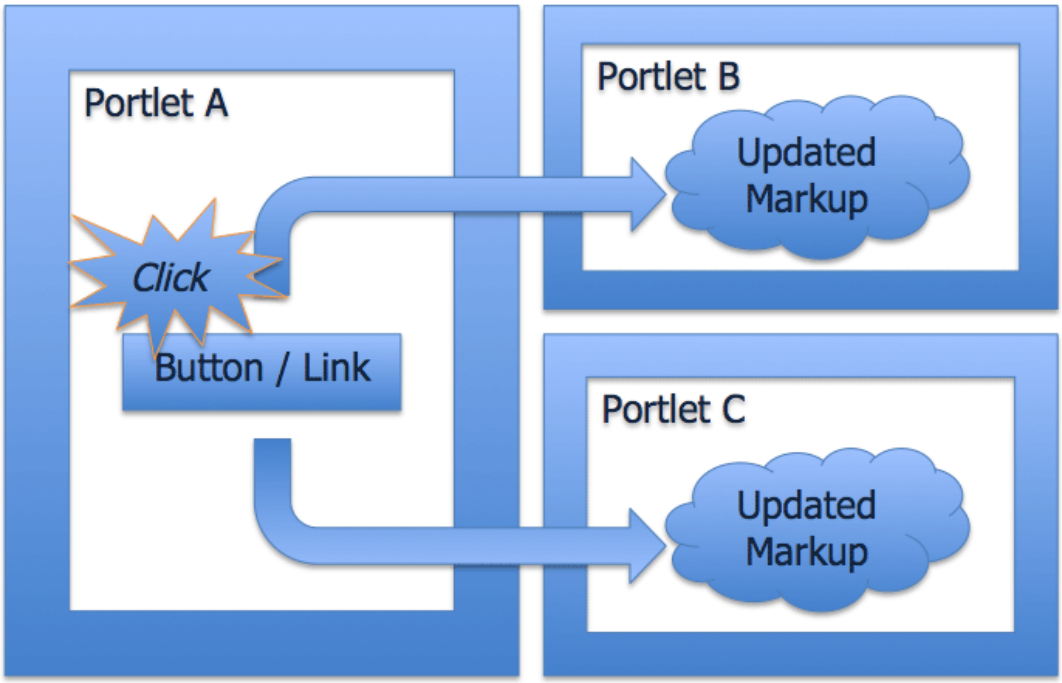

Figure 5.4. Illustration of Standard Portlet 2.0 IPC

Chapter 6. Portlet Bridge Standard

6.1. Overview

The Portlet 1.0 and JSF 1.0 specifications were formulated during roughly the same timeframe. Consequently, the JSF Expert Group (EG) was able to design a framework with portlet compatibility in mind. This is evidenced by methods like ExternalContext.getRequest() which returns a value of type `Object`, rather than a value of type javax.servlet.http.HttpServletRequest. When running inside a portlet container, the same method would return a value of type javax.portlet.PortletRequest. Although the JSF API provides a degree of portlet compatibility, it is necessary to introduce a bridge between the JSF lifecycle and the Portlet lifecycle in order to run JSF applications as portlets.

6.2. Portlet Bridge 1.0

Starting in 2004, several different JSF portlet bridge implementations were developed in order to provide JSF developers with the ability to deploy their JSF webapps as portlets. In 2006 the JCP formed the Portlet Bridge 1.0 (JSR 301) EG in order to define a standard bridge API as well as detailed requirements for bridge implementations. JSR 301 was released in 2010 and targeted Portlet 1.0 and JSF 1.2.

6.3. Portlet Bridge 2.0

When the Portlet 2.0 (JSR 286) standard was released in 2008 it became necessary for the JCP to form the Portlet Bridge 2.0 (JSR 329) EG. JSR 329 was also released in 2010 and targeted Portlet 2.0 and JSF 1.2.

6.4. Portlet Bridge 3.0

After the JSR 314 EG released JSF 2.0 in 2009 and JSF 2.1 in 2010, it became evident that a Portlet Bridge 3.0 standard would be beneficial. At the time of this writing, the JCP has not formed such an EG. In the meantime, the Liferay has developed **Liferay Faces Bridge**, which targets Portlet 2.0 and JSF 2.0/2.1.

Liferay Faces Bridge is an implementation of the JSR 329 Portlet Bridge Standard. It also contains innovative features that make it possible to leverage the power of JSF 2 inside a portlet application.

6.5. Portlet Lifecycle and JSF Lifecycle

JSF portlet bridges are responsible for providing a “bridge” between the portlet lifecycle and the JSF lifecycle. For example, when a portal page that contains a JSF portlet is requested via HTTP GET, then the RENDER_PHASE of the portlet lifecycle should in turn execute the RESTORE_VIEW and RENDER_RESPONSE phases of the JSF lifecycle. Similarly, when the user submits a form contained within a JSF portlet via HTTP POST, then the ACTION_PHASE of the portlet lifecycle should execute the complete JSF lifecycle of RESTORE_VIEW, APPLY_REQUEST_VALUES, PROCESS_VALIDATIONS, UPDATE_MODEL_VALUES, INVOKE_APPLICATION, and RENDER_RESPONSE. Since the portal is in full control of managing URLs, JSF portlet bridges are also responsible for asking the portal to generate URLs that are compatible with actions that invoke JSF navigation rules. If a different JSF view is to be rendered as a result of a JSF navigation-rule, then the JSF portlet bridge simply displays the new JSF view in the same portlet window.

Chapter 7. Liferay Faces Bridge Configuration

7.1. Overview

The JSR 329 standard defines several configuration options prefixed with the javax.portlet.faces namespace. Liferay Faces Bridge defines additional implementation specific options prefixed with the com.liferay.faces.bridge namespace.

7.2. Bridge Request Scope

One of the key requirements in creating a JSF portlet bridge is managing JSF request-scoped data within the Portlet lifecycle. This is normally referred to as the "Bridge Request Scope" by JSR 329. The lifespan of the BridgeRequestScope works like this:

1. ActionRequest/EventRequest: BridgeRequestScope begins
2. RenderRequest: BridgeRequestScope is preserved
3. Subsequent RenderRequest: BridgeRequestScope is reused
4. Subsequent ActionRequest/EventRequest: BridgeRequestScope ends, and a new BridgeRequestScope begins
5. If the session expires or is invalidated, then similar to the PortletSession scope, all BridgeRequestScope instances associated with the session are made avialable for garbage collection by the JVM

The main use-case for having the BridgeRequestScope preserved in #2 (above) is for "re-render" of portlets. One example would be when two or more JSF portlets are placed on a portal page (Portlets X and Y), and those portlets are not using f:ajax for form submission. In such a case, if the user were to submit a form (via full ActionRequest postback) in Portlet X, and then submit a form in Portlet Y, then Portlet X should be re-rendered with its previously submitted form data.

With the advent of JSF 2 and Ajax, there are four drawbacks for supporting this use-case as the default behavior:

- Request-scoped data basically semi-session-scoped in nature, because the BridgeRequestScope is preserved (even though the user might NEVER click the Submit button again).

- BridgeRequestScope can't be stored in the PortletSession because the data is Request-scoped in nature, and the data stored in the scope isn't guaranteed to be Serializable for replication. Therefore it doesn't really work well in a clustered deployment.
- The developer might have to specify the javax.portlet.faces.MAX_MANAGED_REQUEST_SCOPES init-param in the WEB-INF/web.xml descriptor in order to tune the memory settings on the server.
- The developer is forced to add a <listener> for the BridgeSessionListener to the WEB-INF/web.xml descriptor.

Therefore, since Liferay Faces Bridge is designed for JSF 2 and Ajax in mind, the bridge makes the following assumptions:

- That developers are not primarily concerned about the "re-render" of portlets use-case mentioned above.
- That developers don't want any of the drawbacks mentioned above.
- That developers are making heavy use of the f:ajax tag (or implicitly doing so with ICEfaces) and submitting forms via Ajax with their modern-day portlets.
- That developers want to be as zero-config as possible, and don't want to be forced to add anything to the WEB-INF/web.xml descriptor.

Consequently, the default behavior of Liferay Faces Bridge is to cause the BridgeRequestScope to end at the end of the RenderRequest. If the standard behavior is desired, then the following options can be placed in the WEB-INF/web.xml descriptor.

Example 7.1. Specifying standard behavior for BridgeRequestScope lifespan via the WEB-INF/web.xml descriptor

```
<!-- The default value of the following -->
<!-- context-param is false, meaning that -->
<!-- Liferay Faces Bridge will cause -->
<!-- the BridgeRequestScope to end after -->
<!-- the RENDER_PHASE of the portlet -->
<!-- lifecycle. Setting the value to true -->
<!-- will cause Liferay Faces Bridge to -->
<!-- cause the BridgeRequestScope to last -->
<!-- until the next ACTION_PHASE or -->
<!-- EVENT_PHASE of the portlet lifecycle. -->
<context-param>
 <param-name>com.liferay.faces.bridge.bridgeRequestScopePreserved</
param-name>
 <param-value>true</param-value>
</context-param>
<!-- The default value of the following -->
<!-- context-param is 100. It defines -->
<!-- the maximum number of BridgeRequestScope -->
<!-- instances to keep in memory on the server -->
<!-- if the bridgeRequestScopePreserved -->
<!-- option is true. -->
<context-param>
 <param-name>javax.portlet.faces.MAX_MANAGED_REQUEST_SCOPES</param-
name>
 <param-value>2000</param-value>
</context-param>
<!-- The following listener is required to cleanup -->
<!-- BridgeRequestScope instances upon session timeout -->
<listener>
 <listener-
class>com.liferay.faces.bridge.servlet.BridgeSessionListener</
listener-class>
</listener>
```

Alternatively, the com.liferay.faces.bridge.bridgeRequestScopePreserved value can be specified on a portlet-by-portlet basis in the WEB-INF/portlet.xml descriptor.

7.3. PreDestroy & BridgePreDestroy Annotations

When JSF developers want to perform cleanup on managed-beans before they are destroyed, they typically annotate a method inside the bean with the @PreDestroy annotation. However, section 6.8.1 of the JSR 329 standard discusses the need for the @BridgePreDestroy and @BridgeRequestScopeAttributeAdded annotations in the bridge API.

In-Depth Discussion Avialable Online

For a in-depth discussion of this issue, please refer to http://issues.liferay.com/browse/FACES-146

In order to explain this requirement, it is necessary to make a distinction between *local* portals and *remote* portals. Local portals invoke portlets that are deployed within the same (local) servlet container. Remote portals invoke portlets that are deployed elsewhere via WSRP (Web Services for Remote Portlets). The @BridgePreDestroy and @BridgeRequestScopeAttributeAdded annotations were introduced into the JSR 329 standard primarily to support WSRP in remote portals. That being the case, the standard indicates that developers should always use @BridgePreDestroy instead of @PreDestroy. Liferay Faces Bridge however takes a different approach: rather than assuming the remote portal use-case, Liferay Faces Bridge assumes the local portal use-case. When developing with a local portal like Liferay, Liferay Faces Bridge ensures that the standard @PreDestroy annotation works as expected. This means there is no reason to use the @BridgeRequestScope annotation with a local portal when using Liferay Faces Bridge. Developers must manually configure Liferay Faces Bridge via the WEB-INF/web.xml descriptor in order to leverage the @BridgePreDestroy and @BridgeRequestScopeAttributeAdded annotations for WSRP.

Example 7.2. Specifying support for @BridgePreDestroy and @BridgeRequestScopeAttributeAdded in the WEB-INF/web.xml descriptor

```
<!-- The default value of the following -->
<!-- context-param is false, meaning that -->
<!-- Liferay Faces Bridge will invoke -->
<!-- methods annotated with @PreDestroy -->
<!-- over those annoated with -->
<!-- @BridgePreDestroy. -->
<!-- Setting the value of the following -->
<!-- context-param instructs Liferay Faces -->
<!-- Bridge to prefer the @BridgePreDestroy -->
<!-- annotation over the standard @PreDestroy -->
<!-- annotation in order to support a WSRP -->
<!-- remote portal environment. -->
<context-param>
 <param-name>com.liferay.faces.bridge.preferPreDestroy</param-name>
 <param-value>false</param-value>
</context-param>
<!-- The following listener is required to support -->
<!-- the @BridgeRequestScopeAttributeAdded -->
<!-- annotation in a WSRP remote portal environment. -->
<listener>
 <listener-
class>com.liferay.faces.bridge.servlet.BridgeRequestAttributeListener</
listener-class>
</listener>
```

Alternatively, the com.liferay.faces.bridge.preferPreDestroy value can be specified on a portlet-by-portlet basis in the WEB-INF/portlet.xml descriptor.

7.4. Portlet Container Abilities

Liferay Faces Bridge can be run in a variety of portlet containers (Liferay, Pluto, etc.) and is aware of some of the abilities (or limitations) of these containers. Regardless, Liferay Faces Bridge enables the developer to configure the abilities of the portlet container in the WEB-INF/web.xml descriptor.

Example 7.3. Setting portlet container abilities via the WEB-INF/web.xml descriptor

```
<!-- The default value of the following -->
<!-- context-param depends on which -->
<!-- portlet container the bridge -->
<!-- is running in. The value determines -->
<!-- whether or not the bridge resource -->
<!-- handler will attempt to set the -->
<!-- status code of downloaded resources -->
<!-- to values like -->
<!-- HttpServletResponse.SC_NOT_MODIFIED -->
<context-param>
 <param-
name>com.liferay.faces.bridge.containerAbleToSetHttpStatusCode</
param-name>
 <param-value>true</param-value>
</context-param>
```

7.5. Portlet Namespace Optimization

The JSR 329 standard requires the bridge implementation to prepend the portlet namespace to the value of the "id" attribute of every component that is rendered by a JSF view. This guarantees the uniqueness of the "id" attribute when there are multiple JSF portlets on a portal page that contain similar component hierarchies and naming. Also, the JSR 329 standard indicates that the bridge implementation of the ExternalContext.encodeNamesapce(String) method is to prepend the value of javax.portlet.PortletResponse.getNamespace() to the specified String. The problem is that since the value returned by getNamespace() can be a lengthy string, the size of the rendered HTML portal page can become unnecessarily large. This can be especially non-performant when using the f:ajax tag in a Facelet view in order to perform partial-updates the browser's DOM.

Liferay Faces Bridge has a built-in optimization that minimizes the value returned by the the ExternalContext.encodeNamesapce(String) method, while still guaranteeing uniqueness. Developers must manually configure Liferay Faces Bridge via the WEB-INF/web.xml descriptor in order to disable the namespace optimization and leverage the default behavior specified by JSR 329.

Example 7.4. Disabling the namespace optimization via the WEB-INF/web.xml descriptor

```
<!-- The default value of the following -->
<!-- context-param is true, meaning that -->
<!-- Liferay Faces Bridge will optimize -->
<!-- the portlet namespace. Setting the value -->
<!-- of the following context-param to false -->
<!-- disables the optimization. -->
<context-param>
 <param-name>com.liferay.faces.bridge.optimizePortletNamespace</
param-name>
 <param-value>false</param-value>
</context-param>
```

7.6. Resolving XML Entities

Liferay Faces Bridge provides the ability to set a flag indicating whether or not XML entities are required to be resolved when parsing faces-config.xml files in the classpath. The default value of this option is false.

Example 7.5. Specifying the resolving of XML entities via the WEB-INF/web.xml descriptor

```
<!-- The default value of the following -->
<!-- context-param is false. -->
<context-param>
 <param-name>com.liferay.faces.bridge.resolveXMLEntities</param-
name>
 <param-value>true</param-value>
</context-param>
```

7.7. Resource Buffer Size

Liferay Faces Bridge provides the ability to set the size of the buffer used to load resources into memory as the file contents are being copied to the response. The default value of this option is 1024 (1KB).

Example 7.6. Specifying the resource buffer size via the WEB-INF/web.xml descriptor

```
<!-- The default value of the following -->
<!-- context-param is 1024. -->
<context-param>
 <param-name>com.liferay.faces.bridge.resourceBufferSize</param-
name>
 <param-value>4096</param-value>
</context-param>
```

Alternatively, the com.liferay.faces.bridge.resourceBufferSize value can be specified on a portlet-by-portlet basis in the WEB-INF/portlet.xml descriptor.

Chapter 8. JSF Portlet Development

8.1. Overview

The main goal of JSF portlet bridges is to make the JSF portlet development experience as close as possible to JSF webapp development. Consequently, many JSF webapps can be easily migrated to a portlet container using such a bridge.

8.2. Liferay Faces Bridge

In order to use JSF2 in a portlet, developers must specify `javax.portlet.faces.GenericFacesPortlet` in the `<portlet-class>` element of the `WEB-INF/portlet.xml` descriptor.

Example 8.1. Specifying Liferay Faces Bridge in the WEB-INF/portlet.xml configuration file, as well as default Facelet views that are to be rendered for VIEW mode, EDIT mode, and HELP mode

```
<portlet-app>
 <portlet>
  <portlet-name>my_portlet</portlet-name>
  <display-name>My Portlet</display-name>
  <portlet-class>
   javax.portlet.faces.GenericFacesPortlet
  </portlet-class>
  <init-param>
   <name>javax.portlet.faces.defaultViewId.view</name>
   <value>/xhtml/applicantForm.xhtml</value>
  </init-param>
  <init-param>
   <name>javax.portlet.faces.defaultViewId.edit</name>
   <value>/xhtml/edit.xhtml</value>
  </init-param>
  <init-param>
   <name>javax.portlet.faces.defaultViewId.help</name>
   <value>/xhtml/help.xhtml</value>
  </init-param>
  <supports>
   <mime-type>text/html</mime-type>
   <portlet-mode>view</portlet-mode>
   <portlet-mode>edit</portlet-mode>
   <portlet-mode>help</portlet-mode>
  </supports>
  ...
 </portlet>
</portlet-app>
```

8.3. JSF and PortletPreferences

JSF portlet developers often have the requirement to provide the end-user with the ability to personalize the portlet in some way. To meet this requirement, the Portlet 2.0 specification provides the ability to define portlet preferences for each portlet. Preference names and default values can be defined in the `WEB-INF/portlet.xml` descriptor. Portal end-users start out interacting with the portlet user interface in portlet `VIEW` mode but switch to portlet `EDIT` mode in order to select custom preference values.

Example 8.2. Portlet Preferences in WEB-INF/portlet.xml

```
<portlet-preferences>
 <preference>
  <name>datePattern</name>
  <value>MM/dd/yyyy</value>
 </preference>
</portlet-preferences>
```

Additionally, Portlet 2.0 provides the ability to specify support for `EDIT` mode in the `WEB-INF/portlet.xml` descriptor.

Example 8.3. Enabling Support for Portlet EDIT Mode in WEB-INF/portlet.xml

```
<supports>
 <mime-type>text/html</mime-type>
 <portlet-mode>view</portlet-mode>
 <portlet-mode>edit</portlet-mode>
</supports>
```

However, just because support portlet `EDIT` mode has been specified, it doesn't mean that the portlet container knows which JSF view should be rendered when the user enters portlet portlet `EDIT` mode. JSF portlet developers must specify the Facelet view that is to be displayed for each supported portlet mode.

Example 8.4. Specifying a Facelet View for EDIT mode with Liferay Faces Bridge

```
<init-param>
 <name>javax.portlet.faces.defaultViewId.edit</name>
 <value>/edit.xhtml</value>
</init-param>
```

Facelet views that are designed to be used in portlet `EDIT` mode are typically forms that contain JSF component tags that enable the portlet end-user to select custom preference values that override the default values specified in the `WEB-INF/portlet.xml` descriptor. JSR 329 bridge implementations are required to provide an EL resolver that introduces the `mutablePortletPreferencesValues` variable into the EL, which is a mutable `java.util.Map` that provides read/write access to each portlet preference. By utilizing the JSR 329 `mutablePortletPreferencesValues` variable within an EL `ValueExpression`, portlet developers can declaratively bind the Facelet

view to the portlet preference model data. In order to save the preferences, a backing bean must call the PortletPreferences.store() method.

Example 8.5. EDIT Mode Facelet XHTML

```
<!--
This is a file named edit.xhtml that can be used for portlet EDIT
mode. It utilizes the JSR 329 mutablePortletPreferencesValues EL
variable for gaining read/write access to
javax.portlet.PortletPreferences.
-->
<?xml version="1.0" encoding="UTF-8"?>
<f:view xmlns:f="http://java.sun.com/jsf/core"
 xmlns:h="http://java.sun.com/jsf/html"
 xmlns:ui="http://java.sun.com/jsf/facelets">
 <h:head />
 <h:body>
 <h:form>
  <h:messages globalOnly="true" />
  <table>
   <tr>
    <td><h:outputLabel for="datePattern" value="Date Format" /></td>
    <td><h:inputText id="datePattern"
 value="#{mutablePortletPreferencesValues['datePattern'].value}"
 /></td>
    <td><h:message for="datePattern" /></td>
   </tr>
  </table>
  <hr />
  <h:commandButton
 actionListener="#{portletPreferencesBackingBean.submit}"
 value="Submit" />
 </h:form>
 </h:body>
</f:view>
```

Example 8.6. EDIT Mode Backing Bean - Part I

```
/**
 * This is a JSF backing managed-bean that has an action-listener
 * for saving portlet preferences.
 */
@ManagedBean(name = "portletPreferencesBackingBean")
@RequestScoped
public class PortletPreferencesBackingBean {

 public void submit() {

  // The JSR 329 specification defines an EL variable named
  // mutablePortletPreferencesValues that is being used in
  // the portletPreferences.xhtml Facelet composition. This
  // object is of type Map<String, Preference> and is
  // designed to be a model managed-bean (in a sense) that
  // contain preference values. However the only way to
  // access this from a Java class is to evaluate an EL
  // expression (effectively self-injecting) the map into
  // this backing bean.
  FacesContext facesContext = FacesContext.getCurrentInstance();
  ExternalContext externalContext =
   facesContext.getExternalContext();
  String elExpression = "mutablePortletPreferencesValues";
  ELResolver elResolver =
   facesContext.getApplication().getELResolver();
  @SuppressWarnings("unchecked")
  Map<String, Preference> mutablePreferenceMap =
   (Map<String, Preference>) elResolver.getValue(
    facesContext.getELContext(), null, elExpression);

  // Get a list of portlet preference names.
  PortletRequest portletRequest =
   (PortletRequest) externalContext.getRequest();
  PortletPreferences portletPreferences =
   portletRequest.getPreferences();
  Enumeration<String> preferenceNames =
   portletPreferences.getNames();
```

Example 8.7. EDIT Mode Backing Bean - Part II

```
  try {

   // For each portlet preference name:
   while (preferenceNames.hasMoreElements()) {

    // Get the value specified by the user.
    String preferenceName = preferenceNames.nextElement();
    String preferenceValue =
     mutablePreferenceMap.get(preferenceName).getValue();

    // Prepare to save the value.
    if (!portletPreferences.isReadOnly(preferenceName)) {
     portletPreferences.setValue(
      preferenceName, preferenceValue);
    }
   }

   // Save the preference values.
   portletPreferences.store();

   // Switch the portlet mode back to VIEW.
   ActionResponse actionResponse =
    (ActionResponse) externalContext.getResponse();
   actionResponse.setPortletMode(PortletMode.VIEW);

   // Report a successful message back to the user as feedback.
   FacesMessageUtil.addGlobalSuccessInfoMessage(facesContext);
  }
  catch (Exception e) {
   FacesMessageUtil.addGlobalUnexpectedErrorMessage(facesContext);
  }
 }
}
```

8.4. JSF ExternalContext and the Portlet API

Just as JSF web application developers rely on ExternalContext in order to get access to the Servlet API, JSF portlet developers also rely on ExternalContext in order to get access to the Portlet API.

8.4.1. Getting the PortletRequest and PortletResponse Objects

The two most common tasks that JSF portlet developers need to perform is to obtain an instance of the javax.portlet.PortletRequest or javax.portlet.PortletResponse objects.

Example 8.8. Getting the PortletRequest and PortletResponse objects from within a JSF backing managed-bean action method

```
public class BackingBean {

 public void submit() {
  FacesContext facesContext =
   FacesContext.getCurrentInstance();

  ExternalContext externalContext =
   facesContext.getExternalContext();

  PortletRequest portletRequest =
   (PortletRequest) externalContext.getRequest();

  PortletResponse portletResponse =
   (PortletResponse) externalContext.getResponse();
 }
}
```

Liferay Faces Portal Project

For Liferay portlet developers, Liferay Faces Portal project features the LiferayFacesContext singleton which contains convenience methods like liferayFacesContext.getPortletRequest() and liferayFacesContext.getPortletResponse().

8.5. JSF and Inter-Portlet Communication

Liferay Faces Bridge supports Portlet 2.0 IPC using the JSR 329 approach for supporting Portlet 2.0 Events and also Portlet 2.0 Public Render Parameters. It could be argued that the Portlet 2.0 approaches for IPC have a common drawback in that they can lead to a potentially disruptive end-user experience. This is because they cause either an HTTP GET or HTTP POST which results in a full page refresh. Technologies such as ICEfaces Ajax Push can be used to solve this problem. Refer to the Ajax Push IPC section of this document for more details.

Demo Portlets at liferay.com

Visit http://www.liferay.com/community/liferay-projects/liferay-faces/

demos for demo portlets that demonstrate how to use each of these various approaches to IPC.

8.5.1. Portlet 2.0 Public Render Parameters

As discussed in Chapter 1, the Public Render Parameters technique provides a way for portlets to share data by setting public/shared parameter names in a URL controlled by the portal. While the benefit of this approach is that it is relatively easy to implement, the drawback is that only small amounts of data can be shared. Typically the kind of data that is shared is simply the value of a database primary key. As required by the Portlet 2.0 standard, Public Render Parameters must be declared in the WEB-INF/portlet.xml descriptor.

Example 8.9. Specifying Supported Public Render Parameters in the WEB-INF/portlet.xml descriptor

```
<portlet>
 <portlet-name>customersPortlet</portlet-name>
 ...
 <supported-public-render-parameter>selectedCustomerId</supported-
public-render-parameter>
</portlet>
<portlet>
 <portlet-name>bookingsPortlet</portlet-name>
 ...
 <supported-public-render-parameter>selectedCustomerId</supported-
public-render-parameter>
</portlet>
<public-render-parameter>
 <identifier>selectedCustomerId</identifier>
 <qname
 xmlns:x="http://liferay.com/pub-render-
params">x:selectedCustomerId</qname>
</public-render-parameter>
```

The JSR 329 standard defines a mechanism by which developers can use Portlet 2.0 Public Render Parameters for IPC in a way that is more natural to JSF development. Section 5.3.2 requires the bridge to inject the public render parameters into the Model concern of the MVC design pattern (as in JSF model managed-beans) after RESTORE_VIEW phase completes. This is accomplished by evaluating the EL expressions found in the <model-el>...</model-el> section of the WEB-INF/faces-config.xml descriptor.

Example 8.10. Specifying Public Render Parameters in the WEB-INF/faces-config.xml descriptor

```
<faces-config>
 <application>
  <application-extension>
   <bridge:public-parameter-mappings>
    <bridge:public-parameter-mapping>
     <parameter>selectedCustomerId</parameter>

 <model-
el>#{customersPortlet:customersModelBean.selectedCustomerId}</model-
el>

 <model-el>#{bookingsPortlet:bookingsModelBean.selectedCustomerId}</
model-el>
    </bridge:public-parameter-mapping>
   </bridge:public-parameter-mappings>
  </application-extension>
 </application>
</faces-config>
```

Section 5.3.2 of the JSR 329 standard also requires that if a bridgePublicRenderParameterHandler has been registered in the WEB-INF/portlet.xml descriptor, then the handler must be invoked so that it can perform any processing that might be necessary.

Example 8.11. Specifying a bridgePublicRenderParameterHandler in the WEB-INF/portlet.xml descriptor

```
<!-- Optional bridgePublicRenderParameterHandler -->
<init-param>
 <name>javax.portlet.faces.bridgePublicRenderParameterHandler</name>
 <value>com.liferay.faces.example.handler.CustomerSelectedHandler</
value>
</init-param>
```

Example 8.12. bridgePublicRenderParameterHandler Java Code

```
package com.liferay.faces.example.handler;

import javax.faces.context.FacesContext;

import com.liferay.faces.bridge.BridgePublicRenderParameterHandler;

public class CustomerSelectedHandler implements
 BridgePublicRenderParameterHandler {

 public void processUpdates(FacesContext facesContext) {
  // Here is where you would perform any necessary processing of
 public render parameters
 }

}
```

8.5.2. Portlet 2.0 Events

As discussed in Chapter 1, the Server-Side Events technique provides a way for portlets to share data using an event-listener design. When using this form of IPC, the portlet container acts as broker and distributes events and payload (data) to portlets. One requirement of this approach is that the payload must implement the java.io.Serializable interface since it might be sent to a portlet in another WAR running in a different classloader.
As required by the Portlet 2.0 standard, Events must be declared in the WEB-INF/portlet.xml descriptor.

Example 8.13. Specifying Supported Events in the WEB-INF/portlet.xml descriptor

```
<portlet>
 <portlet-name>customersPortlet</portlet-name>
 ...
 <supported-processing-event>
  <qname
 xmlns:x="http://liferay.com/events">x:ipc.customerEdited</qname>
 </supported-processing-event>
 <supported-publishing-event>
  <qname
 xmlns:x="http://liferay.com/events">x:ipc.customerSelected</qname>
 </supported-publishing-event>
</portlet>
<portlet>
 <portlet-name>bookingsPortlet</portlet-name>
 ...
 <supported-processing-event>
  <qname
 xmlns:x="http://liferay.com/events">x:ipc.customerSelected</qname>
 </supported-processing-event>
 <supported-publishing-event>
  <qname
 xmlns:x="http://liferay.com/events">x:ipc.customerEdited</qname>
 </supported-publishing-event>
</portlet>
<event-definition>
 <qname
 xmlns:x="http://liferay.com/events">x:ipc.customerEdited</qname>
 <value-type>com.liferay.faces.example.dto.Customer</value-type>
</event-definition>
<event-definition>
 <qname
 xmlns:x="http://liferay.com/events">x:ipc.customerSelected</qname>
 <value-type>com.liferay.faces.example.dto.Customer</value-type>
</event-definition>
```

Section 5.2.5 of the JSR 329 standard requires that if a BridgeEventHandler has been registered in the WEB-INF/portlet.xml descriptor, then the handler must be invoked so that it can perform any processing that might be necessary.

Example 8.14. Specifying a BridgeEventHandler in the WEB-INF/portlet.xml descriptor

```
<!-- Optional bridgeEventHandler -->
<init-param>
 <name>javax.portlet.faces.bridgeEventHandler</name>
 <value>com.liferay.faces.example.event.CustomerEditedEventHandler</
value>
</init-param>
```

Example 8.15. BridgeEventHandler Java Code

```
package com.liferay.faces.example.event;

import javax.el.ELContext;
import javax.el.ValueExpression;
import javax.faces.context.FacesContext;
import javax.portlet.Event;

import com.liferay.faces.bridge.BridgeEventHandler;
import com.liferay.faces.bridge.event.EventNavigationResult;

public class CustomerEditedEventHandler implements
 BridgeEventHandler {

 public EventNavigationResult handleEvent(FacesContext facesContext,
 Event event) {
  EventNavigationResult eventNavigationResult = null;
  String eventQName = event.getQName().toString();

  if
 (eventQName.equals("{http://liferay.com/
events}ipc.customerEdited")) {
   Customer customer = (Customer) event.getValue();
   getCustomerService(facesContext).save(customer);
   System.err.println("Received event ipc.customerEdited");
   System.err.println("customerId=" + customer.getCustomerId());
  }

  return eventNavigationResult;
 }
}
```

8.5.3. Portlet 2.0 Shared Session Scope

Perhaps the most natural approach for a JSF developer to try for IPC is to specify session scope on a JSF managed-bean. Surprisingly, this approach doesn't work. To understand the reason why, it is necessary to discuss the fact that the Portlet 1.0 and 2.0 standards make a distinction between two kinds of session scopes: javax.portlet.PortletSession.APPLICATION_SCOPE

and javax.portlet.PortletSession.PORTLET_SCOPE. The former can be used for sharing data between portlets packaged in the same WAR, but the latter cannot. The reason why JSF session scope can't be used to share data between portlets is because all JSF portlet bridges use PortletSession.PORTLET_SCOPE.

In order to share data with PortletSession.APPLICATION_SCOPE, the JSF portlet developer can place a JSF model managed-bean in request scope and use the getter/setter as a layer of abstraction.

Example 8.16. Shared Scope Model Managed-Bean - Part I

```
/**
 * This class is a request-scoped JSF managed-bean that has
 * a getter and setter that serves as a layer of abstraction
 * over PortletSession.APPLICATION_SCOPE
 */
@RequestScoped(name="sharedScopeModelBean")
public class SharedScopeModelBean {

 public static final String
  SHARED_STRING_KEY = “sharedStringKey”;

 public String getSharedString() {
  return PortletSessionUtil.getSharedSessionAttribute(
   SHARED_STRING_KEY);
 }

 public void setSharedString(String value) {
  PortletSessionUtil.setSharedSessionAttribute(
   SHARED_STRING_KEY, value);
 }
}

public class PortletSessionUtil {

 public static Object getSharedSessionAttribute(
  String key) {

  FacesContext facesContext =
   FacesContext.getCurrentInstance();

  ExternalContext externalContext =
   facesContext.getExternalContext();

  PortletSession portletSession =
   (PortletSession) externalContext().getSession(false);

  return portletSession.getAttribute(
   key, PortletSession.APPLICATION_SCOPE);
 }
```

Example 8.17. Shared Scope Model Managed Bean - Part II

```
 public static void setSharedSessionAttribute(
  String key, Object value) {

  FacesContext facesContext =
   FacesContext.getCurrentInstance();

  ExternalContext externalContext =
   facesContext.getExternalContext();

  PortletSession portletSession =
   (PortletSession)externalContext().getSession(false);

  portletSession.setAttribute(
   key, value, PortletSession.APPLICATION_SCOPE);
 }
}
```

Chapter 9. JSF Component Tags

9.1. Overview

Although the JSR 329 standard does not define any JSF components that implementations are required to provide, Liferay Faces Bridge comes with a handful of components that are helpful during JSF portlet development.

9.2. Bridge UIComponent Tags

Liferay Faces Bridge provides the following bridge-specific UIComponent tags as part of its component suite.

Table 9.1. UIComponent Tags

Tag	Description
`bridge:inputFile`	Renders an HTML `<input type="file" />` tag which provides file upload capability.

9.2.1. The bridge:inputFile tag

The `bridge:inputFile` tag renders an HTML `<input type="file" />` tag which enables file upload capability.

Dependency on JARs from apache.org

Usage of this tag requires the Apache `commons-fileupload` and `commons-io` dependencies. See the Demo JSF2 Portlet at liferay.com for more details.

Table 9.2. Attributes

Attribute	Type	Description	Required
id	String	The identifier of the component	false
binding	com.liferay.faces.bridge.component.HtmlInputFile	An EL expression that represents a JavaBean proeprty for the corresponding HtmlInputFile component in the runtime component tree.	true
rendered	Boolean	Boolean flag indicating whether or not this component is to be rendered during the RENDER_RESPONSE phase of the JSF lifecycle. The default value is “true”.	false

Example 9.1. Example usage of bridge:inputFile tag

```
<?xml version="1.0" encoding="UTF-8"?>
<f:view xmlns="http://www.w3.org/1999/xhtml"
 xmlns:bridge="http://liferay.com/faces/bridge"
 xmlns:f="http://java.sun.com/jsf/core"
 xmlns:h="http://java.sun.com/jsf/html"
 xmlns:ui="http://java.sun.com/jsf/facelets">
 <h:head />
 <h:body">
  <h:form>
   <bridge:inputFile binding="#{backingBean.attachment1}" />
   <h:commandButton
 actionListener="#{backingBean.uploadAttachments}"
    value="Submit" />
  </h:form>
 </h:body>
</f:view>
```

Example 9.2. Backing Bean Java Code

```
@ManagedBean(name = "backingBean")
@ViewScoped
public class BackingBean implements Serializable {

 private transient HtmlInputFile attachment1;
 public void uploadAttachments(ActionEvent actionEvent) {

  UploadedFile uploadedFile1 = attachment1.getUploadedFile();
  System.err.println("Uploaded file:" + uploadedFile1.getName());
 }
}
```

9.3. Portlet 2.0 UIComponent Tags

Liferay Faces Bridge provides the following portlet UIComponent tags as part of its component suite.

JSF 2 Facelets

Although JSP tags are provided by the portlet container implementation, Liferay Faces Bridge provides these tags in order to support their usage within Facelets.

Table 9.3. UIComponent Tags

Tag	Description
`portlet:actionURL`	If the var attribute is present, introduces an EL variable that contains a javax.portlet.ActionURL adequate for postbacks. Otherwise, the URL is written to the response.
`portlet:namespace`	If the var attribute is present, introduces an EL variable that contains a the portlet namespace. Otherwise, the namespace is written to the response.
`portlet:param`	Provides the ability to add a request parameter name=value pair when nested inside portlet:actionURL, portletRenderURL, or portlet:resourceURL tags.
`portlet:renderURL`	If the var attribute is present, introduces an EL variable that contains a javax.portlet.PortletURL adequate for rendering. Otherwise, the URL is written to the response.
`portlet:resourceURL`	If the var attribute is present, introduces an EL variable that contains a javax.portlet.ResourceURL adequate for rendering. Otherwise, the URL is written to the response.

9.3.1. The portlet:actionURL tag

If the var attribute is present, the `portlet:actionURL` tag introduces an EL variable that contains a javax.portlet.ActionURL adequate for postbacks. Otherwise, the URL is written to the response.

Table 9.4. Attributes

Attribute	Type	Description	Required
id	String	The identifier of the component	false
rendered	Boolean	Boolean flag indicating whether or not this component is to be rendered during the RENDER_RESPONSE phase of the JSF lifecycle. The default value is “true”.	false
var	String	Specifies the name of a variable that will be introduced into the EL.	false

Example 9.3. Example usage of portlet:actionURL tag

```
<?xml version="1.0" encoding="UTF-8"?>
<f:view xmlns="http://www.w3.org/1999/xhtml"
 xmlns:f="http://java.sun.com/jsf/core"
 xmlns:h="http://java.sun.com/jsf/html"
 xmlns:portlet="http://java.sun.com/portlet_2_0"
 xmlns:ui="http://java.sun.com/jsf/facelets">
 <h:head />
 <h:body">
  <h:form>
   <portlet:actionURL var="myActionURL" >
    <portlet:param name="foo" value="1234" />
   </portlet:actionURL>
   <h:outputText var="actionURL=#{myActionURL}" />
  </h:form>
 </h:body>
</f:view>
```

9.3.2. The portlet:namespace tag

If the var attribute is present, the `portlet:namespace` tag introduces an EL variable that contains the portlet namespace. Otherwise, the namespace is written to the response.

Table 9.5. Attributes

Attribute	Type	Description	Required
id	String	The identifier of the component	false
rendered	Boolean	Boolean flag indicating whether or not this component is to be rendered during the RENDER_RESPONSE phase of the JSF lifecycle. The default value is “true”.	false
var	String	Specifies the name of a variable that will be introduced into the EL.	false

Example 9.4. Example usage of portlet:actionURL tag

```
<?xml version="1.0" encoding="UTF-8"?>
<f:view xmlns="http://www.w3.org/1999/xhtml"
 xmlns:f="http://java.sun.com/jsf/core"
 xmlns:h="http://java.sun.com/jsf/html"
 xmlns:portlet="http://java.sun.com/portlet_2_0"
 xmlns:ui="http://java.sun.com/jsf/facelets">
 <h:head />
 <h:body">
  <h:form>
   <portlet:namespace var="mynamespace" />
   <h:outputText var="namespace=#{mynamespace}" />
  </h:form>
 </h:body>
</f:view>
```

9.3.3. The portlet:param tag

The `portlet:param` tag provides the ability to add a request parameter name=value pair when nested inside portlet:actionURL, portletRenderURL, or portlet:resourceURL tags.

Table 9.6. Attributes

Attribute	Type	Description	Required
id	String	The identifier of the component	false
name	String	The name of the parameter.	true
rendered	Boolean	Boolean flag indicating whether or not this component is to be rendered during the RENDER_RESPONSE phase of the JSF lifecycle. The default value is “true”.	false
value	String	The value of the parameter.	true

Example 9.5. Example usage of portlet:actionURL tag

```
<?xml version="1.0" encoding="UTF-8"?>
<f:view xmlns="http://www.w3.org/1999/xhtml"
 xmlns:f="http://java.sun.com/jsf/core"
 xmlns:h="http://java.sun.com/jsf/html"
 xmlns:portlet="http://java.sun.com/portlet_2_0"
 xmlns:ui="http://java.sun.com/jsf/facelets">
 <h:head />
 <h:body">
  <h:form>
   <portlet:actionURL>
    <portlet:param name="foo" value="1234" />
   </portlet:actionURL>
  </h:form>
 </h:body>
</f:view>
```

9.3.4. The portlet:renderURL tag

If the var attribute is present, the `portlet:renderURL` tag introduces an EL variable that contains a javax.portlet.PortletURL adequate for rendering. Otherwise, the URL is written to the response.

Table 9.7. Attributes

Attribute	Type	Description	Required
id	String	The identifier of the component	false
rendered	Boolean	Boolean flag indicating whether or not this component is to be rendered during the RENDER_RESPONSE phase of the JSF lifecycle. The default value is “true”.	false
var	String	Specifies the name of a variable that will be introduced into the EL.	false

Example 9.6. Example usage of portlet:renderURL tag

```
<?xml version="1.0" encoding="UTF-8"?>
<f:view xmlns="http://www.w3.org/1999/xhtml"
 xmlns:f="http://java.sun.com/jsf/core"
 xmlns:h="http://java.sun.com/jsf/html"
 xmlns:portlet="http://java.sun.com/portlet_2_0"
 xmlns:ui="http://java.sun.com/jsf/facelets">
 <h:head />
 <h:body">
  <h:form>
   <portlet:renderURL var="myRenderURL">
    <portlet:param name="foo" value="1234" />
   </portlet:renderURL>
   <h:outputText var="actionURL=#{myRenderURL}" />
  </h:form>
 </h:body>
</f:view>
```

9.3.5. The portlet:resourceURL tag

If the var attribute is present, the `portlet:resourceURL` tag introduces an EL variable that contains a javax.portlet.ActionURL adequate for obtaining resources. Otherwise, the URL is written to the response.

Table 9.8. Attributes

Attribute	Type	Description	Required
id	String	The identifier of the component	false
rendered	Boolean	Boolean flag indicating whether or not this component is to be rendered during the RENDER_RESPONSE phase of the JSF lifecycle. The default value is “true”.	false
var	String	Specifies the name of a variable that will be introduced into the EL.	false

Example 9.7. Example usage of portlet:resourceURL tag

```
<?xml version="1.0" encoding="UTF-8"?>
<f:view xmlns="http://www.w3.org/1999/xhtml"
 xmlns:f="http://java.sun.com/jsf/core"
 xmlns:h="http://java.sun.com/jsf/html"
 xmlns:portlet="http://java.sun.com/portlet_2_0"
 xmlns:ui="http://java.sun.com/jsf/facelets">
 <h:head />
 <h:body">
  <h:form>
   <portlet:resourceURL var="myResourceURL">
    <portlet:param name="foo" value="1234" />
   </portlet:resourceURL>
   <h:outputText var="actionURL=#{myResourceURL}" />
  </h:form>
 </h:body>
</f:view>
```

Chapter 10. Liferay Portal

10.1. Overview

Liferay Portal is an open source portal server that implements the Portlet 2.0 standard. The project home page can be found at http://www.liferay.com.

While Liferay Faces Bridge is theoretically compatible with any portal that implements the Portlet 2.0 standard, it has been carefully tested for use with Liferay Portal 5.2 and Liferay Portal 6.0 and has several optimizations that provide increased performance within Liferay.

10.2. JavaScript Concerns

When any JSF 2 portlet dynamically is added to a portal page at runtime by the end-user, the JSF 2 standard jsf.js JavaScript code will not be executed unless there is a full page refresh.

As a workaround, Liferay Portal provides configuration parameters that allow the developer to specify that a full page refresh is required. Doing this ensures that JSF 2 is properly initialized. The required parameters, render-weight and ajaxable, are specified in the WEB-INF/liferay-portlet.xml configuration file

Example 10.1. Specifying that a full page refresh should take place after the JSF 2 portlet is first added to the portal page

```
<liferay-portlet-app>
 <portlet>
  <portlet-name>my_portlet</portlet-name>
  <instanceable>false</instanceable>
  <render-weight>1</render-weight>
  <ajaxable>false</ajaxable>
 </portlet>
</liferay-portlet-app>
```

Chapter 11. ICEfaces 3 Portlet Development

11.1. Overview

ICEfaces 3 is an open source extension to JSF that enables developers with Java EE application skills to build Ajax-powered Rich Internet Applications (RIA) without writing any JavaScript code. The product contains a robust suite of Ajax-enabled JSF UI components, and also supports a broad array of Java application servers, IDEs, third party components, and JavaScript effect libraries. The project home page can be found at http://www.icesoft.org/projects/ICEfaces.

ICEfaces 3 introduces automatic-Ajax into JSF 2 applications, which makes it a particularly good choice for developing RIA portlets. Consider a portal page that contains two portlets: Portlet A and Portlet B. When submitting a form in Portlet A, an HTTP POST takes place and the entire portal page is refreshed. This can result in a disruptive end-user experience if the user had entered data in Portlet B prior to submitting Portlet A. ICEfaces 3 allows you to combat this disruptive experience with RIA features in your portlets.

11.2. ICEfaces Direct-To-DOM RenderKit

Rather than writing markup directly to the response, ICEfaces 3 components render themselves into a server-side Document Object Model (DOM) via the Direct-to-DOM (D2D) RenderKit. When a JSF view is requested for the first time, the markup inside the server-side DOM is delivered to the browser as part of the response. As the user interacts with the UI of the portlet, ICEfaces transparently submits user actions via Ajax and executes the JSF lifecycle. When the RENDER_RESPONSE phase of the JSF lifecycle completes, ICEfaces will compare the previous server-side DOM with the latest server-side DOM and send the incremental page updates back to the browser via the ICEfaces Ajax Bridge.

This approach is sometimes referred to as “dom-diffing” and provides the following benefits for portlets:

- The end user is immediately presented with form validation failures for visited fields when pressing the tab key
- When a navigation-rule fires and a new JSF view is to be rendered, ICEfaces will render the new JSF view by performing a complete update of

the markup contained in the affected portlet, rather than causing the entire browser page to reload

- ICEfaces portlets will not disturb other portlets on the same portal page

11.3. ICEfaces Ajax Push and Inter-Portlet Communication

While the Portlet 2.0 standard defines techniques for performing inter-portlet communication (IPC), they cause either an HTTP GET or HTTP POST which results in a full page reload and a disruptive end-user experience. ICEfaces provides a natural way for portlets to perform IPC via ICEfaces Ajax Push.

ICEfaces pioneered Ajax Push, which is sometimes referred to as Reverse Ajax or Comet. The technology provides the ability for server-initiated events to cause incremental page updates to be sent to the browser. With ICEfaces Ajax Push, developers can create collaborative and dynamic enterprise applications like never before.

Because the mechanism facilitates asynchronous updates from the server to the client, interaction with one ICEfaces portlet can trigger communication with other ICEfaces portlets by changing values in JSF backing and model managed-beans. This mechanism is not restricted to IPC among portlets on the same portal page in a single browser, but can include updating other browsers that are interacting with the same portal page. The result is not just inter-portlet communication, but inter-portlet, inter-browser communication. Additionally, ICEfaces Ajax Push solves the potentially disruptive end-user experience associated with the Portlet 2.0 standard IPC techniques.

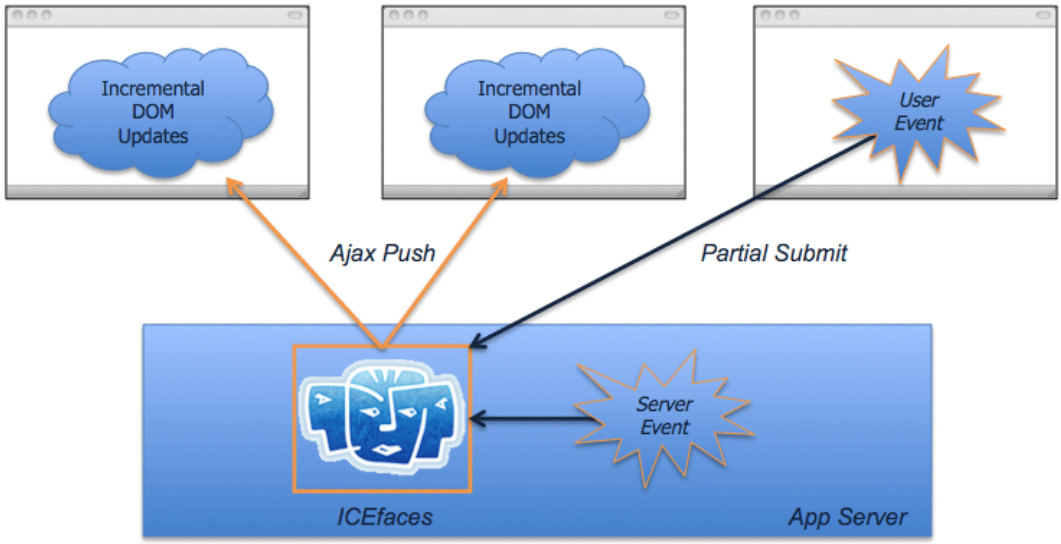

Figure 11.1. Illustration of IPC with ICEfaces Ajax Push

The following is a list of guidelines for achieving IPC with ICEfaces Ajax Push

- Package the portlets that need to communicate in the same WAR.
- In order to share data between portlets, use application-scoped beans or request-scoped beans that store data in PortletSession.APPLICATION_SCOPE.
- Use the ICEfaces Ajax Push SessionRenderer to trigger client updates when the shared data changes.

Example 11.1. Chat Portlet Managed-Bean

```
/*
* This is a file named ChatRoomManagedBean.java
* that is registered as a JSF managed-bean in
* application scope. It maintains a chat log
* that participates in ICEfaces Ajax Push using
* the SessionRenderer.
*/
@ManagedBean(name = "chatRoomManagedBean")
@RequestScoped
public class ChatRoomManagedBean {

 private String messageText;

 private List<String> messages = new ArrayList<String>();

 private static final String
  AJAX_PUSH_GROUP_NAME = "chatRoom";

 public ChatRoomsModel() {
  SessionRenderer.addCurrentSession(
   AJAX_PUSH_GROUP_NAME);
 }

 @PreDestroy
 public void preDestroy() throws Exception {
  SessionRenderer.removeCurrentSession(
   AJAX_PUSH_GROUP_NAME);
 }

 public void addMessage(ActionEvent actionEvent) {
  messages.add(messageText);
  SessionRenderer.render(AJAX_PUSH_GROUP_NAME);
 }

 public List<String> getMessages() {
  return messages;
 }

 public String getMessageText() {
  return messageText;
 }

 public void setMessageText(String messageText) {
  this.messageText = messageText;
 }
}
```

Example 11.2. Chat Portlet Facelet View

```
<!-- This is a Facelet view named chatRoom.xhtml -->
<?xml version="1.0" encoding="UTF-8"?>
<f:view xmlns="http://www.w3.org/1999/xhtml"
 xmlns:f="http://java.sun.com/jsf/core"
 xmlns:h="http://java.sun.com/jsf/html"
 xmlns:ui="http://java.sun.com/jsf/facelets">
 <h:head />
 <h:body styleClass="example-icefaces-portlet">
  <h:form>
   <h:dataTable
    value="#{chatRoomManagedBean.messages}"
    var="message">
    <h:column>
     <h:outputText value="#{message}" />
    </h:column>
   </h:dataTable>
   <h:inputText
    value="#{chatRoomManagedBean.messageText}" />
   <h:commandButton
    actionListener="#{chatRoomManagedBean.addMessage}" />
  </h:form>
 </h:body>
</f:view>
```

Demo Portlets at liferay.com

The liferay.com website hosts a variety open source demonstration portlets that focus on ICEfaces portlets for Liferay Portal. Specifically, there is a portlet featuring an ICEfaces chat room that integrates with Liferay Portal's online users. When a user signs-in to Liferay Portal, a server-initiated event triggers ICEfaces Ajax Push so that other users that are online become aware of online presence. Additionally, when the user clicks on a "Chat" icon, ICEfaces Ajax Push is used for IPC to begin a new chat room in a Chat Portlet. For more information, visit :

http://www.liferay.com/community/liferay-projects/liferay-faces/demos

11.4. ICEfaces Themes and Portal Themes

The ICEfaces Component Suite fully supports consistent component styling via a set of predefined CSS style classes and associated images. Changing the component styles for a web application developed with the ICEfaces Component Suite is as simple as changing the style sheet used. ICEfaces ships with a set of predefined style sheets are available to be used as-is, or customized to meet the specific requirements of the application. There are five predefined ICEfaces style sheets included, two of which are designed to be used inside a portlet container:

- rime.css
- rime-portlet.css
- xp.css
- xp-portlet.css
- royale.css

Example 11.3. Specifying the portlet-compatible version of the ICEfaces "XP" theme

```
<?xml version="1.0" encoding="UTF-8"?>
<f:view xmlns="http://www.w3.org/1999/xhtml"
 xmlns:f="http://java.sun.com/jsf/core"
 xmlns:h="http://java.sun.com/jsf/html"
 xmlns:ui="http://java.sun.com/jsf/facelets">
 <h:head>
 <link href="#{request.contextPath}/xmlhttp/css/xp/xp-portlet.css"
 rel="stylesheet"
  type="text/css" />
 </h:head>
 <h:body styleClass="example-icefaces-portlet">
  <h:form>
   ...
  </h:form>
 </h:body>
</f:view>
```

The Portlet 1.0 and 2.0 standards document a set of common CSS class names that should be applied to specific page elements in order to integrate with the portlet container's theme mechanism. When running in a portlet

container, ICEfaces 1.8 compatibility components will automatically render the following subset of Portlet 1.0 CSS class names where appropriate

- portlet-form-button
- portlet-form-field
- portlet-form-input-field
- portlet-form-label
- portlet-menu
- portlet-menu-cascade-item
- portlet-menu-item
- portlet-menu-item-hover
- portlet-section-alternate
- portlet-section-body
- portlet-section-footer
- portlet-section-header
- portlet-msg-alert
- portlet-msg-error
- portlet-msg-info

To disable this feature, developers can specify the com.icesoft.faces.portlet.renderStyles context parameter in the WEB-INF/web.xml configuration file and set its value to false.

Example 11.4. Disabling automatic rendering of Portlet 1.0 / 2.0 standard CSS class names in the WEB-INF/web.xml configuration file

```
<context-param>
 <param-name>
  com.icesoft.faces.portlet.renderStyles
 </param-name>
 <param-value>false</param-value>
</context-param>
```

11.5. ICEfaces Themes and Liferay Themes

Liferay Portal supports styling for Portlet 1.0 and 2.0 standard CSS class names as well as a set of vendor-specific CSS class names within the context of a Liferay theme. However, since Liferay themes do not contain styling for the ICEfaces Component Suite, it is necessary to select an ICEfaces style sheet that is visually compatible with the Liferay theme.

On some occasions, it becomes necessary to override some of the styling in a Liferay theme in order to make it more visually compatible with an ICEfaces portlet. For example, Liferay themes typically render spans of class portlet-msg-error with a margin that has too much space to be placed alongside a rendered ice:inputText component tag.

Example 11.5. Overriding styling in a Liferay theme from within a Facelet view so that rendered output from ice:messages and ice:message have a more narrow margin

```
<?xml version="1.0" encoding="UTF-8"?>
<f:view xmlns="http://www.w3.org/1999/xhtml"
 xmlns:f="http://java.sun.com/jsf/core"
 xmlns:h="http://java.sun.com/jsf/html"
 xmlns:ui="http://java.sun.com/jsf/facelets">
 <h:head>
 <link href="#{request.contextPath}/xmlhttp/css/xp/xp-portlet.css"
 rel="stylesheet"
  type="text/css" />
 <link href="#{request.contextPath}/css/liferay-theme-override.css"
 rel="stylesheet"
  type="text/css" />
 </h:head>
 <h:body styleClass="example-icefaces-portlet">
  <h:form>
   <h:messages />
   ...
  </h:form>
 </h:body>
</f:view>

/*
 * This is a separate file named liferay-theme-override.css
 */
.example-icefaces-portlet .portlet-msg-error {
 margin: 1px 0px 0px 0px;
 padding: 1px 5px 1px 24px;
}
```

Chapter 12. Overview

Liferay Faces Portal is a JAR that JSF developers can add as a dependency to their portlet WAR projects in order to utilize Liferay-specific utilities and UI components.

The project home page can be found at http://www.liferay.com/community/liferay-projects/liferay-faces/portal

Chapter 13. LiferayFacesContext

JSF web application developers typically call FacesContext.getCurrentInstance() in order to obtain the `ThreadLocal` singleton instance associated with the current request. While JSF portlet developers can certainly do the same, it's easier to call LiferayFacesContext.getInstance() which returns an application-scoped singleton instance.

Example 13.1. Obtaining the LiferayFacesContext Singleton Instance

```
public class SessionScopedManagedBean {

 private final LiferayFacesContext liferayFacesContext =
 LiferayFacesContext.getInstance();

 public List<DlFileEntry> getDocuments() {
  List<DLFileEntry> documents;
  try {
   documents = DLFileEntryLocalServiceUtil.getFileEntries(folderId);
  }
  catch (Exception e) {
   logger.error(e.getMessage(), e);
   // Don't have to call LiferayFacesContext.getInstance() first
 since
   // a reference to it was obtained when the bean was created.
   liferayFacesContext.addGlobalUnexpectedErrorMessage();
  }
  return documents;
 }
}
```

LiferayFacesContext is an abstract class that extends the JSF FacesContext abstract class. Because of thism it supplies all the same method signatures and can therefore do anything that FacesContext can do. The LiferayFacesContext implements the delegation design pattern for methods defined by FacesContext by first calling FacesContext.getCurrentInstance() and then delegating to corresponding methods. The benefit of using this technique is that JSF portlet developers only have to call LiferayFacesContext.getInstance() once, and can save the singleton object reference for future use.

Chapter 14. Liferay Faces Portal Expression Language Additions

Liferay Faces Portal introduces several variables into the Expression Language (EL).

	Type: `com.liferay.portal.security.permission.Permiss`
`liferay.portalURL`	The absolute URL for the portal. *Type:* `String`
`liferay.portlet`	the containing Liferay `Portlet` associated with the `PortletRequest`. *Type:* `com.liferay.portal.model.Portlet`
`liferay.portraitURL`	Designed to be called from the EL by passing a Liferay `User` or `userId` as an array index, returns the absolute URL to the user's portrait. *Type:* `String`
`liferay.theme`	The Liferay `Theme` associated with the Liferay `Layout`. *Type:* `com.liferay.portal.model.Theme`
`liferay.themeDisplay`	The Liferay `ThemeDisplay` associated with the `PortletRequest`. *Type:* `com.liferay.portal.theme.ThemeDisplay`
`liferay.themeImageURL`	Designed to be called from the EL by passing a relative path to a theme image as an array index, returns the absolute URL to the theme image. *Type:* `String`
`liferay.themeImagesURL`	The absolute URL for the image path associated with the current Liferay Theme. *Type:* `String`
`liferay.user`	The Liferay `User` associated with the `PortletRequest`. *Type:* `com.liferay.portal.model.User`
`liferay.userHasPortletPermission`	Designed to be called from the EL by passing an `action-key` as an array index, returns a `Boolean` indicating whether or not the Liferay `User` associated with the `PortletRequest` has permission to execute the specified `action-key` on the current portlet. *Type:* `Boolean`

Table 14.1. Liferay Faces Portal EL Variables

14.1. i18n

As an abbreviation for the word "internationalization", the `i18n` EL variable enables page authors to declaratively specify message keys that are provided by one of the following:

- Liferay's Language Utility
- Portlet WAR additions to Liferay's Language Utility
- The JSF standard message keys

The Liferay Language Utility is typically accessed by portlet developers by calling static Java methods found in the `LanguageUtil` class. The utility operates by reading the locale-specific version of the portal's Language.properties file, which contains thousands of keys and internationalized messages.

Portlet developers can extend the Liferay Language Utility by creating a file within the portlet WAR named `WEB-INF/liferay-hook.xml` that points to locale-specific resource bundles that are in the runtime classpath of the portlet.

Example 14.1. WEB-INF/liferay-hook.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hook PUBLIC "-//Liferay//DTD Hook 6.0.0//EN"
 "http://www.liferay.com/dtd/liferay-hook_6_0_0.dtd">
<hook>
 <language-properties>Language_en_US.properties</language-
properties>
</hook>
```

Example 14.2. Contents of Language_en_US.properties

```
add-new-entry=Add New Entry
save-entry=Save Entry
```

Example 14.3. Usage of the i18n EL Variable

```
<h:outputLabel value="#{i18n['first-name']}" />
```

When using JBoss EL, page authors can take advantage of the `i18n.replace()` method in order to substitute values into the text of the message.

Example 14.4. Usage of the i18n EL Variable with JBoss EL

```
<!--
Note: The US English translation of the x-has-x-friends key would
 look like the following:
x-has-x-friends={0} has {1} friends.
-->
<h:outputText value="#{i18n.replace('x-has-x-friends',
 liferay.groupUser.fullName, friendsModel.dataModel.rowCount)}" />
```

14.2. liferay

This is a utility managed-bean that is designed to be kept in request scope. Its purpose is to introduce some Liferay-specific variables into the JSF EL. The reason why this is implemented as a managed-bean (and not as an ELResolver) is because it needs to kept in JSF 2 ViewScope.

14.3. liferay.companyId

The Liferay companyId primary key value associated with the community/organization portal page that the current portlet is placed upon.

Example 14.5. EL Usage of liferay.companyId

```
<h:outputText value="#{liferay.companyId} is the companyId
 associated with this set of Liferay Portal pages." />
```

14.4. liferay.documentLibraryURL

The absolute URL for the Liferay Document Library Struts action path prefix. The most common use case is to append the /get_file suffix and some additional request parameters in order to provide a hyperlink to a document in the Liferay Document Library. See the Liferay struts-config.xml file for a complete list of available suffixes.

Example 14.6. EL Usage of liferay.documentLibraryURL

```
<h:dataTable id="documents" value="#{documentModelBean.dataModel}"
 var="dlFileEntry">
 <h:column>
  <f:facet name="head">
   <h:outputText value="#{i18n['file-name']}" />
  </f:facet>
  <h:outputLink
   target="_blank"

 value="#{liferay.documentLibraryURL}/
get_file?p_l_id=#{liferay.themeDisplay.plid}&folderId=#{dlFileEntry.folderId}&name=#{d
   <h:outputText value="#{dlFileEntry.title}" />
  </h:outputLink>
 </h:column>
</h:dataTable>
```

14.5. liferay.groupUser

The Liferay `User` that owns the Liferay community/organization portal page that the current portlet is placed upon.

Example 14.7. EL Usage of liferay.groupUser

```
<h:outputText
 value="The user named #{liferay.groupUser.fullName} owns this set
 of Liferay Portal pages." />
```

14.6. liferay.imageGalleryURL

The absolute URL for the Liferay Image Gallery Struts action path prefix. See the Liferay struts-config.xml file for a complete list of available suffixes.

14.7. liferay.imageURL

The absolute URL for the Liferay Image Servlet. Although this can be used to construct a URL that points a Liferay user's portrait/photo, for performance reasons, it is better to use the `portraitURL` EL variable instead.

14.8. liferay.layout

The Liferay `Layout` associated with the community/organization portal page that the current portlet is placed upon.

Example 14.8. EL Usage of liferay.layout

```
<h:outputText
 value="The name of this portal page is #{liferay.layout.name}" />
```

14.9. liferay.permissionChecker

The Liferay `PermissionChecker` associated with the current request and Liferay `User`.

Example 14.9. EL Usage of liferay.permissionChecker

```
<h:commandButton actionListener="#{backingBean.save}"
 rendered="#{liferay.permissionChecker.companyAdmin}"
 value="#{i18n['save']}" />
```

14.10. liferay.portalURL

The absolute URL for the portal. For example: http://localhost:8080

14.11. liferay.portlet

The containing Liferay `Portlet` associated with the `PortletRequest`.

Example 14.10. EL Usage of liferay.portlet

```
<h:outputText
 value="The name of this portlet is #{liferay.portlet.displayName}"
 />
```

14.12. liferay.portraitURL

Designed to be called from the EL by passing a Liferay `User` or `userId` as an array index, returns the absolute URL to the user's portrait.

Example 14.11. EL Usage of liferay.portraitURL

```
<h:graphicImage value="#{liferay.portraitURL[liferay.group.user]}"
 />
```

14.13. liferay.theme

The Liferay `Theme` associated with the Liferay `Layout`.

Example 14.12. EL Usage of liferay.theme

```
<h:outputText value="The name of the Liferay theme applied to this
 portal page is #{liferay.theme.name}" />
```

14.14. liferay.themeDisplay

The Liferay `ThemeDisplay` associated with the `PortletRequest`. Perhaps it is easier to think of the Liferay ThemeDisplay as a "display context" which provides access to a wealth of information including the current Company, User, Layout, Theme, PermissionChecker, and more.

Example 14.13. EL Usage of liferay.themeDisplay

```
<link
 href="#{liferay.themeDisplay.uRLSignIn}">#{i18n['sign-in']}</link>
```

14.15. liferay.themeImageURL

Designed to be called from the EL by passing a relative path to a theme image as an array index, returns the absolute URL to the theme image.

Example 14.14. EL Usage of liferay.themeImageURL

```
<h:graphicImage
 value="#{liferay.themeImageURL['/common/delete.png']}" />
```

14.16. liferay.themeImagesURL

Returns the absolute URL for the image path associated with the current Liferay Theme. For example: http://localhost:8080/image/image_gallery.

Example 14.15. EL Usage of liferay.themeImagesURL

```
<h:graphicImage value="#{liferay.themeImagesURL}/common/delete.png"
 />
```

14.17. liferay.user

the Liferay `User` associated with the `PortletRequest`.

Example 14.16. EL Usage of liferay.user

```
<h:outputText value="#{i18n['welcome']}, #{liferay.user.firstName}"
 />
```

14.18. liferay.userHasPortletPermission

Designed to be called from the EL by passing an `action-key` as an array index, returns a `Boolean` indicating whether or not the Liferay `User` associated with the `PortletRequest` has permission to execute the specified `action-key` on the current portlet. The `action-key` is typically defined in a Liferay `resource-action-mapping` XML file that defines the `<portlet-resource/>` and `<model-resource/>` permissions associated with a Liferay portlet. Please refer to the portal-impl/classes/resource-actions/messageboards.xml file in the Liferay Portal source code distribution for an example of how to write a Liferay `resource-action-mapping` XML file.

Example 14.17. EL Usage of liferay.userHasPortletPermission

```
<h:dataTable
 rendered="#{liferay.userHasPortletPermission['VIEW']}"
 value="#{modelManagedBean.users}"
 var="user">
</h:dataTable>
```

Chapter 15. Liferay Faces Portal UIComponent Tags

Liferay Faces Portal provides the following UIComponent tags as part of its component suite.

Table 15.1. UIComponent Tags

Tag	Description
`liferay-ui:input-editor`	Renders a text area that provides the ability to enter rich text such as bold, italic, and underline.
`liferay-security:permissionsURL`	Renders an HTML anchor tag (hyperlink) that the user can click on in order to see the Liferay Permissions screen for the associated resource.

15.1. The liferay-ui:input-editor tag

The `liferay-ui:input-editor` tag renders a text area that provides the ability to enter rich text such as bold, italic, and underline. The renderer relies on the CKEditor™ to provide the rich text editing area. Since Liferay bundles the CKEditor™ JavaScript and related images with the portal, the portlet developer does not need to include it with the portlet.

Note

Prior to Liferay 6.0 SP2 (6.0.12), the rich text area was rendered as an <iframe>. But due to incompatibilities with IE, the rich text area HTML markup is now rendered "inline" with the portal page. Liferay Faces Portal will automatically detect the version of Liferay and will render the rich text area accordingly. However, if you using Liferay 6.0 (6.0.10) or Liferay 6.0 SP1 (6.0.11) and have received an "inline" patch from Liferay Support, then you will need to add the following to the portlet's WEB-INF/web.xml descriptor:

```
<context-param>

 <param-name>com.liferay.faces.portal.inlineInputEditor</
param-name>
    <param-value>true</param-value>
```

```
</context-param>
```

Also, if the editor is "inline" and you are using a Servlet 2.5 container such as Tomcat 6, then it is necessary to add the following markup to the portlet's WEB-INF/web.xml deployment descriptor:

```
<listener>

 <listener-
class>com.liferay.faces.portal.listener.StartupListener</
listener-class>
</listener>
```

If using ICEfaces, then the "inline" version of liferay-ui:input-editor will expose an inefficiency in the Direct2DOM™ (DOM-diff) algorithm. Typing a single character in the rich text area will cause ICEfaces to detect a DOM-diff, causing the entire liferay-ui:input-editor to be replaced in the browser's DOM with the form is submitted via Ajax. The workaround for this problem is to use the JSF2 f:ajax component to optimize/control which parts of the JSF component tree are DOM-diffed by ICEfaces. For example:

```
<h:panelGroup id="feedback">
    <h:messages globalOnly="true" layout="table" />
</h:panelGroup>
<h:panelGroup id="editor">
    <liferay-ui:input-editor value="#{modelBean.text}" />
</h:panelGroup>
<h:commandButton>
    <f:ajax execute="@form" render="feedback" />
</h:commandButton>
```

Table 15.2. Attributes

Attribute	Type	Description	Required
id	String	The identifier of the component	false
rendered	Boolean	Boolean flag indicating whether or not this component is to be rendered during the RENDER_RESPONSE phase of the JSF lifecycle. The default value is “true”.	false
value	String	The value of the component, the HTML fragment generated by the end-user.	true

Example 15.1. Example usage of liferay-ui:input-editor tag

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<f:view xmlns:f="http://java.sun.com/jsf/core"
 xmlns:h="http://java.sun.com/jsf/html"
 xmlns:liferay-ui="xmlns:liferay-ui="http://liferay.com/faces/ui">

 <h:form>
  <liferay-ui:input-editor id="comments"
 value="#{modelManagedBean.comments}" />
 </h:form>

</f:view>
```

15.2. The liferay-security:permissionsURL tag

The `liferay-security:permissionsURL` tag renders an HTML anchor tag (hyperlink) that the user can click on in order to see the Liferay Permissions screen for the associated resource.

Table 15.3. Attributes

Attribute	Type	Description	Required
id	String	The identifier of the component	false
modelResource	String	The fully qualified Java class of the model resource. For example: `MyModelResource.class.getName()`	true
modelResourceDescription	String	The description of the model resource.	false
redirect	Boolean	The redirect URL.	false
rendered	Boolean	Boolean flag indicating whether or not this component is to be rendered during the RENDER_RESPONSE phase of the JSF lifecycle. The default value is “true”.	false
resourcePrimKey	String	The primary key value of the model resource.	false
value	String	The text of the hyperlink that the user will see and click on.	false

Example 15.2. Example usage of liferay-security:permissionsURL tag

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<f:view xmlns:f="http://java.sun.com/jsf/core"
 xmlns:h="http://java.sun.com/jsf/html"
 xmlns:liferay-security="xmlns:liferay-ui="http://liferay.com/faces/
security">

 <h:form>
  <liferay-security:permissionsURL
   modelResource="myproject.model.Book"
   modelResourceDescription="Book"
   resourcePrimKey="#{modelManagedBean.book.bookId} />
 </h:form>

</f:view>
```

Chapter 16. Liferay Faces Portal Composite Component Tags

Liferay Faces Portal provides the following Facelet Composite Component tags as part of its component suite.

Table 16.1. Facelet Composite Component Tags

Tag	Description
`liferay-ui:ice-info-data-paginator`	Encapsulates an ICEfaces ice:dataPaginator tag that renders pagination information for an associated ice:dataTable. The navigation information will match the internationalized Liferay "showing-x-x-of-x-results" message.
`liferay-ui:ice-nav-data-paginator`	Encapsulates an ICEfaces ice:dataPaginator tag that renders navigation controls for an associated ice:dataTable. The icons will match the current Liferay theme.
`liferay-ui:icon`	Encapsulates an HTML `img` tag whose `src` attribute contains a fully qualified URL to an icon in the Liferay theme.

16.1. The liferay-ui:ice-info-data-paginator tag

The `liferay-ui:ice-info-data-paginator` encapsulates an ICEfaces ice:dataPaginator tag that renders pagination information for an associated ice:dataTable. The navigation information will match the internationalized Liferay "showing-x-x-of-x-results" message.

16.2. The liferay-ui:ice-nav-data-paginator tag

[illegible]	String	Corresponds to the value of an id attribute for an ice:dataTable tag.	true
lastIconRendered	Boolean	Boolean flag indicating whether or not the "Last" button/icon is rendered. The default value is "true".	false
nextIconRendered	Boolean	Boolean flag indicating whether or not the "Next" button/icon is rendered. The default value is "true".	false
paginator	Boolean	Boolean flag indicating whether or not the page number links will be rendered. This is a pass-through attribute for the encapsulated ice:dataPaginator. The default value is "true".	false
paginatorMaxPages	Integer	The maximum amount of pages to be displayed in the paginator. This is a pass-through attribute for the encapsulated ice:dataPaginator. The default value is 7.	false
previousIconRendered	Boolean	Boolean flag indicating whether or not the "Previous" button/icon is rendered. The default value is "true".	false

Example 16.1. Example usage of liferay-ui:ice-info-data-paginator tag

```
<?xml version="1.0" encoding="UTF-8" />
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<f:view xmlns:f="http://java.sun.com/jsf/core"
 xmlns:ice="http://www.icesoft.com/icefaces/component"
 xmlns:liferay-ui="http://liferay.com/faces/ui">

 <liferay-ui:ice-info-data-paginator for="dataTable1" />
 <ice:dataTable id="dataTable1" value="#{modelManagedBean.rows}"
  var="row">
  ...
 </ice:dataTable>

</f:view>
```

Example 16.2. Example usage of liferay-ui:ice-nav-data-paginator tag

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<f:view xmlns:f="http://java.sun.com/jsf/core"
 xmlns:ice="http://www.icesoft.com/icefaces/component"
 xmlns:liferay-ui="http://liferay.com/faces/ui">

 <liferay-ui:ice-nav-data-paginator for="dataTable1" />
 <ice:dataTable id="dataTable1" value="#{modelManagedBean.rows}"
 var="row">
  ...
 </ice:dataTable>

</f:view>
```

16.3. The liferay-ui:icon tag

The `liferay-ui:icon` tag encapsulates an HTML `img` tag whose `src` attribute contains a fully qualified URL to an icon image in the current Liferay theme.

Table 16.4. Attributes

Attribute	Type	Description	Required
alt	String	Corresponds to the value of the `alt` attribute for the embedded `img` tag. The default value is Liferay's internationalized message for the key named `view`.	false
image	String	The name of the theme image icon, which can be the prefix of any image filename in the Liferay theme's "common" image folder. The default value is `view`.	false

Example 16.3. Example usage of liferay-ui:icon tag

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<f:view xmlns:f="http://java.sun.com/jsf/core"
 xmlns:ice="http://www.icesoft.com/icefaces/component"
 xmlns:liferay-ui="http://liferay.com/faces/ui">

 <liferay-ui:icon alt="#{i18n['delete']}" image="delete" />

</f:view>
```

Chapter 17. Liferay Faces Portal Liferay Theme Integration

Liferay Faces Portal offers several features that help integrate JSF portlets with the current Liferay theme.

17.1. ThemeDisplay

Liferay Faces Portal provides the LiferayFacesContext.getThemeDisplay() method at the Java level and also the liferay.themeDisplay EL variable for getting access to the Liferay ThemeDisplay object.

17.2. Theme Icons

Liferay Faces Portal provides the liferay-ui:icon Facelet composite component tag that encapsulates an HTML `img` tag whose `src` attribute contains a fully qualified URL to an icon image in the current Liferay theme. Additionally, Liferay Faces Portal provides the liferay.themeImagesURL and liferay.themeImageURL Facelet composite component tags for gaining access to theme image icons.

17.3. Validation Messages (User Feedback)

Most of the standard JSF HTML component tags render themselves as HTML markup such as `<label />`, `<input />`, `<span />`, etc. and assume the current Liferay theme thanks to the power of CSS. However, the h:messages and h:message tag will not assume the current Liferay theme unless the following JSR 286 standard CSS class names are applied:

- `portlet-msg-error`
- `portlet-msg-info`
- `portlet-msg-warn`

Example 17.1. JSR 286 standard CSS class names applied to the h:messages tag

```
<h:messages errorClass="portlet-msg-error"
 fatalClass="portlet-msg-error"
 infoClass="portlet-msg-info" warnClass="portlet-msg-warn" />
```

As a convenience, Liferay Faces Portal provides the liferay-ui:messages and liferay-ui:message Facelet composite component tags that encapsulate the h:messages and h:message tags respectively, and automatically apply the JSR 286 standard class names as shown above.

Note

When running as a portlet, the ICEfaces ice:messages and ice:message component tags automatically apply the JSR 286 standard class names as shown above. Additionally the ice:dataTable component tag will apply the following JSR 286 standard class names for alternating table rows:

- `portlet-section-alternate`
- `portlet-section-body`

Chapter 18. Liferay Faces Portal Liferay Language Portlet Integration

In a normal JSF web application, the Locale that is used to display internationalized values is dictated by the locale specified in the end-user's web-browser. However, Liferay Portal permits the user to select a different locale (language) using the Language Portlet. The user's choice is ultimately saved as a `languageId` in the Liferay User object and is persisted to the database.

Figure 18.1. Liferay Portal Language Portlet

In order to provide seamless integration between JSF portlets and the language selected by the user, Liferay Faces Portal provides the LiferayLocalePhaseListener. The listener monitors the RESTORE_VIEW phase of the JSF lifecycle and will automatically set the locale inside the UIViewRoot according to the value specified by Liferay's User.getLocale() method, which is aware of the selected `languageId`. This in turn causes internationalization techniques such as the f:loadBundle tag and the i18n EL keyword to automatically translate message keys into the language selected by the user with the Liferay Language Portlet.

Chapter 19. Migration Guide

The Liferay Faces project originates from the portletfaces.org community website. On April 3, 2012 Liferay announced that it would be assuming leadership for the portletfaces.org community. Consequently, projects at portletfaces.org were repackaged under the Liferay Faces umbrella project along with some name changes:

- AlloyFaces -> Liferay Faces Alloy
- PorltetFaces -> Bridge Liferay Faces Bridge
- LiferayFaces -> Liferay Faces Portal

19.1. BridgeRequestAttributeListener

PortletFaces Bridge provided a class named `org.portletfaces.bridge.servlet.BridgeRequestAttributeListener` but Liferay Faces Bridge uses `com.liferay.faces.bridge.servlet.BridgeRequestAttributeListener`. In order to migrate to the new class, you will need to refactor to the new package namespace, as a deprecated class has not been provided.

Example 19.1. Refactoring BridgeRequestAttributeListener in WEB-INF/web.xml

```
<!-- PortletFaces Bridge BridgeRequestAttributeListener -->
<web-app>
 <listener>
  ...
  <listener-class>
   org.portletfaces.bridge.servlet.BridgeRequestAttributeListener
  </listener-class>
  ...
 </listener>
</web-app>

<!-- Liferay Faces Bridge GenericFacesPortlet -->
<web-app>
 <listener>
  ...
  <listener-class>
   com.liferay.faces.bridge.servlet.BridgeRequestAttributeListener
  </listener-class>
  ...
 </listener>
</web-app>
```

19.2. Configuration Option Names

PortletFaces Bridge provided several configuration options for use within the WEB-INF/web.xml and WEB-INF/portlet.xml descriptors. In order to ease migration, the configuration option names have been reproduced in the Liferay Faces project. It is recommended that the new configuration option names be used, as shown in the following listing:

- org.portletfaces.bridge.containerAbleToSetHttpStatusCode -> com.liferay.faces.bridge.containerAbleToSetHttpStatusCode
- org.portletfaces.bridgeRequestScopePreserved -> com.liferay.faces.bridge.bridgeRequestScopePreserved
- org.portletfaces.bridge.optimizePortletNamespace -> com.liferay.faces.bridge.optimizePortletNamespace
- org.portletfaces.bridge.preferPreDestroy -> com.liferay.faces.bridge.preferPreDestroy
- org.portletfaces.bridge.resolveXMLEntities -> com.liferay.faces.bridge.resolveXMLEntities

- org.portletfaces.bridge.resourceBufferSize -> com.liferay.faces.bridge.resourceBufferSize

19.3. File Upload

PortletFaces Bridge provided classes named `org.portletfaces.bridge.component.HtmlInputFile` and `org.portletfaces.bridge.component.UploadedFile` but Liferay Faces Bridge uses `com.liferay.faces.bridge.component.HtmlInputFile` and `com.liferay.faces.bridge.component.UploadedFile`, respectively. In order to migrate to the new classes, you will need to refactor to the new package namespace, as deprecated classes have not been provided.

Example 19.2. Refactoring HtmlInputFile

```
// PortletFaces Bridge package name:
import org.portletfaces.bridge.component.HtmlInputFile;

// Liferay Faces Bridge package name:
import com.liferay.faces.bridge.component.HtmlInputFile;
```

Example 19.3. Refactoring UploadedFile

```
// PortletFaces Bridge package name:
import org.portletfaces.bridge.component.UploadedFile;

// Liferay Faces Bridge package name:
import com.liferay.faces.bridge.component.UploadedFile;
```

19.4. Facelet Tag Library Namespaces

The projects at portletfaces.org provided several UIComponents and Composite Components for use within Facelet views. In order to ease migration, the tag library namespaces have been reproduced in the Liferay Faces project. It is recommended that the new tag library namespaces be used, as shown in the following listing:

- http://portletfaces.org/alloyfaces/aui -> http://liferay.com/faces/aui
- http://portletfaces.org/alloyfaces/aui-cc -> http://liferay.com/faces/aui-cc
- http://portletfaces.org/bridge -> http://liferay.com/faces/bridge

- http://portletfaces.org/liferayfaces/liferay-ui -> http://liferay.com/faces/ui
- http://portletfaces.org/liferayfaces/liferay-util -> http://liferay.com/faces/util
- http://portletfaces.org/liferayfaces/liferay-security -> http://liferay.com/faces/security

19.5. GenericFacesPortlet

PortletFaces Bridge provided its own `org.portletfaces.bridge.GenericFacesPortlet` class but Liferay Faces Bridge uses the JSR 329 standard `javax.portlet.faces.bridge.GenericFacesPortlet` class. In order to ease migration, the old class still exists in Liferay Faces Bridge although it has been deprecated. It is recommended that the standard class name be used in all WEB-INF/portlet.xml `portlet-class` entries.

Example 19.4. Refactoring GenericFacesPortlet in WEB-INF/portlet.xml

```
<!-- PortletFaces Bridge GenericFacesPortlet -->
<portlet-app>
 <portlet>
  ...
  <portlet-class>
   org.portletfaces.bridge.GenericFacesPortlet
  </portlet-class>
  ...
 </portlet>
</portlet-app>

<!-- Liferay Faces Bridge GenericFacesPortlet -->
<portlet-app>
 <portlet>
  ...
  <portlet-class>
   javax.portlet.faces.GenericFacesPortlet
  </portlet-class>
  ...
 </portlet>
</portlet-app>
```

19.6. LiferayFacesContext

LiferayFaces provided a class named `org.portletfaces.liferay.faces.context.LiferayFacesContext`

class but Liferay Faces Portal uses the `com.liferay.faces.portal.context.LiferayFacesContext` class. In order to ease migration, the old class still exists in Liferay Faces Portal although it has been deprecated. It is recommended that the standard class name be used instead.

Example 19.5. Refactoring LiferayFacesContext

```
// LiferayFaces package name:
import org.portletfaces.liferay.faces.context.LiferayFacesContext;

// Liferay Faces Portal package name:
import com.liferay.faces.portal.context.LiferayFacesContext;
```

19.7. Logging

The PortletFaces-Logging project at portletletfaces.org has been moved into the Liferay Faces Bridge codebase. In order to keep using this logging API in your portlets, you will need to refactor to the new package namespace, as deprecated classes have not been provided.

Example 19.6. Refactoring LoggerFactory and Logger

```
// PortletFaces-Logging package names:
import org.portletfaces.logging.LoggerFactory;
import org.portletfaces.logging.Logger;

// Liferay Faces Bridge package names:
import com.liferay.faces.bridge.logging.LoggerFactory;
import com.liferay.faces.bridge.logging.Logger;
```

19.8. Portlet Preferences

PortletFaces Bridge provided its own `org.portletfaces.bridge.preference.Preference` class but Liferay Faces Bridge uses the JSR 329 standard `javax.portlet.faces.preference.Preference` class. In order to migrate to the standard class, you will need to refactor to the new package namespace, as deprecated classes have not been provided.

Example 19.7. Refactoring PortletPreference

```
// PortletFaces Bridge package name:
import org.portletfaces.bridge.preference.Preference;

// Liferay Faces Bridge package name:
import javax.portlet.faces.preference.Preference;
```