



PRIMEFACES

THE REFERENCE GUIDE

Authors

Çagatay Çivici
Yigit Darçın

Build: 30.11.2009
Covers: 1.0.0.RC and 2.0.0.RC

1. Introduction	7
What is PrimeFaces?	7
2. Installation	8
2.1 Downloading	8
<i>2.1.1 Downloading manually</i>	8
<i>2.1.2 Downloading with Maven</i>	8
2.2 Dependencies	9
2.3 Setup Configuration	9
3. Component Suite	12
3.1 AccordionPanel	12
3.2 AjaxStatus	16
3.3 AutoComplete	19
3.4 BreadCrumb	25
3.5 Captcha	28
3.6 Calendar	31
3.7 Carousel	39
3.8 Charts	45
<i>3.8.1 Pie Chart</i>	45
<i>3.8.2 Line Chart</i>	48
<i>3.8.3 Column Chart</i>	51
<i>3.8.4 Stacked Column Chart</i>	53
<i>3.8.5 Bar Chart</i>	55
<i>3.8.6 StackedBar Chart</i>	57
<i>3.8.7 Chart Series</i>	59
<i>3.8.8 Skinning Charts</i>	60
<i>3.8.9 Real-Time Charts</i>	63

3.8.10 Interactive Charts	65
3.9 Collector	67
3.10 Color Picker	70
3.11 Column	74
3.12 CommandButton	75
3.13 CommandLink	80
3.14 ConfirmDialog	83
3.15 DataExporter	85
3.16 DataTable	88
3.17 Dialog	105
3.18 Drag&Drop	110
3.19 Dock	114
3.20 DockItem	116
3.21 Editor	117
3.22 Effect	121
3.23 FileDownload	124
3.24 FileUpload	126
3.25 GraphicImage	132
3.26 Growl	137
3.27 HotKey	139
3.28 IdleMonitor	142
3.29 ImageCropper	145
3.30 ImageSwitch	149
3.31 Inplace	151
3.32 InputMask	153

3.33 Keyboard	156
3.34 Layout	164
3.35 LayoutUnit	171
3.36 LightBox	173
3.37 LinkButton	178
3.38 Media	180
3.39 Menu	182
3.40 Menubar	186
3.41 MenuItem	190
3.42 Message	191
3.43 Messages	193
3.44 OutputPanel	195
3.45 Panel	197
3.46 Password Strength	201
3.47 PickList	206
3.48 Poll	211
3.49 Printer	214
3.50 Push	216
3.51 Rating	217
3.52 RemoteCommand	220
3.53 Resizable	222
3.54 Resource	225
3.55 Resources	226
3.56 Slider	227
3.57 Spinner	230

3.58 Submenu	234
3.59 Stack	235
3.60 StackItem	237
3.61 TabSlider	238
3.62 TabView	240
3.63 Terminal	245
3.64 Tooltip	248
3.65 Tree	252
3.66 UIAjax	259
3.67 Watermark	263
3.68 Wizard	265
4. TouchFaces	270
 4.1 Getting Started with TouchFaces	270
 4.2 Views	272
 4.3 Navigations	275
 4.4 Ajax Integration	277
 4.5 Sample Applications	278
 4.6 TouchFaces Components	279
4.6.1 Application	279
4.6.2 NavBarControl	280
4.6.3 RowGroup	281
4.6.4 RowItem	282
4.6.5 Switch	283
4.6.6 TableView	285
4.6.7 View	286

5. Partial Page Rendering	287
6. Ajax Push/Comet	289
6.1 Atmosphere	289
6.2 PrimeFaces Push	290
<i>6.2.1 Setup</i>	<i>290</i>
<i>6.2.2. CometContext</i>	<i>291</i>
<i>6.2.3 Push Component</i>	<i>291</i>
7. Compatibility	293
7.1 JSF Compatibility	293
7.2 Browser Compatibility	293
8. Javascript	294
9. Resource Management	295
10. Integration with Java EE	296
11. IDE Support	297
12. Portlets	298
13. Project Resources	300
14. FAQ	301

1. Introduction

What is PrimeFaces?

PrimeFaces is an open source component suite for Java Server Faces featuring 70+ Ajax powered rich set of JSF components. Additional TouchFaces module features a UI kit for developing mobile web applications. Main goal of PrimeFaces is to create the ultimate component suite for JSF.



- Rich set of components (HtmlEditor, Dialog, AutoComplete, Charts and more).
- Built-in Ajax with Lightweight Partial Page Rendering.
- Native Ajax Push/Comet support.
- Mobile UI kit to create mobile web applications for handheld devices with webkit based browsers.(iPhone, Palm, Android Phones, Nokia S60 and more)
- Compatible with other component libraries.
- Unobtrusive javascript.
- Extensive documentation.

PrimeFaces is backed by Prime Technology, a Turkish software development company specialized in Agile consulting and Enterprise Java. Prime Technology also contributes to the development of Apache MyFaces since 2006.

2. Installation

2.1 Downloading

PrimeFaces has a single jar called **primefaces-{version}.jar**. There are two ways to download this jar, you can either download from PrimeFaces homepage or if you are a maven user you can let maven to automatically get it for you

2.1.1 Downloading manually

Each release is also available for download at the downloads section of PrimeFaces project site.

<http://primefaces.prime.com.tr/en/downloads.html>

2.1.2 Downloading with Maven

Group id of the dependency is *org.primefaces* and artifact id is *primefaces*.

```
<dependency>
    <groupId>org.primefaces</groupId>
    <artifactId>primefaces</artifactId>
    <version>1.0.0.RC</version>
</dependency>
```

In addition to the configuration above you also need to add Prime Technology maven repository to the repository list so that maven can download it.

```
<repository>
    <id>prime-repo</id>
    <name>Prime Technology Maven Repository</name>
    <url>http://repository.prime.com.tr/</url>
    <layout>default</layout>
</repository>
```

2.2 Dependencies

UI components module only requires a JAVA 5+ runtime, a JSF 1.2+ implementation as mandatory dependencies. Other than these required dependencies, there're some optional libraries for certain features.

Dependency	Version *	Type	Used for
JSF runtime	1.2.x or 2.x	Required	Apache MyFaces or Sun Mojarra
iText	1.4.8	Optional	PDF export support for DataExporter component
apache poi	3.2-FINAL	Optional	Excel export support for DataExporter component
commons-fileupload	1.2.1	Optional	FileUpload
commons-io	1.4	Optional	FileUpload
atmosphere-runtime	0.5.0	Optional	Ajax Push
atmosphere-compat	0.5.0	Optional	Ajax Push

* Listed versions are tested and known to be working with PrimeFaces, other versions of these dependencies may also work but not tested.

2.3 Setup Configuration

Following setup is valid for both PrimeFaces 1.x and 2.x.

Resource Servlet

PrimeFaces UI module uses a servlet to stream bundled resources to the client.

```
<servlet>
    <servlet-name>Resource Servlet</servlet-name>
    <servlet-class>org.primefaces.resource.ResourceServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>Resource Servlet</servlet-name>
    <url-pattern>/primefaces_resource/*</url-pattern>
</servlet-mapping>
```

JSF 2.0 Case

When using PrimeFaces 2.x with Mojarra 2.0, you need to add the following context-param to web.xml.

```
<context-param>
    <param-name>com.sun.faces.allowTextChildren</param-name>
    <param-value>true</param-value>
</context-param>
```

Note that this is not required when using PrimeFaces 1.x with a JSF 1.2 implementation.

Resources Component

Resource component needs to be present on a page that has PrimeFaces components, this component outputs the link and script tags that are necessary for PrimeFaces components to work. The ideal place to put resources component would be the html head element.

```
<head>
    <p:resources />
</head>
```

We could have wrapped the output response with a servlet filter, parse the html, insert the link and script tags to the head element but this would be an expensive operation and effect the applications performance badly.

A tip regarding p:resources is to add this component to the facelets or jsp template once, so that it gets added to each page automatically using the application.

Taglib

If you're a facelets user, the xml namespace configuration would be;

```
xmlns:p="http://primefaces.prime.com.tr/ui"
```

If you're using jsp the taglib definition is;

```
<%@ taglib uri="http://primefaces.prime.com.tr/ui" prefix="p" %>
```

Try a component

That's all for the configuration, now you can test if the setup above works by adding a component to the page. A complete facelets example;

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:p="http://primefaces.prime.com.tr/ui">
<head>
    <p:resources />
</head>
<body>
    <p:editor />
</body>
</html>
```

If you're using jsp, make sure p:resources is nested inside <f:view />

3. Component Suite

3.1 AccordionPanel

AccordionPanel is a container component that renders it's children in separate tabs and displays a sliding animation when a tab is being collapsed or expanded.

Lionel Messi

Lionel Messi

Messi is an unusual player. He is highly creative, and has the skills to take on defenders with ease. He is a versatile left-footed player who can play either in the middle or on either wing, or even as a centre forward. Although he is quite short, he is so fast and physically strong that he can cope with larger opponents. He is incredibly powerful, and a specialist in such dead ball situations as corners, free kicks and penalties. Leo Messi is cool-headed and able to assume several responsibilities in times of need. He is a player who is destined to have a very successful career in football.

Zlatan Ibrahimovic

Thierry Henry

Info

Tag	accordionPanel
Tag Class	org.primefaces.component.accordionpanel.AccordionpanelTag
Component Class	org.primefaces.component.accordionpanel.Accordionpanel
Component Type	org.primefaces.component.AccordionPanel
Component Family	org.primefaces.component
Renderer Type	org.primefaces.component.AccordionPanelRenderer
Renderer Class	org.primefaces.component.accordionpanel.AccordionPanelRenderer

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
activeIndex	null	String	Index of the active tab, use a comma seperated list to specify multiple tabs.
multiple	FALSE	boolean	Allows having more than one active tab
speed	0.5	double	Speed of the toggle animation in seconds.
style	null	String	Style of the root html container element.
styleClass	null	String	Style class of the root html container element.
hover	FALSE	boolean	Enables toggling on hover.
hoverDelay	500	integer	Time to wait in terms of ms to toggle a tab when it is hovered.
widgetVar	null	String	Javascript variable name of the client side widget.

Getting started with Accordion Panel

Accordion panel consists of one or more tabs and each tab can group any other jsf components.

```
<p:accordionPanel>
    <p:tab title="First Tab Title">
        <h:outputText value= "Lorem"/>
        ...More content for first tab
    </p:tab>
    <p:tab title="Second Tab Title">
        <h:outputText value="Ipsum" />
    </p:tab>
    <p:tab title="Third Tab Title">
        any set of components...
    </p:tab>
    ... any number of tabs
</p:accordionPanel>
```

Multiple Selection

By default, only one tab can be active, this behavior can be configured to multiple selection to activate more than one tab.

```
<p:accordionPanel multiple="true">
    //..tabs
</p:accordionPanel>
```

Animation Speed

Toggling of the tabs are animated and animation speed can be configured via the speed attribute, speed is considered in milliseconds and defaults to 0.5 seconds. Following accordion will slide slower than default.

```
<p:accordionPanel speed="2">
    //..tabs
</p:accordionPanel>
```

Hovering

Toggling happens when a tab header is clicked, if you need panels to be toggled on hover enable hover setting. Also hoverDelay specifies the delay of toggle on hover. Following accordionPanel would wait for 200 ms to toggle on hover.

```
<p:accordionPanel hover="true" hoverDelay="200">
    //..tabs
</p:accordionPanel>
```

Skinning

AccordionPanel resides in a main div container, style and styleClass apply to this main element. Use these two attributes to set common properties like width, margin etc. Inside the main container accordion is represented as an unordered list() and each tab is located inside a element.

Following is the list of skinning selectors;

Class	Applies
.yui-accordionview	ul element containing each tab.
.yui-accordion-panel	Each tab element container.

Class	Applies
.yui-accordion-toggle	Header of a tab.
.yui-accordion-toggle active	Header of an active tab.
.yui-accordion-toggle:hover	Header of an hovered tab.
.yui-accordion-content	Tab contents.

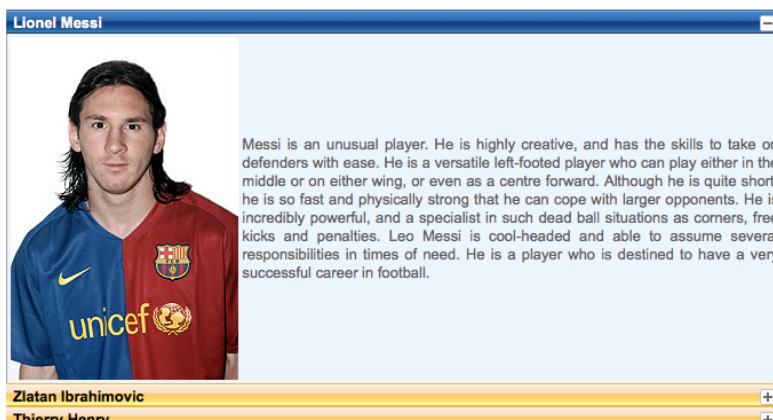
Here's an example of how to skin an accordionPanel using these css selectors.

```
.yui-skin-sam .yui-accordionview li.yui-accordion-panel a.yui-accordion-
toggle {
    height:12px;
    background-position: 0% 0%;
    border:0px;
}
.yui-skin-sam .yui-accordionview li.yui-accordion-panel a.yui-accordion-
toggle {
    background:url(..../images/hy.png) repeat-x;
}

.yui-skin-sam .yui-accordionview li.yui-accordion-panel a.yui-accordion-
toggle:hover {
    background:url(..../images/hy_hover.png) repeat-x;
}

.yui-skin-sam .yui-accordionview li.yui-accordion-panel a.yui-accordion-
toggle.active:hover, .yui-skin-sam .yui-accordionview li.yui-accordion-panel
a.yui-accordion-toggle.active {
    background:url(..../images/hy_active.png) repeat-x;
}
```

With these values accordionPanel will look like;



3.2 AjaxStatus

AjaxStatus is a global notifier to ajax requests made by PrimeFaces Partial Page Rendering components like button, poll, uiajax.



Info

Tag	ajaxStatus
Tag Class	org.primefaces.component.ajaxstatus.AjaxStatusTag
Component Class	org.primefaces.component.ajaxstatus.AjaxStatus
Component Type	org.primefaces.component.AjaxStatus
Component Family	org.primefaces.component
Renderer Type	org.primefaces.component.AjaxStatusRenderer
Renderer Class	org.primefaces.component.ajaxstatus.AjaxStatusRenderer

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component.
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
onstart	null	String	Javascript event handler to be executed after ajax requests start.
oncomplete	null	String	Javascript event handler to be executed after ajax requests complete.
onprestart	null	String	Javascript event handler to be executed before ajax requests start.
onsuccess	null	String	Javascript event handler to be executed after ajax requests is completed successfully.
onerror	null	String	Javascript event handler to be executed when an ajax request fails.

Name	Default	Type	Description
style	null	String	Style of the html element containing the facets.
styleClass	null	String	Style class of the html element containing the facets.
widgetVar	null	String	Javascript variable name of the client side widget.

Getting started with AjaxStatus

AjaxStatus uses facets to represent the ajax request status. Most common used facets are *start* and *complete*. Start facet will be visible once ajax request begins and stay visible until it's completed. Once the ajax response is received start facet becomes hidden and complete facet shows up.

```
<p:ajaxStatus>
    <f:facet name="start">
        <h:outputText value="Loading..." />
    </f:facet>

    <f:facet name="complete">
        <h:outputText value="Done!" />
    </f:facet>
</p:ajaxStatus>
```

More callbacks

Other than start and complete there're three more callback facets you can use. These are; prestart, success and error.

```
<p:ajaxStatus>
    <f:facet name="prestart">
        <h:outputText value="Starting..." />
    </f:facet>

    <f:facet name="error">
        <h:outputText value="Error!" />
    </f:facet>

    <f:facet name="success">
        <h:outputText value="Done!" />
    </f:facet>
</p:ajaxStatus>
```

Custom Events

If you want to execute custom javascript instead of the default usage with facets, use on* event handlers. These are the event handler versions of facets.

```
<p:ajaxStatus onstart="alert('Start')" oncomplete="alert('End')"/>
```

Animations

Generally, it's fancier to display animated gifs with ajax requests rather than plain texts.

```
<p:ajaxStatus>
    <f:facet name="start">
        <h:graphicImage value="ajaxloading.gif" />
    </f:facet>

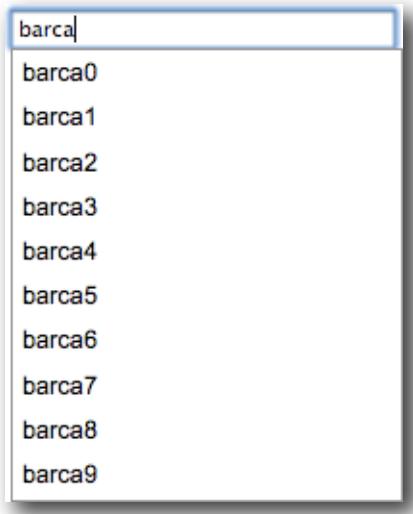
    <f:facet name="complete">
        <h:outputText value="Done!" />
    </f:facet>
</p:ajaxStatus>
```

Skinning AjaxStatus

AjaxStatus is equipped with style and styleClass. Styling directly applies to an html div element which contains the facets.

3.3 AutoComplete

AutoComplete is an ajax component that's used to provide suggestions while an input field is being typed. AutoComplete uses JSON to transfer the suggestions on the server back to the client.



Info

Tag	<code>autoComplete</code>
Tag Class	<code>org.primefaces.component.autocomplete.AutoCompleteTag</code>
Component Class	<code>org.primefaces.component.autocomplete.AutoComplete</code>
Component Type	<code>org.primefaces.component.AutoComplete</code>
Component Family	<code>org.primefaces.component</code>
Renderer Type	<code>org.primefaces.component.AutoCompleteRenderer</code>
Renderer Class	<code>org.primefaces.component.autocomplete.AutoCompleteRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component.
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean.

Name	Default	Type	Description
value	null	java.util.Date	Value of the component than can be either an EL expression or a literal text.
converter	null	Converter/ String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id.
immediate	FALSE	boolean	Boolean value that specifies the lifecycle phase the valueChangeEvents should be processed, when true the events will be fired at "apply request values", if immediate is set to false, valueChange Events are fired in "process validations" phase.
required	FALSE	boolean	Marks component as required.
validator	null	MethodBindi ng	A method binding expression that refers to a method validationong the input.
valueChangeListener	null	ValueChang eListener	A method binding expression that refers to a method for handling a valuchangeevent.
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.
widgetVar	null	String	Javascript variable name of the wrapped widget.
var	null	String	Name of the iterator.
itemLabel	null	String	Label of the item.
itemValue	null	String	Value of the item.
completeMethod	null	javax.el.Met hodExpressi on	Method to be called to fetch the suggestions.
animHoriz	FALSE	boolean	Specifies horizontal animation.
animVert	TRUE	boolean	Specifies vertical animation.
animSpeed	0.3	double	Speed of the animation in seconds, default value is 0.3 seconds.
maxResults	10	int	Maximum number of results to be displayed.

Name	Default	Type	Description
minQueryLength	1	int	Number of characters to be typed before starting to query.
queryDelay	0.2	double	Delay to wait in seconds before sending each query to the server.
autoHighlight	TRUE	boolean	When suggested items are listed, first item is highlighted automatically. This feature can be controlled with autoHighlight attribute.
useShadow	FALSE	boolean	Shadow is displayed under the results list when set to true.
typeAhead	FALSE	boolean	Updates the input field with the first query result.
typeAheadDelay	0.5	double	Delay before updating the input field with the first query result if typeAhead is set to true.

Getting started with AutoComplete

A method expression is called on the server side with the text entered as the query parameter. This method takes a string parameter.

```
public class Bean {

    private String text;

    public String getText() {
        return text;
    }
    public void setText(String text) {
        this.text = text;
    }

    public List<String> complete(String query) {
        List<String> results = new ArrayList<String>();

        for (int i = 0; i < 10; i++)
            results.add(query + i);

        return results;
    }
}
```

AutoComplete can use the complete method when querying the results. Also since autocomplete is an input component, the value attribute can be used to pass the text's value to the server side when the form is submitted.

```
<p:autoComplete value="#{bean.text}" completeMethod="#{bean.complete}" />
```

Pojo Support

Instead of simple strings, pojos are also supported.

```
public class Controller {

    private Player selectedPlayer;

    public Player getSelectedPlayer() {
        return selectedPlayer;
    }
    public void setSelectedPlayer(Player selectedPlayer) {
        this.selectedPlayer = selectedPlayer;
    }

    public List<Player> complete(String query) {
        //List<Player> players = readFromDB(query);

        return players;
    }
}
```

```
<p:autoComplete value="#{autoCompleteBean.selectedPlayer}"
    completeMethod="#{autoCompleteBean.completePlayer}"
    var="player" itemLabel="#{player.name}" itemValue="#{player}"
    converter="player"/>
```

With the same principle of a select component, itemLabel is the text to display as a suggestion and itemValue is the value to be submitted. You may also bind your converter to the autocomplete.

Limiting the results

Number of results shown can be limited, by default the limit is 10.

```
<p:autoComplete value="#{bean.text}" completeMethod="#{bean.complete}"
    maxResultsDisplayed="5" />
```

Minimum query length

By default queries are sent to the server and completeMethod is called as soon as users starts typing at the input text. This behavior is tuned using the *minQueryLength* attribute.

```
<p:autoComplete value="#{bean.text}" completeMethod="#{bean.complete}"
    minQueryLength="3" />
```

With this setting, querying will start when user types the 3rd character at the input field.

Animation

When results are returned from the server, a vertical animation is displayed by default. Orientation of the animation is configured via animVert and animHoriz attributes. In addition animSpeed determines the speed of the animation.

```
<p:autoComplete value="#{bean.text}" completeMethod="#{bean.complete}"
    animVert="false" animHoriz="true" animSpeed="1"/>
```

Skinning

AutoComplete is skinned using CSS selectors, an example would be;

```
.yui-skin-sam .yui-ac {
    width:200px;
}
.yui-skin-sam .yui-ac-content li {
    background:#FFFFCC;
    color:#33CC00;
}
.yui-skin-sam .yui-ac-content li.yui-ac-prehighlight {
    background:#B3D4FF;
}
.yui-skin-sam .yui-ac-content li.yui-ac-highlight {
    background:#CCFF66;color:#FFFFFF;
}
```

Output of these styles;



Full list of CSS Selectors;

http://developer.yahoo.com/yui/examples/autocomplete/ac_skinning.html

3.4 BreadCrumb

Breadcrumb is a handy navigation component that provides contextual information about page hierarchy in the workflow.



Info

Tag	breadCrumb
Tag Class	org.primefaces.component.breadcrumb.BreadCrumbTag
Component Class	org.primefaces.component.breadcrumb.BreadCrumb
Component Type	org.primefaces.component.BreadCrumb
Component Family	org.primefaces.component
Renderer Type	org.primefaces.component.BreadCrumbRenderer
Renderer Class	org.primefaces.component.breadcrumb.BreadCrumbRenderer

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component.
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
expandedEndItems	1	Integer	Number of expanded menuitems at the end.
expandedBeginningItems	1	Integer	Number of expanded menuitems at begining.
expandEffectDuration	800	Integer	Expanded effect duration in milliseconds.
collapseEffectDuration	500	Integer	Collapse effect duration in milliseconds.
initialCollapseEffectDuration	600	Integer	Initial collapse effect duration in milliseconds.

Name	Default	Type	Description
previewWidth	5	Integer	Preview width of a collapsed menuitem.
preview	FALSE	boolean	Specifies preview mode, when set to false menuitems will not collapse.
style	null	String	Style of main container element.
styleClass	null	String	Style class of main container element.

Getting Started with BreadCrumb

Steps are defined as child menuitem components in breadcrumb.

```
<p:breadcrumb>
    <p:menuitem label="Categories" url="#" />
    <p:menuitem label="Sports" url="#" />
    <p:menuitem label="Football" url="#" />
    <p:menuitem label="Countries" url="#" />
    <p:menuitem label="Spain" url="#" />
    <p:menuitem label="F.C. Barcelona" url="#" />
    <p:menuitem label="Squad" url="#" />
    <p:menuitem label="Lionel Messi" url="#" />
</p:breadcrumb>
```

Preview

By default all menuitems are expanded, if you have limited space and many menuitems, breadcrumb can collapse/expand menuitems on mouseover. Also previewWidth attribute defines the reveal amount in pixels.

```
<p:breadcrumb preview="true">
    <p:menuitem label="Categories" url="#" />
    <p:menuitem label="Sports" url="#" />
    <p:menuitem label="Football" url="#" />
    <p:menuitem label="Countries" url="#" />
    <p:menuitem label="Spain" url="#" />
    <p:menuitem label="F.C. Barcelona" url="#" />
    <p:menuitem label="Squad" url="#" />
    <p:menuitem label="Lionel Messi" url="#" />
</p:breadcrumb>
```



Animation Configuration

Duration of effects can be customized using several attributes. Here's an example;

```
<p:breadCrumb preview="true" expandEffectDuration="1000"
               collapseEffectDuration="1000"
               initialCollapseEffectDuration="1000">
    //menuitems
</p:breadCrumb>
```

Durations are defined in milliseconds.

Skinning BreadCrumb

Here's the list of pre defined breadcrumb style classes.

Style Class	Applies
.pf-breadCrumb	Main breadcrumb container element.
.pf-breadCrumb ul	Container list of each menuitem.
.pf-breadCrumb ul li	Each menuitem container.
.pf-breadCrumb ul li a	Link element of each menuitem.
.pf-breadCrumb ul li.first a	First element of breadcrumb.
.pf-breadCrumb ul li div.pf-breadCrumb-chevron	Separator of menuitems.

3.5 Captcha

Captcha is a form validation component used to make sure submitter of the form is a human, not a bot. Captcha is based on the generic recaptcha api.

Info

Tag	<code>captcha</code>
Tag Class	<code>org.primefaces.component.captcha.CaptchaTag</code>
Component Class	<code>org.primefaces.component.captcha.Captcha</code>
Component Type	<code>org.primefaces.component.Captcha</code>
Component Family	<code>org.primefaces.component</code>
Renderer Type	<code>org.primefaces.component.CaptchaRenderer</code>
Renderer Class	<code>org.primefaces.component.captcha.CaptchaRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component.
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean.
<code>value</code>	null	<code>java.util.Date</code>	Value of the component than can be either an EL expression of a literal text.
<code>converter</code>	null	Converter/ String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id.
<code>immediate</code>	FALSE	boolean	Boolean value that specifies the lifecycle phase the valueChangeEvents should be processed, when true the events will be fired at "apply request values", if immediate is set to false, valueChange Events are fired in "process validations" phase.
<code>required</code>	FALSE	boolean	Marks component as required.

Name	Default	Type	Description
validator	null	MethodBinding	A method binding expression that refers to a method validationg the input.
valueChangeListener	null	ValueChangeListene	A method binding expression that refers to a method for handling a valuchangeevent.
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.
publicKey	null	String	Public recaptcha key for a specific domain
theme	red	String	Theme of the captcha, valid values are "red", "white", "blackglass", "clean" and "custom"
language	en	String	Key of the supported languages, default is "en"

Getting Started with Captcha

Catptcha uses reCaptcha api as the underlying captcha mechanism and captcha has a built-in captcha validator that always checks the value entered with reCaptcha.

First thing to do is to sign up to reCaptcha and gain public&private keys. Once you have the keys for your domain, add your private key to web deployment descriptor as follows.

```
<context-param>
    <param-name>org.primefaces.component.captcha.PRIVATE_KEY</param-name>
    <param-value>YOUR_PRIVATE_KEY</param-value>
</context-param>
```

Once private key is installed, place the captcha component on your page as;

```
<p:captcha publicKey="YOUR_PUBLIC_KEY"/>
```

That's it, now invalid values entered to the captcha will result in validation errors.

Themes

Captcha supports several themes, note that custom styling is not yet supported. Following are the valid built-in themes.

- red (default)
- white
- blackglass
- clean

Themes are applied via the theme attribute.

```
<p:captcha publicKey="YOUR_PUBLIC_KEY" theme="white"/>
```

Languages

Text instructions displayed on captcha is customized with the language attribute. Below demonstrates a Turkish captcha.

```
<p:captcha publicKey="YOUR_PUBLIC_KEY" language="tr"/>
```

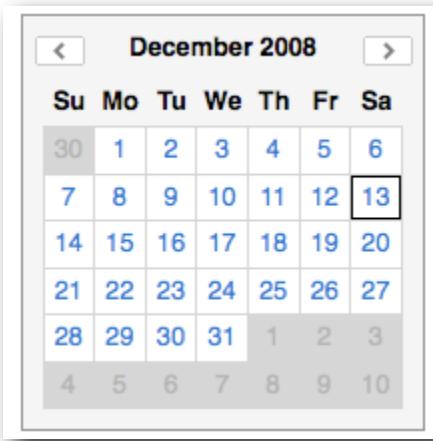
Overriding Validation Messages

By default captcha displays it's own validation messages, this can be easily overridden by the JSF message bundle mechanism. Corresponding keys are;

Summary	org.primefaces.component.captcha.CaptchaValidator.INVALID
Detail	org.primefaces.component.captcha.CaptchaValidator.INVALID_detail

3.6 Calendar

Calendar is an input component allowing to enter a date in various ways. Other than basic features calendar supports multiple date selection, paging, localization and more.



Info

Tag	calendar
Tag Class	org.primefaces.component.calendar.CalendarTag
Component Class	org.primefaces.component.calendar.Calendar
Component Type	org.primefaces.component.Calendar
Component Family	org.primefaces.component
Renderer Type	org.primefaces.component.CalendarRenderer
Renderer Class	org.primefaces.component.calendar.CalendarRenderer

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	java.util.Date or java.util.Date[]	Value of the component than can be either an EL expression of a literal text

Name	Default	Type	Description
converter	null	Converter/ String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id
immediate	FALSE	boolean	Boolean value that specifies the lifecycle phase the valueChangeEvents should be processed, when true the events will be fired at "apply request values", if immediate is set to false, valueChange Events are fired in "process validations" phase
required	FALSE	boolean	Marks component as required
validator	null	MethodBindin g	A method binding expression that refers to a method validationg the input
valueChangeListe ner	null	ValueChange Listener	A method binding expression that refers to a method for handling a valuchangeevent
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.
title	null	String	Title text at the top of calendar
mode	popup	String	inlinelpopup, Defines how the calendar will be displayed; "inline" only displays a calendar, "popup" displays an input text and a popup button
close	FALSE	boolean	Displays a close icon the top of calendar
mindate	null	Date or String	Sets calendar's minimum visible date
maxdate	null	Date or String	Sets calendar's maximum visible date
selection	single	String	Sets calendar's selection mode, "single" selects only one date, "multiple" can select multiple dates. Default value is "single"
pages	int	1	Enables multiple page rendering if more than 1
pattern	MM/dd/ yyyy	String	DateFormat pattern for localization
locale	null	java.util.Local e or String	Locale to be used for labels and conversion.

Name	Default	Type	Description
showWeekdays	TRUE	boolean	Determines rendering of weekday headers
monthFormat	long	String	Specifies display format of months, possible values are "short", "medium", "long"(default)
weekdayFormat	short	String	Determines rendering of weekday format, possible values are "1char", "short"(default), "medium", "long"
startWeekday	0	int	Specifies first day of week, by default it's 0 corresponding to sunday
popupIcon	null	String	Icon of the popup button
navigator	FALSE	boolean	Enables month/year navigator
pagedate	null	Date or String	Initial month and year shown on calendar in MM/yyyy format
timeZone	null	java.util.TimeZone	String or a java.util.TimeZone instance to specify the timezone used for date conversion, defaults to TimeZone.getDefault()
showWeekHeader	FALSE	boolean	Determines displaying week headers.
showWeekFooter	FALSE	boolean	Determines displaying week footer.
widgetVar	null	String	Javascript variable name of the wrapped widget

Getting started with Calendar

Calendar works with java.util.Date class. In simple selection a Date object needs to be bound as the value.

```
public class DateController {

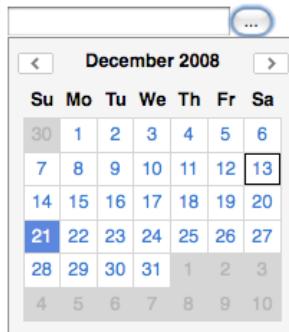
    private Date date;

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }
}
```

```
<p:calendar value="#{dateController.date}" />
```

Since default mode is popup, this calendar would render as;



Calendar modes

Calendar has two different rendering modes, “popup”(default) and “inline”. Popup mode displays and input text and a calendar button that pops up the calendar. Inline mode just displays the calendar without an inputtext.

Multiple Selection

Calendar also has support for multiple date selection, in this case the value should be a Date array instead of a single Date.

```
public class DateController {  
  
    private Date[] dates;  
  
    public Date[] getDates() {  
        return dates;  
    }  
  
    public void setDates(Date[] dates) {  
        this.dates = dates;  
    }  
}
```

```
<p:calendar value="#{dateController.date}" selection="multiple"/>
```

Date pattern

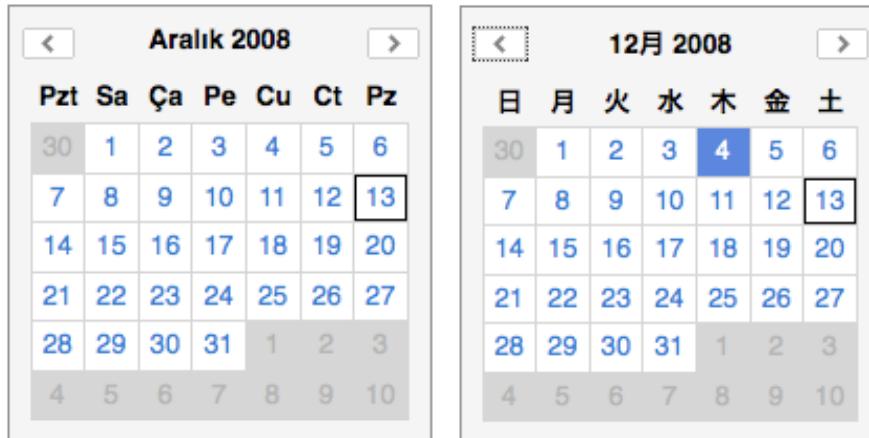
Calendar has a built-in converter so there's no need to define a datetimeconverter. Default pattern is “MM/dd/yyyy”. It's possible to change the default pattern for localization using the pattern attribute.

```
<p:calendar value="#{dateController.date}" pattern="dd.MM.yyyy"/>
```

Localization

By default locale information is retrieved from the view's locale and can be overridden by the locale attribute. Locale attribute can take a locale key as a String or a java.util.Locale instance.

```
<p:calendar value="#{dateController.date}" locale="tr"/>
```



Following languages are supported out of the box;

Language	Key
English	en
Turkish	tr
Catalan	ca
Portuguese	pt
Italian	it
French	fr
Spanish	es
German	de
Japanese	ja

More languages will be added in future releases, patches are welcome for more i18n support.

Paging

Calendar can also be rendered in multiple pages where each page corresponds to one month. This feature is tuned with the *pages* attribute.

```
<pf:calendar value="#{dateController.date}" pages="3"/>
```

December 2008							January 2009							February 2009						
Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
30	1	2	3	4	5	6	28	29	30	31	1	2	3	1	2	3	4	5	6	7
7	8	9	10	11	12	13	4	5	6	7	8	9	10	8	9	10	11	12	13	14
14	15	16	17	18	19	20	11	12	13	14	15	16	17	15	16	17	18	19	20	21
21	22	23	24	25	26	27	18	19	20	21	22	23	24	22	23	24	25	26	27	28
28	29	30	31	1	2	3	25	26	27	28	29	30	31	1	2	3	4	5	6	7
4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10	11	12	13	14

PageDate

PageDate defines the initial month and year visible on calendar. By default current day's month and year is displayed. This can be changed using the pagedate attribute which can be a java.util.Date or a String, note that format used is MM/yyyy. Following example displays all months in a year.

```
<p:calendar value="#{calendarBean.date}" mode="inline" pages="12"
pagedate="01/2009" showWeekHeader="true"/>
```

January 2009							February 2009							March 2009						
Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
1	28	29	30	31	1	2	3	6	1	2	3	4	5	6	7	10	1	2	3	4
2	4	5	6	7	8	9	10	7	8	9	10	11	12	13	14	11	8	9	10	11
3	11	12	13	14	15	16	17	8	15	16	17	18	19	20	21	12	15	16	17	18
4	18	19	20	21	22	23	24	9	22	23	24	25	26	27	28	13	22	23	24	25
5	25	26	27	28	29	30	31	10	1	2	3	4	5	6	7	14	29	30	31	1
6	1	2	3	4	5	6	7	11	8	9	10	11	12	13	14	15	5	6	7	8

April 2009							May 2009							June 2009						
Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
14	29	30	31	1	2	3	18	26	27	28	29	30	1	2	3	4	5	6	7	
15	5	6	7	8	9	10	11	19	3	4	5	6	7	8	9	24	7	8	9	10
16	12	13	14	15	16	17	18	20	10	11	12	13	14	15	16	25	14	15	16	17
17	19	20	21	22	23	24	25	21	17	18	19	20	21	22	23	26	21	22	23	24
18	26	27	28	29	30	1	2	22	24	25	26	27	28	29	30	27	28	29	30	1
19	3	4	5	6	7	8	9	23	31	1	2	3	4	5	6	28	5	6	7	8

July 2009							August 2009							September 2009						
Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
27	28	29	30	1	2	3	31	26	27	28	29	30	31	1	2	3	4	5	6	
28	5	6	7	8	9	10	11	32	2	3	4	5	6	7	8	37	6	7	8	9
29	12	13	14	15	16	17	18	33	9	10	11	12	13	14	15	38	13	14	15	16
30	19	20	21	22	23	24	25	34	16	17	18	19	20	21	22	39	20	21	22	23
31	26	27	28	29	30	31	1	35	23	24	25	26	27	28	29	40	27	28	29	30
32	2	3	4	5	6	7	8	36	30	31	1	2	3	4	5	41	4	5	6	7

October 2009							November 2009							December 2009						
Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
40	27	28	29	30	1	2	3	45	1	2	3	4	5	6	7	49	29	30	1	2
41	4	5	6	7	8	9	10	46	6	9	10	11	12	13	14	50	6	7	8	9
42	11	12	13	14	15	16	17	47	15	16	17	18	19	20	21	51	13	14	15	16
43	18	19	20	21	22	23	24	48	22	23	24	25	26	27	28	52	20	21	22	23
44	25	26	27	28	29	30	31	49	29	30	1	2	3	4	5	53	27	28	29	30
45	1	2	3	4	5	6	7	50	6	7	8	9	10	11	12	2	3	4	5	6

Start of week

Start of week is sunday by default(0 index), this can be configured via the startWeekday attribute.

```
<p:calendar value="#{dateController.date}" startWeekday="1"/>
```

This calendar will start the weeks from monday.

Label formats

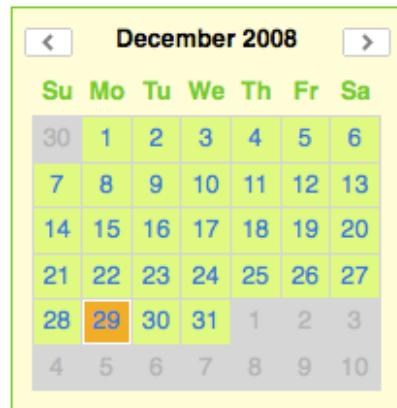
Month and weekday label supports various formats. By default full name of months and short name of weekdays are displayed. Formats can be configured by monthFormat and weekdayFormat attributes.

```
<p:calendar value="#{dateController.date}" monthFormat="medium"
weekdayFormat="1char"/>
```

Please see the attributes list description section for the all possible values for these fields.

Skinning calendar

```
.yui-skin-sam .yui-calcontainer {
    background-color: #FFFFCC;
    border:1px solid #33CC00;
}
.yui-skin-sam .yui-calendar .calweekdaycell {
    color: #33CC00;
}
.yui-skin-sam .yui-calendar td.calcell {
    background: #CCFF66;
}
.yui-skin-sam .yui-calendar td.calcell.today {
    background-color:#FFFFFF;
}
.yui-skin-sam .yui-calendar td.calcell.today a {
    background-color:#FF9900;
}
```

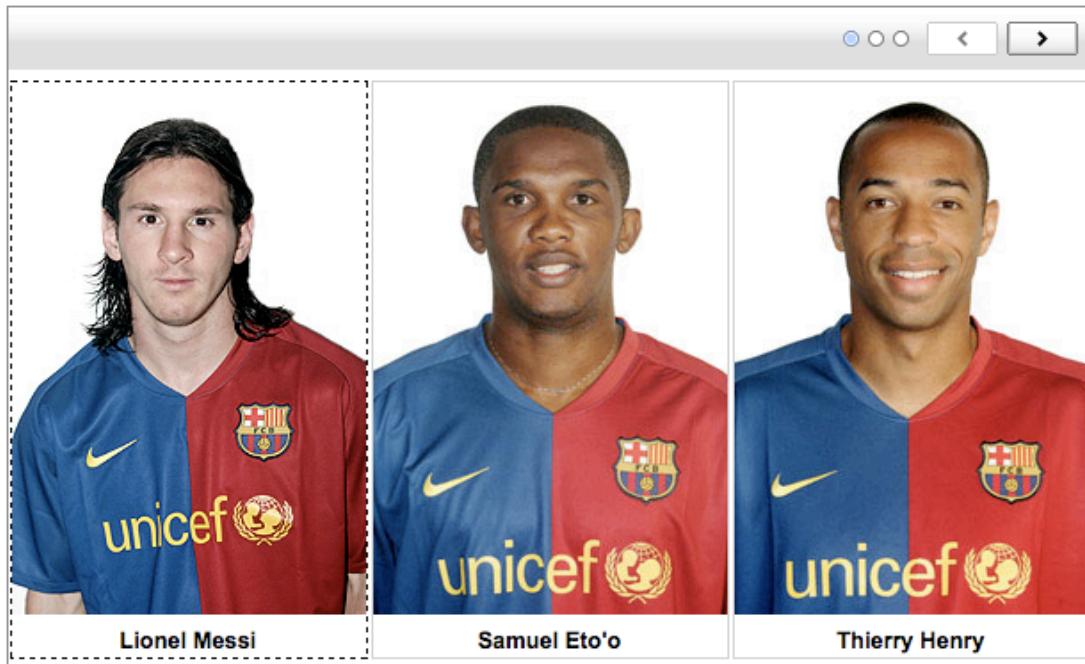


CSS Selectors List

<http://developer.yahoo.com/yui/examples/calendar/calskin.html>

3.7 Carousel

Carousel is a generic datalist component that displays its children in a slideshow style. Carousel gets a collection as the value, iterates the collection and renders the children for each item.



Info

Tag	<code>carousel</code>
Tag Class	<code>org.primefaces.component.carousel.CarouselTag</code>
Component Class	<code>org.primefaces.component.carousel.Carousel</code>
Component Type	<code>org.primefaces.component.Carousel</code>
Component Family	<code>org.primefaces.component</code>
Renderer Type	<code>org.primefaces.component.CarouselRenderer</code>
Renderer Class	<code>org.primefaces.component.carousel.CarouselRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component

Name	Default	Type	Description
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Collection	A value expression that refers to a collection instance to be listed
var	null	String	Name of the request scope based iterator name
rows	3	int	Number of visible items per page
first	0	int	Index of the first element to be displayed
selectedItem	0	int	Index of the selected item
scrollIncrement	1	int	Number of items to pass in each scroll
circular	FALSE	boolean	Sets continuous scrolling
vertical	FALSE	boolean	Sets vertical scrolling
autoPlayInterval	0	int	Sets the time in milliseconds to have Carousel start scrolling automatically after being initialized
reveralAmount	0	int	The percentage of the previous and next item of the current item to be revealed
animate	true	boolean	When enabled scrolling is animated, animation is turned on by default
speed	0.5	double	Sets the speed of the scrolling animation
effect	null	String	Name of the animation effect
widgetVar	null	string	Javascript variable name of the wrapped widget

Getting started with Carousel

To begin with, Carousel needs a java.util.Collection as it's value to iterate. Following example gets a players list and renders the player's photo along with player's name. Let's start with creating the player list to display with Carousel.

```

public class Player {

    private String name;
    private String photo;

    public Player() {}

    public Player(String name, String photo) {
        this.name = name;
        this.photo = photo;
    }

    //getters and setters
}

```

```

public class PlayerListController {

    private List<Player> players;

    public PlayerListController() {
        players = new ArrayList<Player>();
        players.add(new Player("Lionel Messi", "barca/messi.jpg"));
        players.add(new Player("Samuel Eto'o", "barca/etoo.jpg"));
        players.add(new Player("Thierry Henry", "barca/henry.jpg"));
        players.add(new Player("Xavi Hernandez", "barca/xavi.jpg"));
        players.add(new Player("Andres Iniesta", "barca/iniesta.jpg"));
        players.add(new Player("Carles Puyol", "barca/puyol.jpg"));
        players.add(new Player("Rafael Marquez", "barca/marquez.jpg"));
        players.add(new Player("Dani Alves", "barca/alves.jpg"));
        players.add(new Player("Victor Valdes", "barca/valdes.jpg"));
    }

    public List<Player> getPlayers() {
        return players;
    }

    public void setPlayers(List<Player> players) {
        this.players = players;
    }
}

```

```

<p:carousel value="#{carouselBean.players}" var="player">
    <h:graphicImage value="#{player.photo}" />
    <h:outputText value="#{player.name}" />
</p:carousel>

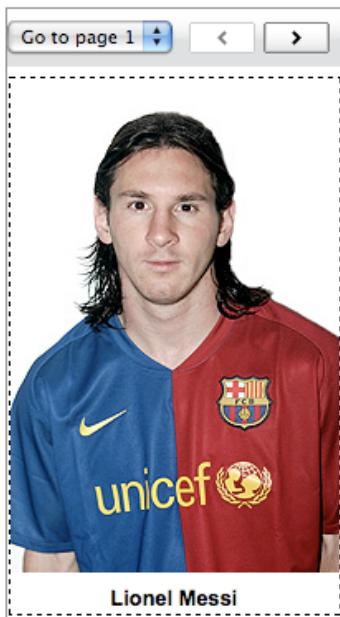
```

Similar to the datatable usage, carousel iterates through the players collection and renders it's children for each of the player.

Limiting Visible Items

By default carousel lists it's items in pages with size 3. This is customizable with the numVisible attribute.

```
<p:carousel value="#{carouselBean.players}" var="player"
             rows="1">
    <h:graphicImage value="#{player.photo}" />
    <h:outputText value="#{player.name}" />
</p:carousel>
```



Reveal Amount

Reveal amount is the percentage of the next and previous item to be shown, it can be tuned by the revealAmount attribute. Example above reveals %20 of the next and previous items.

```
<p:carousel value="#{carouselBean.players}" var="player"
             rows="1" revealAmount="20">
    <h:graphicImage value="#{player.photo}" />
    <h:outputText value="#{player.name}" />
</p:carousel>
```



Effects

By default paging happens with a slider effect, there are also more effects available for paging. Valid values are;

- backBoth
- backIn
- backOut
- bounceBoth
- bounceIn
- bounceOut
- easeBoth
- easeBothStrong
- easeIn
- easeInStrong
- easeNone
- easeOut
- easeOutStrong
- elasticBoth
- elasticIn
- elasticOut

Note: Effect names are case sensitive and incorrect usage may result in javascript errors

SlideShow

Carousel can display the contents in a slideshow as well, for this purpose autoPlayInterval and circular attributes are used.

Skinning

An example skinning is as follows;

```
.yui-skin-sam .yui-carousel-nav {
    background:transparent url(..../design/nav.gif) repeat scroll 0 0;
}

.yui-skin-sam .yui-carousel-content {
    background:#FFFF00 ;
}

.yui-skin-sam .yui-carousel-element {
    background:#FFFFCC ;
}

.yui-carousel .yui-carousel-item-selected {
    background:#33CC00 none repeat scroll 0 0;
    border:1px solid #33CC00;
}
```



CSS Selectors List

<http://developer.yahoo.com/yui/carousel/#skinning>

3.8 Charts

Charts are flash based JSF components to display graphical data. There're various chart types like pie, column, line and more. Charts can also display real-time data and also can fire server side events as response to user interaction.

3.8.1 Pie Chart

Pie chart displays category-data pairs in a pie graphic.

Info

Tag	<code>pieChart</code>
Tag Class	<code>org.primefaces.component.chart.pie.PieChartTag</code>
Component Class	<code>org.primefaces.component.chart.pie.PieChart</code>
Component Type	<code>org.primefaces.component.chart.PieChart</code>
Component Family	<code>org.primefaces.component</code>
Renderer Type	<code>org.primefaces.component.chart.PieChartRenderer</code>
Renderer Class	<code>org.primefaces.component.chart.pie.PieChartRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
<code>value</code>	null	<code>java.util.List</code>	Datasource to be displayed on the chart
<code>var</code>	null	String	Name of the data iterator
<code>categoryField</code>	null	Object	Pie category field
<code>dataField</code>	null	Object	Pie data field
<code>live</code>	FALSE	boolean	When a chart is live, the data is refreshed based on the refreshInterval period.
<code>refreshInterval</code>	3000	Integer	Refresh period of a live chart data in milliseconds

Name	Default	Type	Description
update	null	String	Client side id of the component(s) to be updated after async partial submit request
oncomplete	null	String	Javascript event to be called when ajax request for item select event is completed.
itemSelectListener	null	MethodExpression	Method expression to listen chart series item select events
styleClass	null	String	Style to apply to chart container element
style	null	String	Javascript variable name representing the styles
seriesStyle	null	String	Javascript variable name representing the series styles
width	500px	String	Width of the chart.
height	350px	String	Height of the chart.

Getting started with PieChart

Chart needs a collection like a java.util.List to display the data, in addition to the datasource categoryField is used to identify the pie section and dataField is used to hold the value of the corresponding categoryField. As an example, suppose there are 4 brands and each brand has made x amount of sales last year. We begin with creating the sale class to represent this model.

```
public class Sale {

    private String brand;
    private int amount;

    public Sale() {}

    public Sale(String brand, int amount) {
        this.brand = brand;
        this.amount = amount;
    }

    //getters and setters for brand and amount
}
```

In SaleDisplay bean, a java.util.List holds sale data of the 4 brands.

```

public class SaleDisplayBean {

    private List<Sale> sales;

    public SaleDisplayBean() {
        sales = new ArrayList<Sale>();
        sales.add(new Sale("Brand 1", 540));
        sales.add(new Sale("Brand 2", 325));
        sales.add(new Sale("Brand 3", 702));
        sales.add(new Sale("Brand 4", 421));
    }

    public List<Sale> getSales() {
        return sales;
    }
}

```

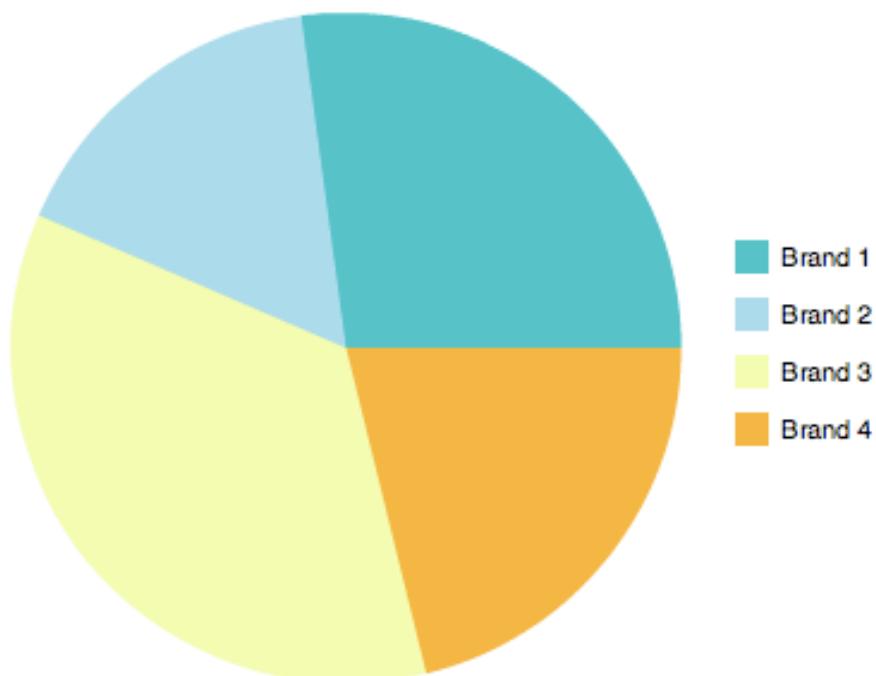
That's all the information needed for the pieChart to start working. Sales list can be visualized as follows;

```

<p:pieChart value="#{chartBean.sales}" var="sale"
            categoryField="#{sale.brand}" dataField="#{sale.amount}" />

```

Output would be;



3.8.2 Line Chart

Line chart visualizes one or more series of data in a line graph.

Info

Tag	<code>lineChart</code>
Tag Class	<code>org.primefaces.component.chart.line.LineChartTag</code>
Component Class	<code>org.primefaces.component.chart.line.LineChart</code>
Component Type	<code>org.primefaces.component.chart.LineChart</code>
Component Family	<code>org.primefaces.component</code>
Renderer Type	<code>org.primefaces.component.chart.LineChartRenderer</code>
Renderer Class	<code>org.primefaces.component.chart.line.LineChartRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
<code>value</code>	null	<code>java.util.List</code>	Datasource to be displayed on the chart
<code>var</code>	null	String	Name of the data iterator
<code>xField</code>	null	Object	Data of the x-axis
<code>live</code>	FALSE	boolean	When a chart is live, the data is refreshed based on the refreshInterval period.
<code>refreshInterval</code>	3000	Integer	Refresh period of a live chart data in milliseconds
<code>update</code>	null	String	Client side id of the component(s) to be updated after async partial submit request
<code>oncomplete</code>	null	String	Javascript event to be called when ajax request for item select event is completed.
<code>itemSelectListener</code>	null	MethodExpression	Method expression to listen chart series item select events

Name	Default	Type	Description
styleClass	null	String	Style to apply to chart container element
style	null	String	Javascript variable name representing the styles
minY	null	double	Minimum boundary value for y-axis.
maxY	null	double	Maximum boundary value for y-axis.
width	500px	String	Width of the chart.
height	350px	String	Height of the chart.

Getting started with LineChart

LineChart mainly needs a collection as the value, the xField data for the x-axis and one or more series data each corresponding to a line on the graph. To give an example, we'd display and compare the number of boys and girls year by year who was born last year at some place on earth. To model this, we'd need the Birth class.

```
public class Birth {

    private int year;
    private int boys;
    private int girls;

    public Birth() {}

    public Birth(int year, int boys, int girls) {
        this.year = year;
        this.boys = boys;
        this.girls = girls;
    }
    //getters and setters for fields
}
```

Next thing to do is to prepare the data year by year in BirthDisplayBean.

```

public class BirthDisplayBean {

    private List<Birth> births;

    public ChartBean() {
        births = new ArrayList<Birth>();
        births.add(new Birth(2004, 120, 52));
        births.add(new Birth(2005, 100, 60));
        births.add(new Birth(2006, 44, 110));
        births.add(new Birth(2007, 150, 135));
        births.add(new Birth(2008, 125, 120));
    }

    public List<Birth> getBirths() {
        return births;
    }
}

```

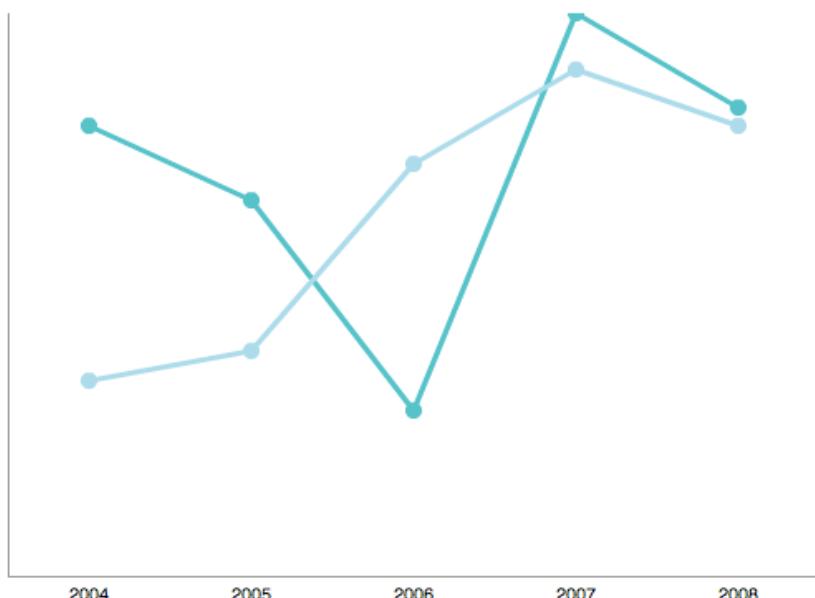
Given this birth p:chartSeriescollection, a linechart can visualize this data as follows;

```

<p:lineChart value="#{chartBean.births}" var="birth" xfield="#{birth.year}">
    <p:chartSeries label="Boys" value="#{birth.boys}" />
    <p:chartSeries label="Girls" value="#{birth.girls}" />
</p:lineChart>

```

Output of this lineChart would be;



3.8.3 Column Chart

Column chart visualizes one or more series of data using a column graph.

Info

Tag	<code>columnChart</code>
Tag Class	<code>org.primefaces.component.chart.column.ColumnChartTag</code>
Component Class	<code>org.primefaces.component.chart.column.ColumnChart</code>
Component Type	<code>org.primefaces.component.chart.ColumnChart</code>
Component Family	<code>org.primefaces.component</code>
Renderer Type	<code>org.primefaces.component.chart.ColumnChartRenderer</code>
Renderer Class	<code>org.primefaces.component.chart.column.ColumnChartRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
<code>value</code>	null	<code>java.util.List</code>	Datasource to be displayed on the chart
<code>var</code>	null	String	Name of the data iterator
<code>xField</code>	null	Object	Data of the x-axis
<code>live</code>	FALSE	boolean	When a chart is live, the data is refreshed based on the refreshInterval period.
<code>refreshInterval</code>	3000	Integer	Refresh period of a live chart data in milliseconds
<code>update</code>	null	String	Client side id of the component(s) to be updated after async partial submit request
<code>oncomplete</code>	null	String	Javascript event to be called when ajax request for item select event is completed.
<code>itemSelectListener</code>	null	MethodExpression	Method expression to listen chart series item select events

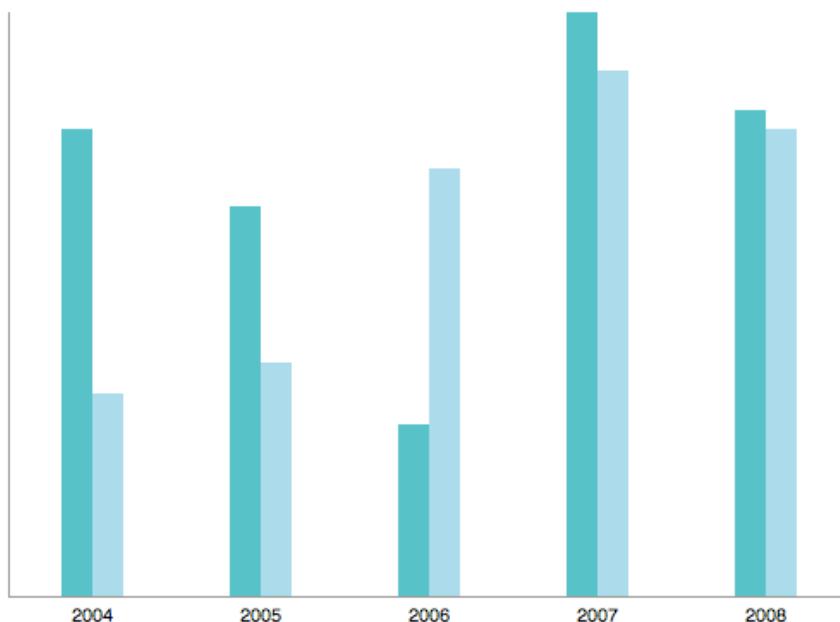
Name	Default	Type	Description
styleClass	null	String	Style to apply to chart container element
style	null	String	Javascript variable name representing the styles
minY	null	double	Minimum boundary value for y-axis.
maxY	null	double	Maximum boundary value for y-axis.
width	500px	String	Width of the chart.
height	350px	String	Height of the chart.

Getting started with Column Chart

Column chart usage is very similar to line chart, as an example following column chart displays the birth rate data in the lineChart example. Please see the lineChart section to get more information about the structure of the birth data.

```
<p:columnChart value="#{birthDisplayBean.births}" var="birth" xfield="#{birth.year}">
    <p:chartSeries label="Boys" value="#{birth.boys}" />
    <p:chartSeries label="Girls" value="#{birth.girls}" />
</p:lineChart>
```

Output of this column chart is;



3.8.4 Stacked Column Chart

Stacked Column chart is similar to column chart but the columns are stacked per each xField data.

Info

Tag	<code>stackedColumnChart</code>
Tag Class	<code>org.primefaces.component.chart.stackedcolumn.StackedColumnChartTag</code>
Component Class	<code>org.primefaces.component.chart.stackedcolumn.StackedColumnChart</code>
Component Type	<code>org.primefaces.component.chart.StackedColumnChart</code>
Component Family	<code>org.primefaces.component</code>
Renderer Type	<code>org.primefaces.component.chart.StackedColumnChartRenderer</code>
Renderer Class	<code>org.primefaces.component.chart.stackedcolumn.StackedColumnChartRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
<code>value</code>	null	<code>java.util.List</code>	Datasource to be displayed on the chart
<code>var</code>	null	String	Name of the data iterator
<code>xField</code>	null	Object	Data of the x-axis
<code>live</code>	FALSE	boolean	When a chart is live, the data is refreshed based on the refreshInterval period.
<code>refreshInterval</code>	3000	Integer	Refresh period of a live chart data in milliseconds
<code>update</code>	null	String	Client side id of the component(s) to be updated after async partial submit request
<code>oncomplete</code>	null	String	Javascript event to be called when ajax request for item select event is completed.

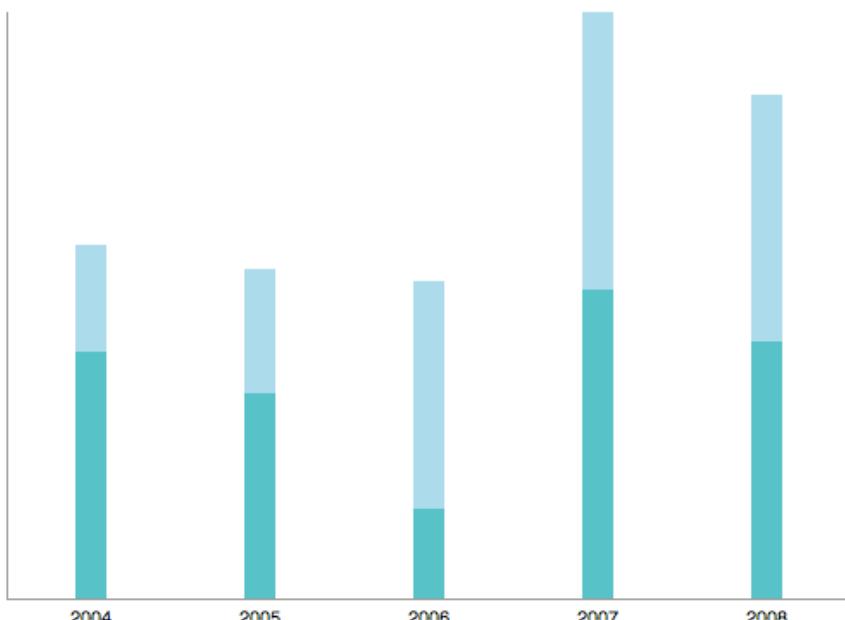
Name	Default	Type	Description
itemSelectListener	null	MethodExpression	Method expression to listen chart series item select events
styleClass	null	String	Style to apply to chart container element
style	null	String	Javascript variable name representing the styles
minY	null	double	Minimum boundary value for y-axis.
maxY	null	double	Maximum boundary value for y-axis.
width	500px	String	Width of the chart.
height	350px	String	Height of the chart.

Getting started with Stacked Column Chart

Stacked column chart usage is very similar to line chart, as an example following stacked column chart displays the birth rate data in the lineChart example. Please see the lineChart section to get more information about the structure of the birth data.

```
<p:stackedColumnChart value="#{birthDisplayBean.births}" var="birth"
    xfield="#{birth.month}"
    <p:chartSeries label="Boys" value="#{birth.boys}" />
    <p:chartSeries label="Girls" value="#{birth.girls}" />
</p:stackedColumnChart>
```

Output of this column chart is;



3.8.5 Bar Chart

Bar Chart is the horizontal version of the column chart where columns are aligned on x axis as bars.

Info

Tag	barChart
Tag Class	org.primefaces.component.chart.bar.BarChartTag
Component Class	org.primefaces.component.chart.bar.BarChart
Component Type	org.primefaces.component.chart.BarChart
Component Family	org.primefaces.component
Renderer Type	org.primefaces.component.chart.BarChartRenderer
Renderer Class	org.primefaces.component.chart.bar.BarChartRenderer

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	java.util.List	Datasource to be displayed on the chart
var	null	String	Name of the data iterator
yField	null	Object	Data of the y-axis
live	FALSE	boolean	When a chart is live, the data is refreshed based on the refreshInterval period.
refreshInterval	3000	Integer	Refresh period of a live chart data in milliseconds
update	null	String	Client side id of the component(s) to be updated after async partial submit request
oncomplete	null	String	Javascript event to be called when ajax request for item select event is completed.

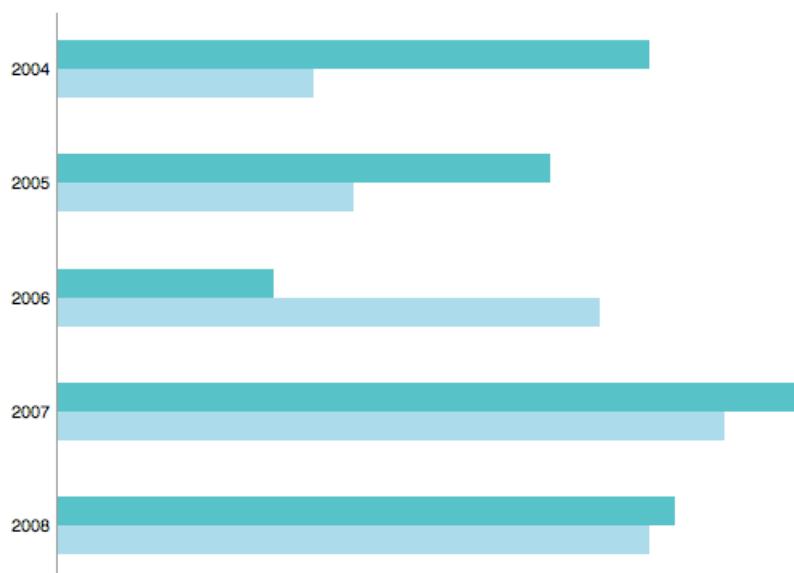
Name	Default	Type	Description
itemSelectListener	null	MethodExpression	Method expression to listen chart series item select events
styleClass	null	String	Style to apply to chart container element
style	null	String	Javascript variable name representing the styles
minX	null	double	Minimum boundary value for x-axis.
maxX	null	double	Maximum boundary value for x-axis.
width	500px	String	Width of the chart.
height	350px	String	Height of the chart.

Getting started with Bar Chart

Bar chart usage is very similar to line chart, as an example following bar chart displays the birth rate data in the lineChart example. Important difference is that barchart uses yfield attribute instead of the xfield attribute. Please see the lineChart section to get more information about the structure of the birth data.

```
<p:barChart value="#{birthDisplayBean.births}" var="birth" yfield="#{birth.month}">
    <p:chartSeries label="Boys" value="#{birth.boys}" />
    <p:chartSeries label="Girls" value="#{birth.girls}" />
</p:barChart>
```

Output of this bar chart is;



3.8.6 StackedBar Chart

Stacked Bar chart is similar to bar chart but the bar are stacked per each yField data.

Info

Tag	stackedBarChart
Tag Class	org.primefaces.component.chart.stackedbar.StackedBarChartTag
Component Class	org.primefaces.component.chart.stackedbar.StackedBarChart
Component Type	org.primefaces.component.chart.StackedBarChart
Component Family	org.primefaces.component
Renderer Type	org.primefaces.component.chart.StackedBarChartRenderer
Renderer Class	org.primefaces.component.chart.stackedbar.StackedBarChartRenderer

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	java.util.List	Datasource to be displayed on the chart
var	null	String	Name of the data iterator
yField	null	Object	Data of the y-axis
live	FALSE	boolean	When a chart is live, the data is refreshed based on the refreshInterval period.
refreshInterval	3000	Integer	Refresh period of a live chart data in milliseconds
update	null	String	Client side id of the component(s) to be updated after async partial submit request
oncomplete	null	String	Javascript event to be called when ajax request for item select event is completed.

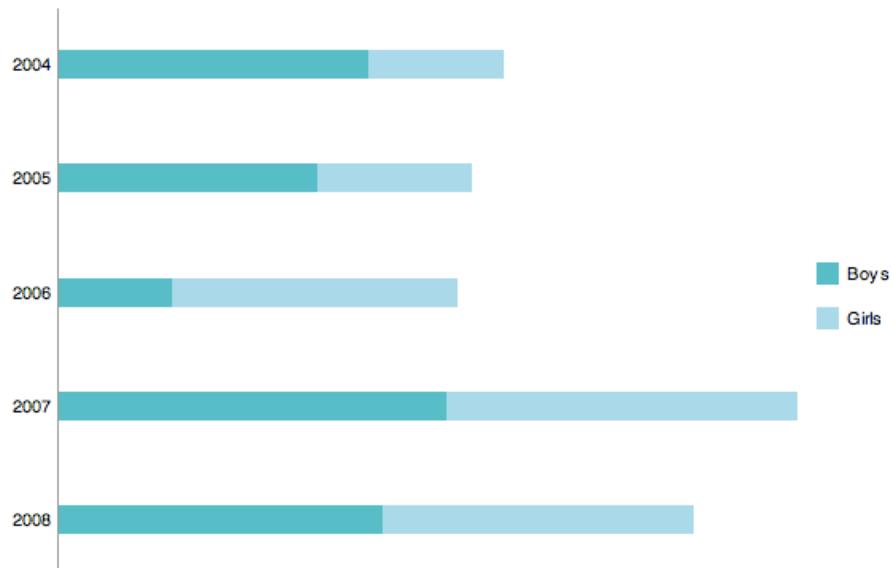
Name	Default	Type	Description
itemSelectListener	null	MethodExpression	Method expression to listen chart series item select events
styleClass	null	String	Style to apply to chart container element
style	null	String	Javascript variable name representing the styles
minX	null	double	Minimum boundary value for x-axis.
maxX	null	double	Maximum boundary value for x-axis.
width	500px	String	Width of the chart.
height	350px	String	Height of the chart.

Getting started with StackedBar Chart

StackedBar chart usage is very similar to line chart, as an example following stacked bar chart displays the birth rate data in the lineChart example. Important difference is that stackedbarchart uses yfield attribute instead of the xfield attribute. Please see the lineChart section to get more information about the structure of the birth data.

```
<p:stackedBarChart value="#{birthDisplayBean.births}" var="birth"
    yfield="#{birth.month}">
    <p:chartSeries label="Boys" value="#{birth.boys}" />
    <p:chartSeries label="Girls" value="#{birth.girls}" />
</p:stackedBarChart>
```

Output of this stacked bar chart is;



3.8.7 Chart Series

A chart can have one or more series and a chartSeries component represents each series in a chart.

Info

Tag	chartSeries
Tag Class	org.primefaces.component.chart.series.ChartSeriesTag
Component Class	org.primefaces.component.chart.series.ChartSeries
Component Type	org.primefaces.component.ChartSeries
Component Family	org.primefaces.component

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Value to be displayed on the series
converter	null	Converter	Output converter to be used if any.
label	null	java.lang.String	Label of the series
style	null	String	Javascript variable name representing the styles

Getting started with ChartSeries

ChartSeries is nested inside a chart component, you can have as many series as you want on a chart by nesting multiple series. Please see the other chart component documentations to see the usage of chartSeries.

3.8.8 Skinning Charts

Charts are highly customizable in terms of skinning however they are flash based, as a result regular CSS styling is not possible. Charts are styled through Javascript and the object is passed to the chart's style attribute.

There are two attributes in chart components related to skinning.

styleClass : Each chart resides in an html div element, style class applies to this container element. Style class is mainly useful for setting the width and height of the chart.

```
<style type="text/css">
    .chartClass {
        width:700px;
        height:400px;
    }
</style>
```

style : Style should be the javascript object variable name used in styling, as a simple example to start with; Style below effects chart padding, border and legend. See the full list of style selectors link for the complete list of selectors.

```
var chartStyle = {
    padding : 20,
    border: {color: 0x96acb4, size: 8},
    legend: {
        display: "right"
    }
};
```

Skinning Series

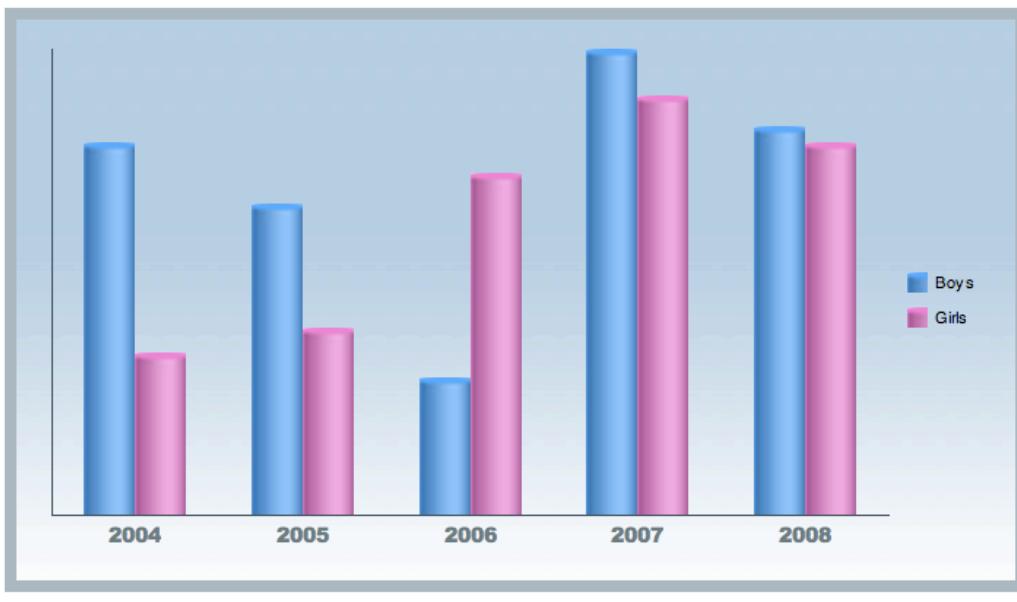
ChartSeries can be styled individually using the style attribute. Styling is same as charts and done via javascript.

```
var boysStyle = {
    color: 0x3399FF,
    size: 35
};
```

```
<p:chartSeries value="#{birth.boys}" label="Boys" style="boysStyle" />
```

Extreme Makeover

To give a complete styling example, we'll skin the chart described in colum chart section. In the end, after the extreme makeover chart will look like;



```

<style type="text/css">
    .chartClass {
        width:700px;
        height:400px;
    }
</style>

<script type="text/javascript">
    var chartStyle = {
        border: {color: 0x96acb4, size: 12},
        background: {
            image : "../design/bg.jpg"
        },
        font: {name: "Arial Black", size: 14, color: 0x586b71},
        dataTip:
        {
            border: {color: 0x2e434d, size: 2},
            font: {name: "Arial Black", size: 13, color: 0x586b71}
        },
        xAxis:
        {
            color: 0x2e434d
        },
        yAxis:
        {
            color: 0x2e434d,
            majorTicks: {color: 0x2e434d, length: 4},
            minorTicks: {color: 0x2e434d, length: 2},
            majorGridLines: {size: 0}
        }
    };
</script>

```

```
var boysSeriesStyle =  
{  
    image: "../design/column.png",  
    mode: "no-repeat",  
    color: 0x3399FF,  
    size: 35  
};  
  
var girlsSeriesStyle =  
{  
    image: "../design/column.png",  
    mode: "no-repeat",  
    color: 0xFF66CC,  
    size: 35  
};
```

```
<p:columnChart value="#{chartBean.births}" var="birth" xfield="#{birth.year}"  
styleClass="column" style="chartStyle">  
    <p:chartSeries label="Boys" value="#{birth.boys}" style="boysSeriesStyle"/>  
    <p:chartSeries label="Girls" value="#{birth.girls}" style="girlsSeriesStyle"/>  
</p:columnChart>
```

Full List of Style Selectors

<http://developer.yahoo.com/yui/charts/#basicstyles>

3.8.9 Real-Time Charts

Charts have built-in support for ajax polling and live data display. As an example suppose there's an ongoing vote between two candidates. To start with, create the Vote class representing the voting model.

```
public class Vote {  
  
    private String candidate;  
  
    private int count;  
  
    public Vote() {  
        //NoOp  
    }  
  
    public Vote(String candidate, int count) {  
        this.candidate = candidate;  
        this.count = count;  
    }  
  
    public String getCandidate() {  
        return candidate;  
    }  
  
    public void setCandidate(String candidate) {  
        this.candidate = candidate;  
    }  
  
    public int getCount() {  
        return count;  
    }  
  
    public void setCount(int count) {  
        this.count = count;  
    }  
  
    public void add(int count) {  
        this.count = this.count + count;  
    }  
}
```

Next step is to provide the data;

```

public class ChartBean implements Serializable {

    private List<Vote> votes;

    public ChartBean() {
        votes = new ArrayList<Vote>();
        votes.add(new Vote("Candidate 1", 100));
        votes.add(new Vote("Candidate 2", 100));
    }

    public List<Vote> getVotes() {
        int random1 = (int)(Math.random() * 1000);
        int random2 = (int)(Math.random() * 1000);

        votes.get(0).add(random1);
        votes.get(1).add(random2);

        return votes;
    }
}

```

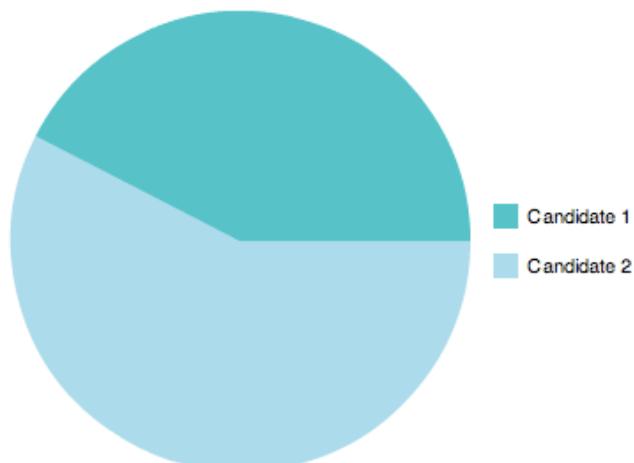
For displaying the voting, we'll be using a pie chart as follows;

```

<p:pieChart id="votes" value="#{chartBean.votes}" var="vote"
            live="true" refreshInterval="5000"
            categoryField="#{vote(candidate)}"
            dataField="#{vote.count}" />

```

This live piechart is almost same as a static pie chart, except live attribute is set to true. When a chart is live, the collection bind to the value is read periodically in a specified interval. In this example, getVotes() would be called continuously in 5 seconds interval. Polling interval is tuned using the refreshInterval attribute which is set to 3000 milliseconds.



3.8.10 Interactive Charts

Charts are interactive components and they can respond to events like series item selection. When a series item is clicked an ajax request is sent to the server and an itemSelectListener is notified passing an itemSelectEvent. ItemSelectEvent contains useful information about the selected item like series index and item index.

Chart components also use PrimeFaces Partial Page Rendering mechanism so using the update attribute, it's possible to refresh other components on the page. In the example below, message outputText is refreshed with the message provided in itemSelectListener.

```
<p:pieChart id="votes" value="#{chartBean.votes}" var="vote"
            itemSelectListener="#{chartBean.itemSelect}"
            update="msg"
            categoryField="#{vote.candidate}"
            dataField="#{vote.count}" />

<h:outputText value="#{chartBean.message}" />
```

```
public class ChartBean implements Serializable {

    //Data creation omitted

    public void itemSelect(ItemSelectEvent event) {
        message = "Item Index: " + event.getItemIndex() + ", Series Index:" +
        event.getSeriesIndex();
    }

}
```

Please note that interactive charts must be nested inside a form.

3.8.11 Charting FAQ

Flash Version

Chart components require flash player version 9.0.45 or higher.

Express Install

In case the users of your application use an older unsupported version of flash player, chart components will automatically prompt to install or update users' flash players. The screen would look like this for these users.



JFreeChart Integration

If you like to use static image charts instead of flash based charts, see the JFreeChart integration example at p:graphicImage section.

3.9 Collector

Collector is a simple utility component to manage collections without writing java code on backing beans.

Info

Tag	collector
Tag Class	org.primefaces.component.collector.CollectorTag
ActionListener Class	org.primefaces.component.collector.Collector

Attributes

Name	Default	Type	Description
value	null	Object	Value to be used in collection operation
addTo	null	java.util.Collection	Reference to the Collection instance
removeFrom	null	java.util.Collection	Reference to the Collection instance

Getting started with Collector

Collector requires a collection and a value to work with. It's important to override equals and hashCode methods of the value object to make collector work.

```
public class CreateBookBean {

    private Book book = new Book();

    private List<Book> books;

    public CreateBookBean() {
        books = new ArrayList<Book>();
    }

    public String createNew() {
        book = new Book();      //reset form

        return null;
    }

    //getters and setters
}
```

Value attribute is required and sets the object to be added or removed to/from a collection.

```
<p:commandButton value="Add" action="#{createBookBean.createNew}">
    <p:collector value="#{createBookBean.book}" addTo="#"
    {createBookBean.books}" />
</p:commandButton>
```

```
<p:commandLink value="Remove">
    <p value="#{book}" removeFrom="#"
    {createBookBean.books}" />
</p:commandLink>
```

Following is the complete example demonstrating both uses cases.

```

<h:form>
    <h:messages />
    <h:panelGrid columns="2" style="border:1px solid;" width="300px"
headerClass="formHeader">
        <f:facet name="header"><h:outputText value="Create a New Book" />
        </f:facet>

        <h:outputLabel value="Title : *" for="txt_title"></h:outputLabel>
        <h:inputText id="txt_title" value="#{createBookBean.book.title}"
required="true"/>

        <h:outputLabel value="Author : *" for="txt_author"></h:outputLabel>
        <h:inputText id="txt_author" value="#{createBookBean.book.author}"
required="true"/>

        <h:outputText value="" />

        <h:panelGroup>
            <h:commandButton value="Add" action="#{createBookBean.createNew}">
                <p:collector value="#{createBookBean.book}" addTo="#
{createBookBean.books}" />
            </h:commandButton>

            <h:commandButton value="Reset" type="reset"/>
        </h:panelGroup>
    </h:panelGrid>
</h:form>

<h:form>
    <p:dataTable value="#{createBookBean.books}" var="book">

        <p:column>
            <f:facet name="header"><h:outputText value="Title" /></f:facet>
            <h:outputText value="#{book.title}" />
        </p:column>

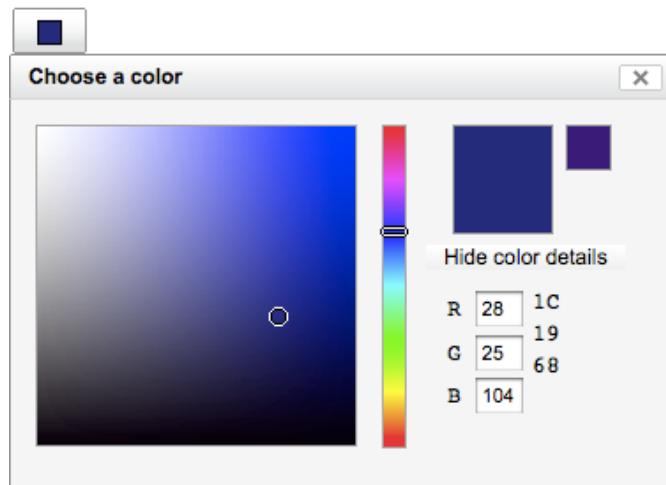
        <p:column>
            <f:facet name="header">
                <h:outputText value="Author" />
            </f:facet>
            <h:outputText value="#{book.author}" />
        </p:column>

        <p:column>
            <f:facet name="header"><h:outputText value="Operation" /></f:facet>
            <h:commandLink value="Remove">
                <p:collector value="#{book}" removeFrom="#
{createBookBean.books}" />
            </h:commandLink>
        </p:column>
    </p:dataTable>
</h:form>

```

3.10 Color Picker

ColorPicker component enhances color selection with color visualization.



Info

Tag	<code>colorPicker</code>
Tag Class	<code>org.primefaces.component.colorpicker.ColorPickerTag</code>
Component Class	<code>org.primefaces.component.colorpicker.ColorPicker</code>
Component Type	<code>org.primefaces.component.ColorPicker</code>
Component Family	<code>org.primefaces.component</code>
Renderer Type	<code>org.primefaces.component.ColorPickerRenderer</code>
Renderer Class	<code>org.primefaces.component.colorpicker.ColorPickerRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean

Name	Default	Type	Description
value	null	java.util.Date	Value of the component than can be either an EL expression or a literal text
converter	null	Converter/String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id
immediate	FALSE	boolean	Boolean value that specifies the lifecycle phase the valueChangeEvents should be processed, when true the events will be fired at "apply request values", if immediate is set to false, valueChange Events are fired in "process validations" phase
required	FALSE	boolean	Marks component as required.
validator	null	MethodExpression	A method expression that refers to a method for validation the input.
valueChangeListener	null	ValueChangeListener	A method binding expression that refers to a method for handling a valuchangeevent.
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.
widgetVar	null	String	Javascript variable name of the wrapped widget.
header	Choose a color	String	Header text for the color picker title.
modal	FALSE	boolean	Modality of the colorpicker dialog.
showControls	TRUE	String	Sets visibility of whole set of controls.
showHexControls	TRUE	String	Sets visibility of hex controls.
showHexSummary	TRUE	String	Sets visibility of hex summary.
showHsvControls	FALSE	String	Sets visibility of hsv controls.

Name	Default	Type	Description
showRGBControls	TRUE	String	Sets visibility of rgb controls.
showWebSafe	TRUE	String	Sets visibility of web safe controls.

Getting started with ColorPicker

ColorPicker component just needs a value to work. By default this value should be in type of java.awt.Color class. When user selects a color from the popup, the selected color's corresponding red,green,blue (RGB) values are populated to that object.

```
import java.awt.Color;

public class ColorPickerController {

    private Color selectedColor;

    public Color getSelectedColor(){
        return selectedColor;
    }
    public void setSelectedColor(Color color){
        selectedColor = color;
    }
}
```

```
<p:colorPicker value="#{colorBean.color}" />
```

Headless ColorPicker

For headless servers that have no java.awt.* you can still use colorPicker by the help of a custom converter. Here's an example.

```
public class ColorPickerConverter implements Converter {

    public Object getAsObject(FacesContext facesContext, UIComponent
            component, String submittedValue) {
        return submittedValue; //just return the rgb value as string
    }

    public String getAsString(FacesContext facesContext, UIComponent
            component, Object value) {
        return value == null ? null : value.toString();
    }
}
```

```
public class ColorPickerController {  
  
    private String selectedColor;  
  
    public String getSelectedColor(){  
        return selectedColor;  
    }  
    public void setSelectedColor(String color){  
        selectedColor = color;  
    }  
}
```

```
<p:colorPicker value="#{colorBean.color}">  
    <f:converter converterId="colorPickerConverer" />  
</p:colorPicker>
```

This way selected color will not be converted to java.awt.Color but used as a simple rgb string such as '250, 214, 255'.

3.11 Column

Column is an extended version of the standard column providing features like sorting, selection, resizing, filtering and more.

Info

Tag	<code>column</code>
Tag Class	<code>org.primefaces.component.column.ColumnTag</code>
Component Class	<code>org.primefaces.component.column.Column</code>
Component Type	<code>org.primefaces.component.Column</code>
Component Family	<code>org.primefaces.component</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
<code>sortBy</code>	null	Object	Property to be used when sorting this column.
<code>resizable</code>	FALSE	boolean	Boolean value to make the column width resizable
<code>filter</code>	FALSE	boolean	Specifies the data filter.
<code>selectionMode</code>	null	String	Defines this column as a selection column.
<code>parser</code>	null	String	Client side column parser.
<code>width</code>	null	Integer	Width in pixels.

Getting started with Column

Please see DataTable section to find out how column is used with the datatable.

3.12 CommandButton

CommandButton extends standard JSF commandButton by adding ajax and confirmation features.

Info

Tag	commandButton
Tag Class	org.primefaces.component.commandbutton.CommandButtonTag
Component Class	org.primefaces.component.commandbutton.CommandButton
Component Type	org.primefaces.component.CommandButton
Component Family	org.primefaces.component
Renderer Type	org.primefaces.component.CommandButtonRenderer
Renderer Class	org.primefaces.component.commandbutton.CommandButtonRenderer

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	String	Label for the button
action	null	javax.el.MethodExpression	A method expression that'd be processed when button is clicked.
actionListener	null	javax.faces.event.ActionListener	An actionlistener that'd be processed when button is clicked.
immediate	FALSE	boolean	Boolean value that determines the phaseId, when true actions are processed at apply_request_values, when false at invoke_application phase.
type	submit	String	Sets the behavior of the button. Possible values are "submit" and "reset".

Name	Default	Type	Description
async	TRUE	Boolean	Specifies the submit mode, when set to true, submit would be made with Ajax.
update	null	String	Client side id of the component(s) to be updated after async partial submit request.
style	null	String	Style to be applied on the button element
styleClass	null	String	StyleClass to be applied on the button element
onstart	null	String	Javascript code to execute before ajax request is begins.
oncomplete	null	String	Javascript code to execute after ajax request is completed.
onblur	null	String	onblur dom event handler
onchange	null	String	onchange dom event handler
onclick	null	String	onclick dom event handler
ondblclick	null	String	ondblclick dom event handler
onfocus	null	String	onfocus dom event handler
onkeydown	null	String	onkeydown dom event handler
onkeypress	null	String	onkeypress dom event handler
onkeyup	null	String	onkeyup dom event handler
onmousedown	null	String	onmousedown dom event handler
onmousemove	null	String	onmousemove dom event handler
onmouseout	null	String	onmouseout dom event handler
onmouseover	null	String	onmouseover dom event handler
onmouseup	null	String	onmouseup dom event handler
onselect	null	String	onselect dom event handler
accesskey	null	String	Html accesskey attribute.
alt	null	String	Html alt attribute.
dir	null	String	Html dir attribute.
disabled	FALSE	String	Html disabled attribute.
image	null	String	Html image attribute.
label	null	String	Html label attribute.

Name	Default	Type	Description
lang	null	String	Html lang attribute.
tabindex	null	String	Html tabindex attribute.
title	null	String	Html title attribute.
readonly	FALSE	String	Html readonly attribute.

Getting started with commandButton

CommandButton component submits it's enclosed form unless it is defined as **reset**. The **submit** type works exactly same as a standard commandButton. To have a reset button the *type* of the button should be written as "**reset**".

```
public class NewBookController {
    public String saveBook() {
        //button action is called
        return null;
    }
}
```

```
<p:commandButton value= "Save"
    action="#{newBookController.saveBook}" />
```

Reset

Reset mode does not do a form submit and just resets the form contents.

```
<p:commandButton type="reset" value="Reset Form" />
```

AJAX

CommandButton has built-in ajax capabilities, to enable partial submit the attribute **async** needs to be true which is the default setting (Set it to false to disable ajax submission). When button is in async mode, it submits the parent form via ajax.

The **update** attribute is used to partially update the components after the ajax response is received. Update attribute takes a comma seperated list **client ids** of JSF components to be updated. Basically any JSF component, not just primefaces components can be updated with the Ajax response. **If you do not provide the update attribute, parent form is updated by default.**

In the following example, form is submitted with ajax and form contents are updated with the ajax response..

```
<h:form prependId="false">
    <h:panelGrid columns="2">
        <h:outputLabel for="firstname" value="Firstname:" />
        <h:inputText id="firstname" value="#{pprBean.firstname}" />

        <h:outputLabel for="surname" value="Surname" />
        <h:inputText id="surname" value="#{pprBean.surname}" />

        <p:commandButton value="Reset" type="reset"/>
        <p:commandButton value="Ajax Submit"/>
    </h:panelGrid>

    <h:panelGrid id="display" columns="2">
        <h:outputText value="Firstname:" />
        <h:outputText value="#{pprBean.firstname}" />

        <h:outputText value="Surname:" />
        <h:outputText value="#{pprBean.surname}" />
    </h:panelGrid>
</h:form>
```

Tip: You can use the ajaxStatus component to notify users about the ajax request.

Update with Ajax Response

In previous example commandButton processes and updates it's parent form by default. In cases where you require to update any JSF component on page, use the update attribute and provide the client side id(s) of the component you wish to update with ajax response. As an example in order to just update the display panelGrid's contents;

```
<p:commandButton value="Ajax Submit" update="display"/>
```

onstart and oncomplete

The two callbacks onstart and oncomplete allows you to execute custom javascript for these events. onstart is called before ajax request begins, similiar oncomplete after ajax requests ends and dom is updated.

```
<p:commandButton value="Ajax Submit" update="display"
    onstart="alert('Starting')" oncomplete="alert('Done')"/>
```

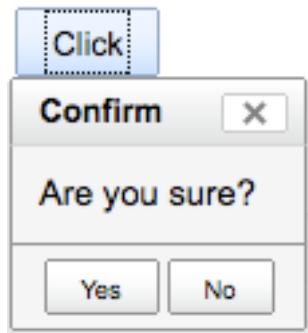
Note: See the ajaxStatus component for global start and complete callbacks.

Confirmation

Confirmation on commandButton click is a common use case in applications, the traditional way to do is using javascript confirm function, downside is that confirm boxes lack customization, styling and can be blocked by popup blockers. CommandButton is equipped with a built-in confirmation dialog.

```
<p:commandButton value="Click" action="#{buttonBean.submitButtonAction}">
    <p:confirmDialog message="Are you sure?" />
</p:commandButton>
```

When there's a nested confirmDialog component, commandButton click would not trigger instantly and display a confirmation before taking action. Default dialog looks like the following;



ConfirmDialog is a highly customizable component, buttons, message, header, styles and more can be tuned. Please see the confirmDialog section for more information.

Skinning

CommandButton renders an *input type="submit"* element and can be easily skinned using style&styleClass attributes.

3.13 CommandLink

CommandLink extends standard JSF commandLink with Ajax capabilities.

Info

Tag	commandLink
Tag Class	org.primefaces.component.commandlink.CommandLinkTag
Component Class	org.primefaces.component.commandlink.CommandLink
Component Type	org.primefaces.component.CommandLink
Component Family	org.primefaces.component
Renderer Type	org.primefaces.component.CommandLinkRenderer
Renderer Class	org.primefaces.component.commandlink.CommandLinkRenderer

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	String	Label for the button
action	null	javax.el.MethodExpression	A method expression that'd be processed when button is clicked.
actionListener	null	javax.faces.event.ActionListener	An actionlistener that'd be processed when button is clicked.
immediate	FALSE	boolean	Boolean value that determines the phaseId, when true actions are processed at apply_request_values, when false at invoke_application phase.
update	null	String	Client side id of the component(s) to be updated after async partial submit request.
style	null	String	Style to be applied on the anchor element

Name	Default	Type	Description
styleClass	null	String	StyleClass to be applied on the anchor element
onstart	null	String	Javascript code to execute before ajax request is begins.
oncomplete	null	String	Javascript code to execute after ajax request is completed.
onblur	null	String	onblur dom event handler
onchange	null	String	onchange dom event handler
onclick	null	String	onclick dom event handler
ondblclick	null	String	ondblclick dom event handler
onfocus	null	String	onfocus dom event handler
onkeydown	null	String	onkeydown dom event handler
onkeypress	null	String	onkeypress dom event handler
onkeyup	null	String	onkeyup dom event handler
onmousedown	null	String	onmousedown dom event handler
onmousemove	null	String	onmousemove dom event handler
onmouseout	null	String	onmouseout dom event handler
onmouseover	null	String	onmouseover dom event handler
onmouseup	null	String	onmouseup dom event handler
onselect	null	String	onselect dom event handler
accesskey	null	String	HTML accesskey attribute
charset	null	String	HTML charset attribute
coords	null	String	HTML coords attribute
dir	null	String	HTML dir attribute
disabled	null	String	HTML disabled attribute
hreflang	null	String	HTML hreflang attribute
rel	null	String	HTML rel attribute
rev	null	String	HTML rev attribute
shape	null	String	HTML shape attribute
tabindex	null	String	HTML tabindex attribute

Name	Default	Type	Description
target	null	String	HTML target attribute
title	null	String	HTML title attribute
type	null	String	HTML type attribute

Getting started with commandLink

CommandLink is used just like the standard h:commandLink. The difference is commandLink requires component client ids to update after the ajax request.

```
<p:commandLink actionListener="#{bean.action}" update="text">
    <h:outputText value="Ajax Submit" />
</p:commandLink>

<h:outputText id="text" value="#{bean.text}" />
```

Note: If you don't provide the components to update via the update attribute, by default parent form of commandLink is updated.

onstart and oncomplete

The two callbacks onstart and oncomplete allows you to execute custom javascript for these events. onstart is called before ajax request begins, similarly oncomplete after ajax requests ends and dom is updated.

```
<p:commandLink actionListener="#{bean.action}" update="text"
    onstart="alert('Started'); oncomplete='alert('Done');">
    <h:outputText value="Ajax Submit" />
</p:commandLink>

<h:outputText id="text" value="#{bean.text}" />
```

Skinning

CommandLink renders an html anchor element, use style and styleClass attributes for skinning this anchor element.

3.14 ConfirmDialog

ConfirmDialog is a replacement to the legacy javascript confirmation box. Its main use is to have the user do a binary decision(either yes or no). Skinning, customization and avoiding popup blockers are some the advantages over classic javascript confirmation.



Info

Tag	<code>confirmDialog</code>
Tag Class	<code>org.primefaces.component.confirmdialog.ConfirmDialogTag</code>
Component Class	<code>org.primefaces.component.confirmdialog.ConfirmDialog</code>
Component Type	<code>org.primefaces.component.ConfirmDialog</code>
Component Family	<code>org.primefaces.component</code>
Renderer Type	<code>org.primefaces.component.ConfirmDialogRenderer</code>
Renderer Class	<code>org.primefaces.component.confirmdialog.ConfirmDialogRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
<code>message</code>	null	String	Text to be displayed in body
<code>header</code>	null	String	Text for the header
<code>yesLabel</code>	Yes	String	Label of the yes button
<code>noLabel</code>	No	String	Label of the no button

Name	Default	Type	Description
severity	null	String	Message severity for the displayed icon, possible values are help info warn tip error alarm block.
visible	FALSE	boolean	Set's dialogs visibility
constraintToViewport	FALSE	boolean	Boolean value to keep the tooltip panel inside the confines of the size of viewport
close	TRUE	boolean	Displays a close icon in header
draggable	TRUE	boolean	Controls draggability
underlay	shadow	String	Specifies the type of underlay to display .Possible values: none shadow matte
modal	FALSE	boolean	Boolean value that specifies whether the document should be shielded with a partially transparent mask to require the user to close the Panel before being able to activate any elements in the document.
x	-1	int	Sets the element's "left" style property
y	-1	int	Sets the element's "top" style property
fixedCenter	FALSE	boolean	specifies whether the component should be automatically centered in the viewport on window scroll and resize
width	null	String	Width of the dialog
height	null	String	Width of the dialog
effect	null	String	Effect to be displayed when showing and hiding the dialog, values are FADE or SLIDE
effectDuration	1	double	Duration of effect in seconds

Getting started with ConfirmDialog

ConfirmDialog is used by commandButton and commandLink. Please see the commandButton component documentation for more information about the usage.

Skinning

Please check dialog component for styling confirm dialog.

3.15 DataExporter

DataExporter features is handy for exporting data listed in a datatable to various formats such as excel, pdf, csv and xml.

Info

Tag	dataExporter
Tag Class	org.primefaces.component.export.DataExporterTag
ActionListener Class	org.primefaces.component.export.DataExporter

Attributes

Name	Default	Type	Description
type	null	String	Export type: "xls", "pdf", "csv", "xml"
target	null	String	Server side id of the datatable whose date would be exported
fileName	null	String	Filename of the generated export file, defaults to datatable server side id
excludeColumns	null	String	Comma seperated list(if more than one) of column indexes to be excluded from export
preProcessor	null	javax.el.MethodExpression	PreProcessor for the exported document.
postProcessor	null	javax.el.MethodExpression	PostProcessor for the exported document.

Getting started with DataExporter

DataExporter is nested in a UICommand component such as commandButton or commandLink. For pdf exporting **iText** and for xls exporting **poi** libraries are required in the classpath.

Excel export

```
<h:commandButton value="Export as Excel">
    <p:dataExporter type="xls" target="tableId" fileName="cars"/>
</h:commandButton>
```

PDF export

```
<h:commandButton value="Export as PDF">
    <p:dataExporter type="pdf" target="tableId" fileName="cars"/>
</h:commandButton>
```

CSV export

```
<h:commandButton value="Export as CSV">
    <p:dataExporter type="csv" target="tableId" fileName="cars"/>
</h:commandButton>
```

XML export

```
<h:commandButton value="Export as XML">
    <p:dataExporter type="xml" target="tableId" fileName="cars"/>
</h:commandButton>
```

Excluding Columns

Usually datatable listings contain command components like buttons or links that need to be excluded from the exported data. For this purpose optional excludeColumns property is used to defined the column indexes to be omitted during data export.

Exporter below ignores first column, to exclude more than one column define the indexes as a comma seperated string (excludeColumns="0,2,6")

```
<h:commandButton value="Export as Excel">
    <opt:exportActionListener type="xls" target="tableId" fileName="cars"
        excludeColumns="0"/>
</h:commandButton>
```

Pre and Post Processors

In case you need to customize the exported document (add logo, caption ...), use the processor method expressions. PreProcessors are executed before the data is exported and PostProcessors are processed after data is included in the document. Processors are simple java methods taking the document as a parameter.

[Change Excel Table Header](#)

First example of processors changes the background color of the exported excel's headers.

```
<h:commandButton value="Export as XLS">
    <p:dataExporter type="xls" target="tableId" fileName="cars"
        postProcessor="#{bean.postProcessXLS}"/>
</h:commandButton>
```

```
public void postProcessXLS(Object document) {
    HSSFWorkbook wb = (HSSFWorkbook) document;
    HSSFSheet sheet = wb.getSheetAt(0);
    HSSFRow header = sheet.getRow(0);

    HSSFCellStyle cellStyle = wb.createCellStyle();
    cellStyle.setFillForegroundColor(HSSFColor.GREEN.index);
    cellStyle.setFillPattern(HSSFCellStyle.SOLID_FOREGROUND);

    for(int i=0; i < header.getPhysicalNumberOfCells();i++) {
        HSSFCell cell = header.getCell(i);

        cell.setCellStyle(cellStyle);
    }
}
```

Add Logo to PDF

This example adds a logo to the PDF before exporting begins.

```
<h:commandButton value="Export as PDF">
    <p:dataExporter type="xls" target="tableId" fileName="cars"
        preProcessor="#{bean.preProcessPDF}"/>
</h:commandButton>
```

```
public void preProcessPDF(Object document) throws IOException,
    BadElementException, DocumentException {
    Document pdf = (Document) document;
    ServletContext servletContext = (ServletContext)
FacesContext.getCurrentInstance().getExternalContext().getContext();
    String logo = servletContext.getRealPath("") + File.separator +
"images" + File.separator + "prime_logo.png";

    pdf.add(Image.getInstance(logo));
}
```

3.16 DataTable

DataTable is an enhanced version of the standard Datatable and provides built-in solutions to many commons use cases like paging, sorting, scrolling, selection, lazy loading, filtering, resizable columns and more.

<< first < prev 1 2 3 4 5 6 7 8 9 10 next > last >>			
Model	Year	Manufacturer	Color
Model_0	1963	Brand_0	Color_0
Model_1	1998	Brand_1	Color_1
Model_2	1985	Brand_2	Color_2
Model_3	1965	Brand_3	Color_3
Model_4	1989	Brand_4	Color_4
Model_5	2009	Brand_5	Color_5
Model_6	1984	Brand_6	Color_6
Model_7	2008	Brand_7	Color_7
Model_8	2009	Brand_8	Color_8
Model_9	1998	Brand_9	Color_9

<< first < prev 1 [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) next > last >>

Info

Tag	dataTable
Tag Class	org.primefaces.component.datatable.DataTableTag
Component Class	org.primefaces.component.datatable.DataTable
Component Type	org.primefaces.component.DataTable
Component Family	org.primefaces.component
Renderer Type	org.primefaces.component.DataTableRenderer
Renderer Class	org.primefaces.component.datatable.DataTableRenderer

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component

Name	Default	Type	Description
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Value of the component than can be either an EL expression of a literal text
var	null	String	Name of the request-scoped variable that'll be used as the holder of each rowdata before processing a row
rows	null	int	Number of rows to display per page
first	0	int	Index of the first row to be displayed
widgetVar	null	String	Javascript variable name of the wrapped widget
paginator	FALSE	boolean	Sets paginator
paginatorTemplate	null	String	Template for the paginator layout, default value is "{FirstPageLink} {PreviousPageLink} {PageLinks} {NextPageLink} {LastPageLink}"
rowsPerPageTemplate	null	String	Template for the rowsPerPage dropdown, default value is "25,50,100"
scrollable	FALSE	boolean	Controls scrolling when used with height and width attributes
width	null	String	Width of the datatable
height	null	String	Height of the datatable
firstPageLinkLabel	null	String	Label of the first link in paginator
previousPageLinkLabel	null	String	Label of the previous link in paginator
nextPageLinkLabel	null	String	Label of the next link in paginator
lastPageLinkLabel	null	String	Label of the last link in paginator
selection	null	Object	An object to populate the selected row data.
dynamic	FALSE	boolean	Specifies sorting/paging mode, when set to true sorting and paging is handled with ajax.

Name	Default	Type	Description
lazy	FALSE	boolean	Enables lazy loading feature.
rowIndexVar	null	String	Variable name referring to the rowIndex being processed.
paginatorPosition	both	String	Position of paginator, valid values are 'both', 'top' or 'bottom'.
emptyMessage	null	String	Message to be shown when there're records to display.
errorMessage	null	String	Message to be shown when an error occurs during data loading.
loadingMessage	null	String	Message to be shown when loading data with ajax.
sortAscMessage	null	String	Tooltip to be shown to sort a column data in ascending order.
sortDescMessage	null	String	Tooltip to be shown to sort a column data in descending order.
update	null	String	Client side id of the component(s) to be updated after ajax row selection.
style	null	String	Style of the main container element of table.
onselectStart	null	String	Javascript event handler to be called before ajax request for instant ajax row selection request begins.
onselectComplete	null	String	Javascript event handler to be called after ajax request for instant ajax row selection request is completed.

Getting started with the DataTable

Datatable is a uidata component and needs a collection of data to display. As an example, we display a car's list using the datatable throughout the datatable examples.

```
public class Car {

    private String model;
    private int year;
    private String manufacturer;
    private String color;
    ...
}
```

The code for CarListController that would be used to bind the datatable to the car list.

```
public class CarListController {

    private List<Car> cars;

    public CarListController() {
        cars = new ArrayList<Car>();
        cars.add(new Car("myModel", 2005, "ManufacturerX", "blue"));
        //add more cars
    }

    public List<Car> getCars() {
        return cars;
    }
}
```

Given the car collection, datatable can display the car list as follows;

```
<p:dataTable var="car" value="#{carListController.cars}">
    <p:column>
        <f:facet name="header">
            <h:outputText value="Model" />
        </f:facet>
        <h:outputText value="#{car.model}" />
    </p:column>

    <p:column>
        <f:facet name="header">
            <h:outputText value="Year" />
        </f:facet>
        <h:outputText value="#{car.year}" />
    </p:column>

    <p:column>
        <f:facet name="header">
            <h:outputText value="Manufacturer" />
        </f:facet>
        <h:outputText value="#{car.manufacturer}" />
    </p:column>

    <p:column>
        <f:facet name="header">
            <h:outputText value="Color" />
        </f:facet>
        <h:outputText value="#{car.color}" />
    </p:column>
</p:dataTable>
```

Paging

Pagination is a powerful feature of DataTable that can be handled purely on client side or on server side enhanced with ajax. Set the paginator to true enable the pagination.

```
<p:dataTable var="car" value="#{carListController.cars}"
    rows="10"
    paginator="true">

    ...columns

</p:dataTable>
```

When paginator is set to true, value of the rows attribute is used determine how many rows would be displayed per page.

Ajax Pagination

By default all data is sent with the response and pagination is handled on client side. Although this is very fast, for huge data it's not the optimal solution. In order to save some bandwidth and still keep the rich user experience enable the *dynamic* mode to bring in the ajax pagination.

```
<p:dataTable var="car" value="#{carListController.cars}"
    rows="10"
    paginator="true"
    dynamic="true">

    ...columns
```

Customized Paginator

Default paginator is based on the template below;

```
{FirstPageLink}{PreviousPageLink}{PageLinks}{NextPageLink}{LastPageLink}
```

Paginator is highly customizable and *paginatorTemplate* tunes this feature.

```
<p:dataTable var="car" value="#{carListController.cars}"
    paginatorTemplate="{PreviousPageLink} {CurrentPageReport}
{NextPageLink} {RowsPerPageDropdown} "
    rowsPerPageTemplate="10,15,20">

    <p:column sortable="true">
        <f:facet name="header">
            <h:outputText value="Model" />
        </f:facet>
        <h:outputText value="#{car.model}" />
    </p:column>
```

With this setting paginator looks like;

Model	Year	Manufacturer	Color
Model_40	1987	Brand_40	Color_40
Model_41	1967	Brand_41	Color_41
Model_42	1971	Brand_42	Color_42
Model_43	1989	Brand_43	Color_43
Model_44	1972	Brand_44	Color_44
Model_45	2009	Brand_45	Color_45
Model_46	1975	Brand_46	Color_46
Model_47	1963	Brand_47	Color_47
Model_48	2001	Brand_48	Color_48
Model_49	2001	Brand_49	Color_49

Full list of valid template values for paginator are;

- FirstPageLink
- PreviousPageLink
- NextPageLink
- LastPageLink
- PageLinks
- RowsPerPageDropdown (Also customizable with *rowsPerPageTemplate* attribute)
- CurrentPageReport

Location of the datatable is also customizable with the *paginatorPosition* attribute, default setting is both meaning two paginators are displayed at the top and bottom of datatable. If you only need one paginator use 'top' or 'bottom' depending on your choice.

Sorting

Sorting is controlled at column level, defining sortBy attribute enables sorting on that particular column. If datatable is dynamic, sorting is handled on server side with ajax, if it is not sorting happens on client side without a round trip.

```
<p:dataTable var="car" value="#{carListController.cars}">

    <p:column sortBy="#{car.model}">
        <f:facet name="header">
            <h:outputText value="Model" />
        </f:facet>
        <h:outputText value="#{car.model}" />
    </p:column>

    ...more columns

</p:dataTable>
```

When sorting is handled on client side, you need to define client side data parsers for proper sorting. Possible values for parser attribute of column are ‘string’(default), ‘number’ and ‘date’.

```
<p:dataTable var="car" value="#{carListController.cars}">

    <p:column sortBy="#{car.year}" parser="number">
        <f:facet name="header">
            <h:outputText value="Year" />
        </f:facet>
        <h:outputText value="#{car.year}" />
    </p:column>

    ...more columns

</p:dataTable>
```

Scrolling

Scrolling is another way of displaying huge amount data in a space saving way. When a datatable is scrollable the header becomes fixed and a scrollbar is displayed for the scrolling. Currently scrolling can be enabled both vertically and horizontally. Scrolling attribute is used along with height and width attribute to manage the scrolling. Following example is a vertically scrolling datatable with fixed header.

```
<p:DataTable var="car" value="#{carListController.cars}"
    scrollable="true" height="200px">

    ...columns

</p:DataTable>
```

Output would be;

Model	Year	Manufacturer	Color
Model_0	1977	Brand_0	Color_0
Model_1	1970	Brand_1	Color_1
Model_2	2007	Brand_2	Color_2
Model_3	1979	Brand_3	Color_3
Model_4	1964	Brand_4	Color_4
Model_5	1963	Brand_5	Color_5
Model_6	1981	Brand_6	Color_6
Model_7	2003	Brand_7	Color_7
Model_8	2008	Brand_8	Color_8

When the datatable contents get more space then 200px, datatable starts scrolling. To preserve more space horizontally, use the width attribute to enable horizontal scrolling.

```
<p:DataTable var="car" value="#{carListController.cars}"
    scrollable="true" height="200px" width="200px">

    ...columns

</p:DataTable>
```

Model	Year	Manufactur
Model_0	1977	Brand_0
Model_1	1970	Brand_1
Model_2	2007	Brand_2
Model_3	1979	Brand_3
Model_4	1964	Brand_4
Model_5	1963	Brand_5
Model_6	1981	Brand_6
Model_7	2003	Brand_7

Resizable Columns

A column can be made resizable by setting *resizable* attribute to true. A resizable column enables users to change the width of the column using the column header.

```
<p:dataTable var="car" value="#{carListController.cars}">

    <p:column resizable="true">
        <f:facet name="header">
            <h:outputText value="Model" />
        </f:facet>
        <h:outputText value="#{car.model}" />
    </p:column>

    ...more columns

</p:dataTable>
```

Row Selection

There are several built-in solutions that make row selection a piece of cake. One way is to directly click on table rows and second way is to use a selection column.

Select Single with Row Click

To select a single row when a row is clicked use the *selectionMode* attribute of the datatable.

```
<p:dataTable var="car" value="#{carListController.cars}"
    selection="#{carListController.selectedCars}" selectionMode="single">

    ...columns
</p:dataTable>
```

selectedCars is a simple object array that will be populated with the selected data once the form is submitted. Note that when a row is clicked, it'd be highlighted.

```

public class CarListController {

    private List<Car> cars;

    private Object[] selectedCars;

    public CarListController() {
        cars = new ArrayList<Car>();
        cars.add(new Car("myModel", 2005, "ManufacturerX", "blue"));
        //add more cars
    }

    public List<Car> getCars() {
        return cars;
    }

    public Object[] getSelectedCars() {
        return selectedCars;
    }

    public void setSelectedCars(Object[] selectedCars) {
        this.selectedCars = selectedCars;
    }
}

```

Model	Year	Manufacturer	Color
Model_0	1973	Brand_0	Color_0
Model_1	1973	Brand_1	Color_1
Model_2	1992	Brand_2	Color_2
Model_3	1968	Brand_3	Color_3
Model_4	2009	Brand_4	Color_4
Model_5	1997	Brand_5	Color_5
Model_6	1990	Brand_6	Color_6
Model_7	1992	Brand_7	Color_7
Model_8	1984	Brand_8	Color_8
Model_9	1994	Brand_9	Color_9

Multiple Row Selection

If you require selecting multiple rows, set the selectionMode to multiple. This way using modifier keys like ctrl, multiple rows can be selected.

Model	Year	Manufacturer	Color
Model_0	1983	Brand_0	Color_0
Model_1	1986	Brand_1	Color_1
Model_2	1970	Brand_2	Color_2
Model_3	1974	Brand_3	Color_3
Model_4	1989	Brand_4	Color_4
Model_5	2006	Brand_5	Color_5
Model_6	1964	Brand_6	Color_6
Model_7	1962	Brand_7	Color_7
Model_8	1975	Brand_8	Color_8
Model_9	1990	Brand_9	Color_9

Instant Ajax Row Selection

Two methods described above require the form to be submitted before the row selection can happen. If you need instant row selection with ajax define an update attribute pointing to the component to be updated. This way when a row is clicked ajax request is triggered and selected row(s) are assigned to the selection model instantly without a need for an implicit form submit.

```
<p:dataTable var="car" value="#{carListController.cars}"
    selection="#{carListController.selectedCars}" selectionMode="single"
    update="display">

    ...columns

</p:dataTable>

<p:outputPanel id="display">
    <ui:repeat value="#{carListController.selectedCars}" var="selectedCar">
        <h:outputText value="#{selectedCar.model}" />
    </ui:repeat>
</p:outputPanel>
```

When a row is selected on the datatable above, ajax request updates the selectedCars model with the selected cars and display outputPanel is updated with the ajax response. This is quite useful if you need to display detailed information about selected data.

Single Selection Column

Another common way to select rows from a datatable is use radios or checkboxes on a column. DataTable greatly simplifies this with the selectionMode of a column. You just need to place your selection column with the desired mode.

```
<p:dataTable var="car" value="#{carListController.cars}"
    selection="#{carListController.selectedCars}" >

    <p:column selectionMode="single" />
    ...more columns
</p:dataTable>
```

	Model	Year	Manufacturer	Color
<input type="radio"/>	Model_0	1970	Brand_0	Color_0
<input type="radio"/>	Model_1	1967	Brand_1	Color_1
<input type="radio"/>	Model_2	2008	Brand_2	Color_2
<input type="radio"/>	Model_3	1983	Brand_3	Color_3
<input type="radio"/>	Model_4	1996	Brand_4	Color_4
<input type="radio"/>	Model_5	2000	Brand_5	Color_5
<input type="radio"/>	Model_6	1990	Brand_6	Color_6
<input type="radio"/>	Model_7	1975	Brand_7	Color_7
<input type="radio"/>	Model_8	1995	Brand_8	Color_8
<input type="radio"/>	Model_9	1977	Brand_9	Color_9

Multiple Selection Column

Similarly if you need to select multiple columns with checkboxes;

```
<p:dataTable var="car" value="#{carListController.cars}"
    selection="#{carListController.selectedCars}">

    ...more columns

    <p:column selectionMode="multiple" />

</p:dataTable>
```

Model	Year	Manufacturer	Color	
Model_0	1996	Brand_0	Color_0	<input type="checkbox"/>
Model_1	1989	Brand_1	Color_1	<input checked="" type="checkbox"/>
Model_2	1999	Brand_2	Color_2	<input type="checkbox"/>
Model_3	1969	Brand_3	Color_3	<input type="checkbox"/>
Model_4	1991	Brand_4	Color_4	<input checked="" type="checkbox"/>
Model_5	1960	Brand_5	Color_5	<input type="checkbox"/>
Model_6	1979	Brand_6	Color_6	<input type="checkbox"/>
Model_7	2005	Brand_7	Color_7	<input checked="" type="checkbox"/>
Model_8	1995	Brand_8	Color_8	<input type="checkbox"/>
Model_9	1993	Brand_9	Color_9	<input type="checkbox"/>

Data Filtering

Data filters can be used either globally or on a particular column. A global filter queries text being search in all columns whereas a column based filter only searches the text at that column.

Global Filter

```

<h:outputText value="Filter:" />
<h:inputText onkeyup="cars.filter(this.value);"/>

<p:dataTable var="car" value="#{carListController.cars}" widgetVar="cars">
    ..columns
</p:dataTable>

```

Filter:

Model	Year	Manufacturer	Color
Model_0	2004	Brand_0	Color_0
Model_1	1984	Brand_1	Color_1
Model_2	1972	Brand_2	Color_2
Model_3	1996	Brand_3	Color_3
Model_4	1970	Brand_4	Color_4

Column Filters

Setting a filter is as easy as specifying filter attribute to true.

```
<p:dataTable var="car" value="#{carListController.cars}">
    <p:column filter="true">
        //contents
    </p:column>
    //more columns
</p:dataTable>
```

Model	Year	Manufacturer	Color
Model_0	2004	Brand_0	Color_0
Model_1	1984	Brand_1	Color_1
Model_2	1972	Brand_2	Color_2
Model_3	1996	Brand_3	Color_3
Model_4	1970	Brand_4	Color_4

Lazy Loading

Dealing with huge sets of data like thousand and even millions is not a trivial task, good news is datatable provides a built-in feature that can even handle billions of data in an efficient way. The idea behind lazy loading is to load only the rows of the datatable page being displayed.

In order to enable lazy loading you just need to set lazy attribute to true and provide a LazyDataModel as the value of the datatable.

```
<p:dataTable var="car" value="#{carListController.lazyModel}"
    dynamic="true" lazy="true">
    //columns
</p:dataTable>
```

```

public class CarListController {

    private LazyDataModel<Car> lazyModel;

    public CarListController() {
        /**
         * Test with one hundred million records.
         * In a real application use a count query to get the rowcount.
         */
        lazyModel = new LazyDataModel<Car>(100000000) {

            /**
             * Dummy implementation of loading a certain segment of data.
             * In a real applicaiton, this method should access db and do a limit
             * based query
             */
            @Override
            public List<Car> fetchLazyData(int first, int pageSize) {
                //Query a list of cars starting with offset first and max size
                //pagesize
            }
        };
    }

    public LazyDataModel getLazyModel() {
        return lazyModel;
    }
}

```

When lazy loading is enabled, datamodel will executed your LazyModel's fetchLazyData method with the first and pagesize variables. It's your responsibility to load a chunk of data that starts from the first offset and with size pageSize. For example if you're using jpa you can use setFirstResult(first) and setMaxResults(pageSize) api to load a certain amount of data. With lazy loading you never load the whole dataset but only the necessary portion. LazyLoading feature is also enhanced with ajax paging for rich user experience.

Skinning

As like any other primefaces component skinning is done via CSS selectors. An example skinning is as follows;

```
.yui-skin-sam .yui-dt table {
    border-color: #99FF00;
}

.yui-skin-sam .yui-dt th {
    background: url(..../design/nav.gif);
    border: none;
}

.yui-skin-sam tr.yui-dt-odd {
    background: #FFFFCC;
}

.yui-skin-sam tr.yui-dt-even {
    background: #FFFFFF;
}

.yui-skin-sam .yui-dt td {
    border-color: #FFFFFF;
}

.yui-skin-sam .yui-dt-paginator a.yui-pg-last,
.yui-skin-sam .yui-dt-paginator a.yui-pg-first,
.yui-skin-sam .yui-dt-paginator a.yui-pg-next,
.yui-skin-sam .yui-dt-paginator a.yui-pg-previous {
    color: #33CC00;
}
```

```
.yui-skin-sam a.yui-pg-page:link,
.yui-skin-sam a.yui-pg-page:hover,
.yui-skin-sam a.yui-pg-page:visited {
    background: #FFFFCC;
    color: #33CC00;
}
```

[<< first](#) [< prev](#) [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [next >](#) [last >>](#)

Model	Year	Manufacturer	Color
Model_10	1993	Brand_10	Color_10
Model_11	1974	Brand_11	Color_11
Model_12	1989	Brand_12	Color_12
Model_13	1965	Brand_13	Color_13
Model_14	1987	Brand_14	Color_14
Model_15	1979	Brand_15	Color_15
Model_16	1983	Brand_16	Color_16
Model_17	1996	Brand_17	Color_17
Model_18	1980	Brand_18	Color_18
Model_19	1992	Brand_19	Color_19

[<< first](#) [< prev](#) [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [next >](#) [last >>](#)

Full list of CSS Selectors

http://developer.yahoo.com/yui/examples/datatable/dt_skinning.html

3.17 Dialog

Dialog is a popup that doesn't reside in the normal css flow so has the ability to overlay other elements on the page. Dialog avoid popup blockers, provides customization and ajax interactions.



Info

Tag	<code>dialog</code>
Tag Class	<code>org.primefaces.component.dialog.DialogTag</code>
Component Class	<code>org.primefaces.component.dialog.Dialog</code>
Component Type	<code>org.primefaces.component.Dialog</code>
Component Family	<code>org.primefaces.component</code>
Renderer Type	<code>org.primefaces.component.DialogRenderer</code>
Renderer Class	<code>org.primefaces.component.dialog.DialogRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
<code>header</code>	null	String	Text for the header
<code>footer</code>	null	String	Text for the footer
<code>visible</code>	FALSE	boolean	Set's dialogs visibility

Name	Default	Type	Description
constrainToViewport	FALSE	boolean	Boolean value to keep the dialog inside the confines of the size of viewport
close	TRUE	boolean	Displays a close icon in header
draggable	TRUE	boolean	Controls draggability
resizable	FALSE	boolean	Controls dialog resizing
underlay	shadow	String	Specifies the type of underlay to display. Possible values: none shadow matte
modal	FALSE	boolean	Boolean value that specifies whether the document should be shielded with a partially transparent mask to require the user to close the Panel before being able to activate any elements in the document.
x	-1	int	Sets the element's "left" style property
y	-1	int	Sets the element's "top" style property
fixedCenter	FALSE	boolean	Specifies whether the component should be automatically centered.
width	null	String	Width of the dialog
height	null	String	Width of the dialog
effect	null	String	Effect to be displayed when showing and hiding the dialog, valid values are FADE or SLIDE
effectDuration	1	double	Duration of effect in seconds
zindex	null	Integer	Specifies zindex property.
minWidth	null	Integer	Minimum width of a resizable dialog.
minHeight	null	Integer	Maximum width of a resizable dialog.
style	null	String	Style of the main container element
styleClass	null	String	Style class of the main container element
widgetVar	null	String	Javascript variable name of the wrapped widget

Getting started with the Dialog

Since dialog is a container component it needs children components to display.

```
<p:dialog header="Header Text" footer="Footer Text">
    <h:outputText value="Visca el Barca!" />
    <h:outputText value="Mes que un club!" />
</p:dialog>
```

Positioning

Dialog can be positioned manually anywhere on the screen. X and Y attributes defined the left-top coordinate of the panel. Another way is to set the fixedCenter attribute to true which will fix the panel at the center of the page. By default dialog will show up at where the cursor is.

Showing&Hiding the Dialog

Showing and hiding the dialog is easy using the client side api of wrapped YUI panel. This is a case where widgetVar attribute comes in handy.

```
<p:dialog header="Header Text" widgetVar="dialog">
    <h:outputText value="Visca el Barca!" />
    <h:outputText value="Mes que un club!" />
</p:dialog>

<a href="#" onclick="dialog.show()">Show</a>
<a href="#" onclick="dialog.hide()">Hide</a>
```

Ajax Interaction

A dialog can also be used for form submitting if it has a child form. Following example demonstrates an example powered by PrimeFaces PPR.

```
<h:outputText value="Firstname:" />
<h:outputText id="name" value="#{pprBean.firstname}" />
<a href="#" onclick="dlg.show()">Enter FirstName</a>

<p:dialog header="Enter FirstName" widgetVar="dlg">
    <h:form prependId="false">
        <h:panelGrid columns="2" style="margin-bottom:10px">
            <h:outputLabel for="firstname" value="Firstname:" />
            <h:inputText id="firstname" value="#{pprBean.firstname}" />

            <p:button value="Reset" type="reset"/>
            <p:button value="Ajax Submit" update="name"
                      async="true" oncomplete="dlg.hide();"/>
        </h:panelGrid>
    </h:form>
</p:dialog>
```

When the dialog is shown, it displays a form to enter the firstname, once Ajax Submit button is clicked, dialog is hidden and outputText with id="name" is partially updated.

Header, Footer and HeaderControls

header and footer attributes are the easiest way to define the texts of dialogs. Additionally dialog supports facets for custom controls.

```
<p:dialog>
    <f:facet name="header">
        <h:outputText value="F.C. Barcelone" />
    </f:facet>
    <h:outputText value="Visca el Barca! Mes que un club!" />
</p:dialog>
```

Available facets are *header*, *footer* and *headerControls*. HeaderControls are useful if you need to place custom controls like minimize next to the close icon. This is a PrimeFaces extension to YUI dialog and .pf-dialog-headercontrols style class applies to the header controls element.

Effects

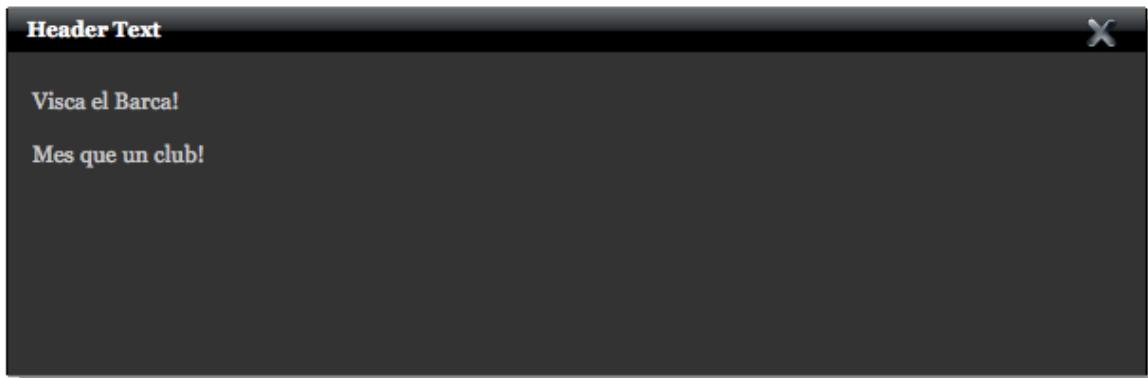
When showing and hiding the dialog, effect can be displayed to emphasize dialog. There are two effects, FADE and SLIDE. In addition effect duration is customized via effectDuration attribute in seconds.

```
<p:dialog header="F.C. Barcelona" effect="FADE" effectDuration="0.5">
    <h:outputText value="Visca el Barca! Mes que un club!" />
</p:dialog>
```

Dialog above displays a FADE effect when showing&hiding and it takes 0.5 seconds.

Skinning

Following is a styled dialog



```
.yui-skin-sam .yui-panel .bd{  
    background: #333333;  
    border-color: #000000;  
    height: 150px;  
    color:#CCCCCC;  
}  
  
.yui-skin-sam .yui-panel .hd {  
    background: url(..../images/dialoghd.gif);  
    border-color: #000000;  
    color: #FFFFFF;  
}  
  
.yui-skin-sam .yui-panel-container.shadow .underlay {  
    background-color: #333333;  
    bottom:-7px;  
    left: 7px;  
    opacity:0.5;  
    position:absolute;  
    right:-7px;  
    top:7px;  
}  
  
.yui-skin-sam .container-close {  
    background: url(..../images/dialogclose.png) no-repeat;  
}
```

Full list of CSS Selectors

<http://developer.yahoo.com/yui/examples/container/panelskin1.html>
<http://developer.yahoo.com/yui/examples/container/panelskin2.html>

3.18 Drag&Drop

Drag&Drop utilities of PrimeFaces consists of two components; Draggable and Droppable.

Draggable

Info

Tag	<code>draggable</code>
Tag Class	<code>org.primefaces.component.dnd.DraggableTag</code>
Component Class	<code>org.primefaces.component.dnd.Draggable</code>
Component Type	<code>org.primefaces.component.Draggable</code>
Component Family	<code>org.primefaces.component</code>
Renderer Type	<code>org.primefaces.component.DraggableRenderer</code>
Renderer Class	<code>org.primefaces.component.dnd.DraggableRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
<code>widgetVar</code>	null	String	Javascript variable name of the wrapped widget
<code>proxy</code>	FALSE	boolean	Uses a proxy element as drag indicator
<code>dragOnly</code>	FALSE	boolean	When set to true, draggable item cannot be dropped to a drop zone.
<code>update</code>	null	String	Client side id of the component(s) to be updated after async partial submit request.

Getting started with Draggable

Basically draggable is used a child component and makes it's parent draggable. Suppose you have a panel that you want to enable dragging.

```
<p:panel header="Draggable Panel">
    <h:outputText value="This is actually a regular p:panel." />
    <p:draggable />
</p:panel>
```

Similarly a standard JSF component can be enabled for dragging as well. Following image is enabled for dragging.

```
<h:graphicImage id="campnou" value="/ui/barca/campnou.jpg">
    <p:draggable />
</h:graphicImage>
```

Proxy

By default, the actual item is used as drag indicator, for better performance a simple proxy item can be used instead.

```
<h:graphicImage id="campnou" value="/ui/barca/campnou.jpg">
    <p:draggable proxy="true"/>
</h:graphicImage>
```

DragOnly

A draggable component can interact with drop zones which will be described in the next section. To disable this interaction, set dragOnly to false.

Droppable

Info

Tag	droppable
Tag Class	org.primefaces.component.dnd.DroppableTag
Component Class	org.primefaces.component.dnd.Droppable
Component Type	org.primefaces.component.Droppable
Component Family	org.primefaces.component
Renderer Type	org.primefaces.component.DroppableRenderer
Renderer Class	org.primefaces.component.dnd.DroppableRenderer

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
widgetVar	null	String	Javascript variable name of the wrapped widget
dropListener	null	javax.el.Method Expression	A server side listener to process a DragDrop event.

Getting started with Droppable

Usage of droppable is very similar to draggable, simply droppable makes it parent a drop zone, meaning the draggable components can be dropped onto the parent. Following examples demonstrates how to drop an image to a drop zone.

```
<p:graphicImage id="messi" value="barca/messi_thumb.jpg">
    <p:draggable />
</p:graphicImage>

<p:outputPanel id="zone" styleClass="slot">
    <p:droppable />
</p:outputPanel>
```

slot styleClass represents a small rectangle.

```
<style type="text/css">
.slot {
    background:#FF9900;
    width:64px;
    height:96px;
    display:block;
}
</style>
```

With this setup, an image with id *messi* can be dropped onto the outputPanel with id *zone*. Best part is you can execute custom callbacks when drag&drop happens on server side.

Drop Listener

A dropListener is a simple java method that's executed when a draggable item is dropped onto a droppable component. A DragDrop event is passed as a parameter holding information about the dragged and dropped components. Using the previous example just add a dropListener to the droppable.

```
<p:graphicImage id="messi" value="barca/messi_thumb.jpg">
    <p:draggable />
</p:graphicImage>

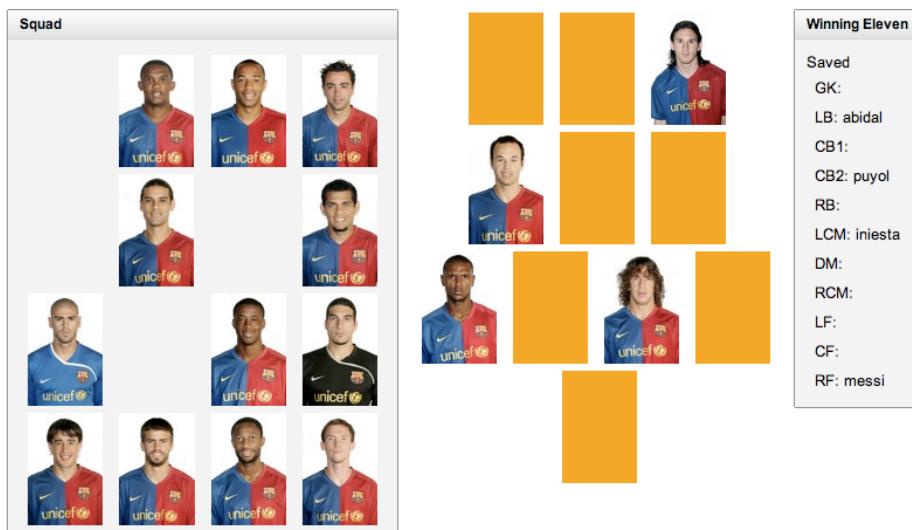
<p:outputPanel id="zone" styleClass="slot">
    <p:droppable dropListener="#{ddController.onDrop}" />
</p:outputPanel>
```

```
public void onDrop(DragDropEvent ddEvent) {
    logger.info("Dragged Id: {}", ddEvent.getDragId());
    logger.info("Dropped Id: {}", ddEvent.getDropId());
}
```

The method above just logs the item being dragged and dropped. Output of this method would be;

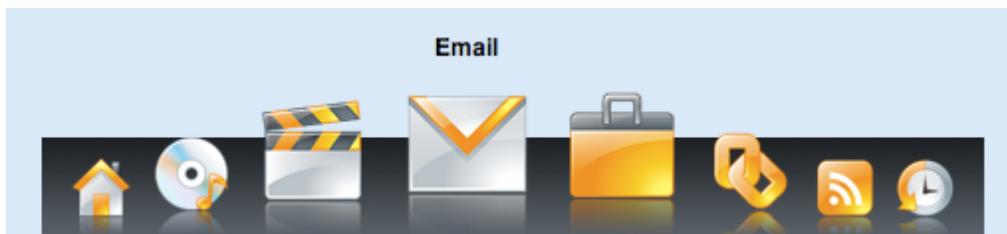
Dragged Id: messi
Dropped Id: zone

PrimeFaces component showcase demo contains a functional example to setup tactical formation of F.C. Barcelona, see the source code for more information.



3.19 Dock

Dock component mimics the famous dock in Mac Os.



Info

Tag	dock
Tag Class	org.primefaces.component.dock.DockTag
Component Class	org.primefaces.component.dock.Dock
Component Type	org.primefaces.component.Dock
Component Family	org.primefaces.component
Renderer Type	org.primefaces.component.DockRenderer
Renderer Class	org.primefaces.component.dock.DockRenderer

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
position	bottom	String	Position of the dock, <i>bottom</i> or <i>top</i> .
itemWidth	40	int	Initial width of items.
maxWidth	50	int	Maximum width of items.
proximity	90	int	Distance to enlarge.
halign	center	String	Horizontal alignment,

Getting started with the Dock

A dock is composed of the dock itself and the dockItems.

```
<p:dock>
    <p:dockItem label="Home" icon="/images/dock/home.png" url="#" />
    <p:dockItem label="Music" icon="/images/dock/music.png" url="#" />
    <p:dockItem label="Video" icon="/images/dock/video.png" url="#" />
    <p:dockItem label="Email" icon="/images/dock/email.png" url="#" />
    <p:dockItem label="Link" icon="/images/dock/link.png" url="#" />
    <p:dockItem label="RSS" icon="/images/dock/rss.png" url="#" />
    <p:dockItem label="History" icon="/images/dock/history.png" url="#" />
</p:dock>
```

Position

Dock can be located in two locations, top or bottom(default). For a dock positioned at top set position to top.

Dock Effect

When mouse is over the dock items, icons enlarge. The configuration of this effect is done via the maxWidth and proximity attributes.

3.20 DockItem

DockItem represents each element in the dock component.

Info

Tag	<code>dockItem</code>
Tag Class	<code>org.primefaces.component.dock.DockTag</code>
Component Class	<code>org.primefaces.component.dock.Dock</code>
Component Type	<code>org.primefaces.component.DockItem</code>
Component Family	<code>org.primefaces.component</code>

Attributes

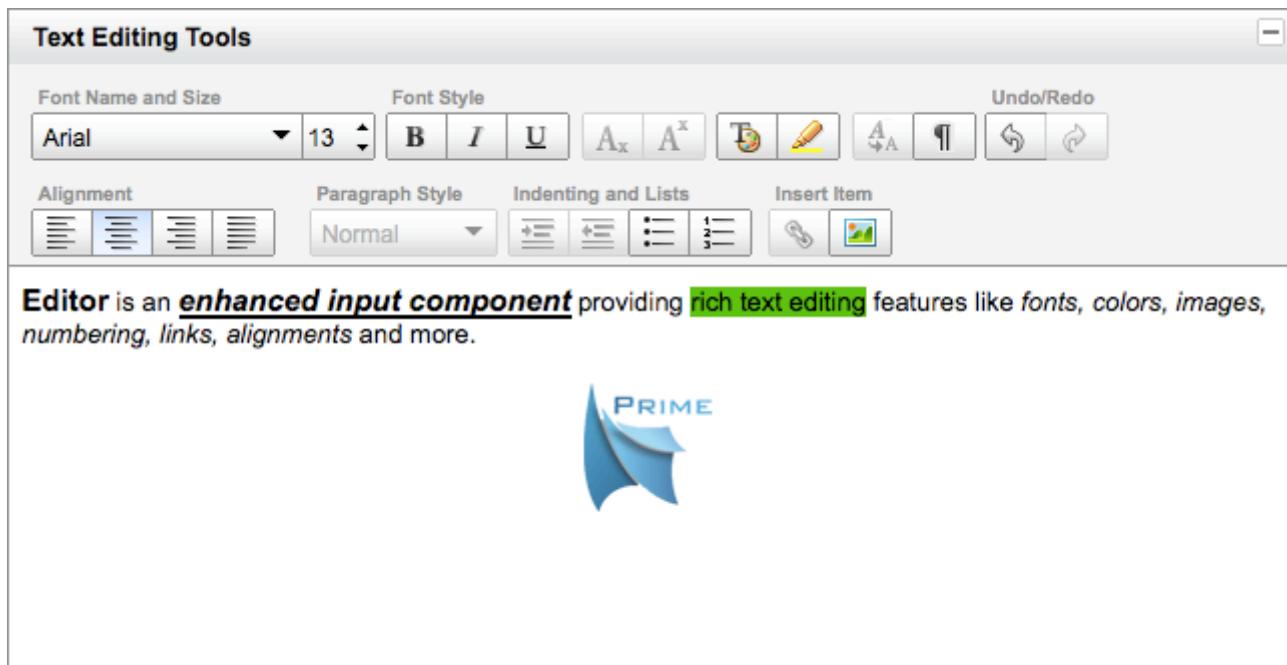
Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
<code>label</code>	null	String	Label of the item.
<code>onclick</code>	null	String	Onclick handler.
<code>icon</code>	null	String	Icon to be displayed.
<code>url</code>	null	String	URL to be used for navigation.

Getting started with DockItem

Please see Dock component section to find out how dockItem is used.

3.21 Editor

Editor is an enhanced input component providing rich text editing features. Editor supports advanced text editing features like fonts, colors, images, alignment and more.



Info

Tag	<code>editor</code>
Tag Class	<code>org.primefaces.component.editor.EditorTag</code>
Component Class	<code>org.primefaces.component.editor.Editor</code>
Component Type	<code>org.primefaces.component.Editor</code>
Component Family	<code>org.primefaces.component</code>
Renderer Type	<code>org.primefaces.component.EditorRenderer</code>
Renderer Class	<code>org.primefaces.component.editor.EditorRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.

Name	Default	Type	Description
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Value of the component than can be either an EL expression or a literal text
converter	null	Converter/String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id
immediate	FALSE	boolean	Boolean value that specifies the lifecycle phase the valueChangeEvents should be processed, when true the events will be fired at "apply request values", if immediate is set to false, valueChange Events are fired in "process validations" phase
required	FALSE	boolean	Marks component as required
validator	null	MethodBinding	A method binding expression that refers to a method validationg the input
valueChangeListerner	null	ValueChangeListener	A method binding expression that refers to a method for handling a valuchangeevent
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.
width	500px	String	Width of the editor
height	300px	String	Height of the editor
widgetVar	null	String	Javascript variable name of the wrapped widget
resizable	FALSE	boolean	Makes editor resizable when set to true
language	null	String	Language of editor labels, default is en
title	null	String	Title text of editor
disabled	fa	Boolean	Disabled editing.

Getting started with the Editor

Rich Text entered using the Editor is passed to the server using *value* expression.

```
public class MyController {
    private String text;

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

```
<p:editor value="#{myController.text}" />
```

Editor and I18N

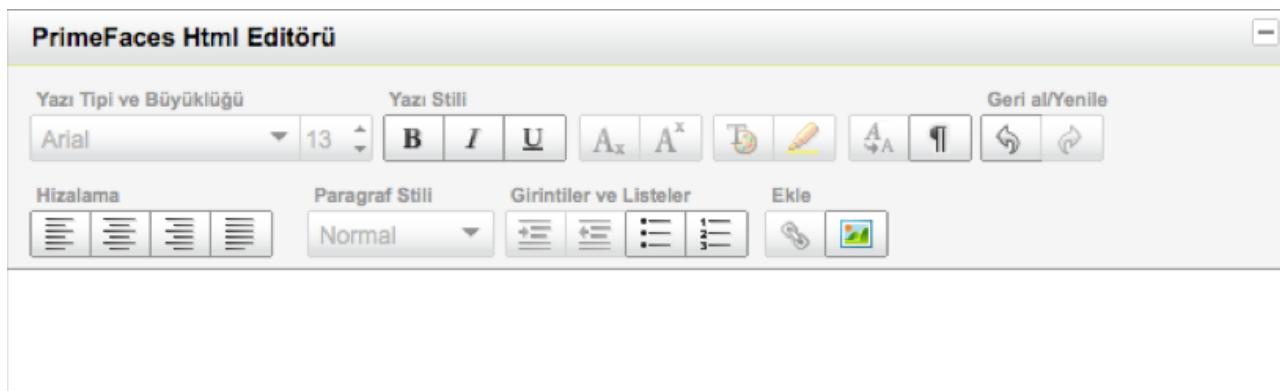
Labels like Font Style, Paragraph Style can be localized using the language attribute. Default language is English and following languages are supported out of the box.

- “tr” : Turkish
- “pt” : Portuguese

Please contact PrimeFaces team using support forum if you’re willing to provide a new translation.

Header text of the editor can also be changed via the title attribute. Following is a Turkish editor.

```
<p:editor value="#{myController.text}" title="PrimeFaces Html Editörü"
          language="tr"/>
```



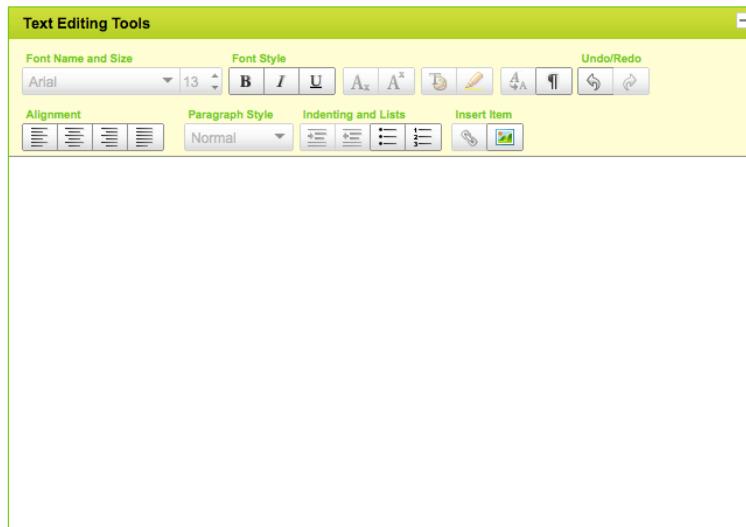
Skinning Editor

```
.yui-skin-sam .yui-editor-container {
    border-color: #33CC00;
}

.yui-skin-sam .yui-toolbar-container .yui-toolbar-titlebar h2 {
    background: url(..../design/nav.gif);
}

.yui-skin-sam .yui-toolbar-container {
    background-color: #FFFFCC ;
}

.yui-skin-sam .yui-toolbar-container .yui-toolbar-group h3 {
    color: #33CC00 ;
}
```



Full list of CSS Selectors

http://developer.yahoo.com/yui/examples/editor/skinning_editor.html

3.22 Effect

Effect component is based on the jQuery effects library.

Info

Tag	effect
Tag Class	org.primefaces.component.effect.EffectTag
Component Class	org.primefaces.component.effect.Effect
Component Type	org.primefaces.component.Effect
Component Family	org.primefaces.component
Renderer Type	org.primefaces.component.EffectRenderer
Renderer Class	org.primefaces.component.effect.EffectRenderer

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
event	null	String	Dom event to attach the event that executes the animation
type	null	String	Specifies the name of the animation
for	null	String	Component that is animated
speed	1000	int	Speed of the animation in ms

Getting started with Effect

Effect component needs to be nested inside another component. If for attribute is not provided, by default parent would be animated.

```
<h:outputText value="#{bean.value}">
    <p:effect type="pulsate" event="click" />
</h:outputText>
```

List of Effects

Following is the list of effects supported by PrimeFaces.

- Blind
- Clip
- Drop
- Explode
- Fold
- Puff
- Slide
- Scale
- Bounce
- Highlight
- Pulsate
- Shake
- Size
- Transfer

Effect Configuration

Each effect has different parameters for animation like colors, duration and more. In order to change the configuration of the animation, provide these parameters with the f:param tag.

```
<h:outputText value="#{bean.value}">
    <p:effect type="scale" event="mouseover">
        <f:param name="percent" value="90"/>
    </p:effect>
</h:outputText>
```

It's important to provide string options with single quotes.

```
<h:outputText value="#{bean.value}">
    <p:effect type="blind" event="click">
        <f:param name="direction" value="'horizontal'" />
    </p:effect>
</h:outputText>
```

For the full list of configuration parameters for each effect, please see the jquery documentation;

<http://docs.jquery.com/UI/Effects>

Animation Target

By default, effect is attached to it's parent on the specified event. There may be cases where you want to display an effect on another component on the same page while keeping the parent as the trigger. For attribute is added for this purpose.

```
<h:outputLink id="lnk" value="#">  
    <h:outputText value="Show the Barca Temple" />  
    <p:effect type="appear" event="click" for="img" />  
</h:outputLink>  
  
<h:graphicImage id="img" value="/ui/barca/campnou.jpg"  
                style="display:none"/>
```

With this setting, outputLink becomes the trigger for the effect on graphicImage. When the link is clicked, initially hidden graphicImage comes up with a fade effect.

Note: It's important for components that have the effect component as a child to have an assigned id because some components do not render their clientId's if you don't give them an id explicitly.

3.23 FileDownload

The legacy way to present dynamic binary data to the client is to write a servlet or a filter and stream the binary data. FileDownload does all the hardwork and presents an easy binary data like files stored in database.

Info

Tag	<code>fileDownload</code>
Tag Class	<code>org.primefaces.component.filedownload.FileDownloadTag</code>
ActionListener Class	<code>org.primefaces.component.filedownload.FileDownloadActionListener</code>

Attributes

Name	Default	Type	Description
<code>value</code>	<code>null</code>	<code>StreamedContent</code>	A streamed content instance

Getting started with FileDownload

A user command action is required to trigger the filedownload process. FileDownload can be attached to any command component like a commandButton or commandLink.

The value of the FileDownload must be an `org.primefaces.model.io.StreamedContent` instance. We suggest using the ready DefaultStreamedContent implementation. First parameter of the constructor is the binary stream, second is the mimeType and the third parameter is the name of the file.

```
public class FileDownloadController {
    private StreamedContent file;

    public FileDownloadController() {
        InputStream stream = this.getClass()
            .getResourceAsStream("primefaces.pdf");
        file = new DefaultStreamedContent(stream, "application/pdf",
            "downloaded_primefaces.pdf");
    }

    //getters and setters
}
```

Once the streamed image is ready, set it as the value of the fileDownload.

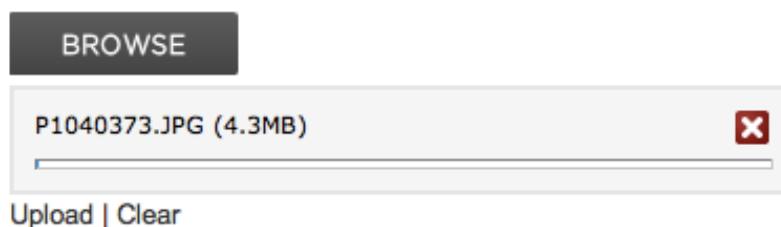
```
<h:commandButton value="Download">
    <p:fileDownload value="#{fileDownloadController.file}" />
</h:commandButton>
```

Similarly a more graphical presentation would be to use a commandlink with an image.

```
<h:commandLink value="Download">
    <p:fileDownload value="#{fileDownloadController.file}" />
    <h:graphicImage value="pdficon.gif" />
</h:commandLink>
```

3.24 FileUpload

FileUpload goes beyond the browser input type="file" functionality and features a flash-javascript solution for uploading files. File filtering, multiple uploads, partial page rendering and progress tracking are the significant features compared to legacy fileUploads. Additionally in case the user agent does not support flash or javascript, fileUpload will fallback to the legacy input type="file" and still work.



Info

Tag	<code>fileUpload</code>
Tag Class	<code>org.primefaces.component.fileupload.FileUploadTag</code>
Component Class	<code>org.primefaces.component.fileupload.FileUpload</code>
Component Type	<code>org.primefaces.component.FileUpload</code>
Component Family	<code>org.primefaces.component</code>
Renderer Type	<code>org.primefaces.component.FileUploadRenderer</code>
Renderer Class	<code>org.primefaces.component.fileupload.FileUploadRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component.
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean.
<code>fileUploadListener</code>	null	MethodExpression	Method expression to listen file upload events.
<code>multiple</code>	FALSE	boolean	Allows multi file uploads, turned off by default.

Name	Default	Type	Description
update	null	String	Client side ids of the component(s) to be updated after file upload completes.
auto	FALSE	boolean	When set to true, selecting a file starts the upload process implicitly.
label	null	String	Label of the browse button, default is 'Browse'
image	null	String	Background image of the browse button.
cancellImage	null	String	Image of the cancel button
width	null	String	Width of the browse button
height	null	String	Height of the browse button
allowTypes	null	String	Command seperated list of file extensions to accept.
description	null	String	Label to describe what types of files can be uploaded.
sizeLimit	null	Integer	Number of maximum bytes to allow.
wmode	null	String	wmode property of the flash object.
customUI	null	boolean	When custom UI is turned on upload and cancel links won't be rendered.

Getting started with FileUpload

First thing to do is to configure the fileupload filter which parses the multipart request. It's important to make PrimeFaces file upload filter the very first filter to consume the request.

```

<filter>
    <filter-name>PrimeFaces FileUpload Filter</filter-name>
    <filter-class>
        org.primefaces.webapp.filter.FileUploadFilter
    </filter-class>
</filter>

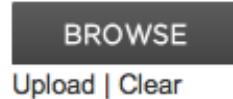
<filter-mapping>
    <filter-name>PrimeFaces FileUpload Filter</filter-name>
    <servlet-name>Faces Servlet</servlet-name>
</filter-mapping>

```

Single File Upload

By default file upload allows selecting and uploading only one file at a time, simplest file upload would be;

```
<p:fileUpload fileUploadListener="#{backingBean.handleFileUpload}" />
```



FileUploadListener is the way to access the uploaded files, when a file is uploaded defined fileUploadListener is processed with a FileUploadEvent as the parameter.

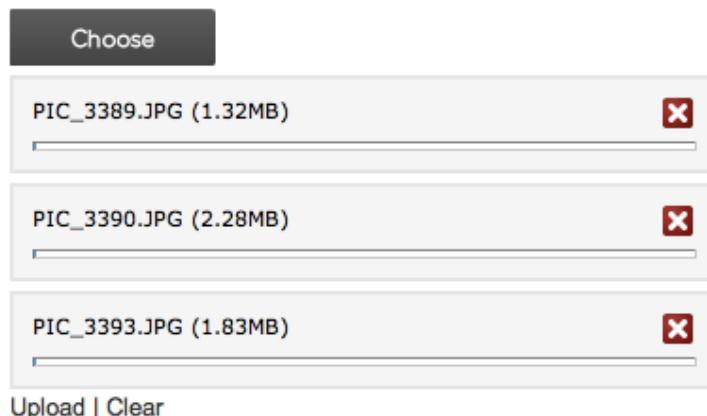
```
public class Controller {  
  
    public void handleFileUpload(FileUploadEvent event) {  
        UploadedFile file = event.getFile();  
        //application code  
    }  
}
```

UploadedFile belongs to the PrimeFaces API and contains methods to retrieve various information about the file such as filesize, contents, file type and more. Please see the JavaDocs for more information.

Multi FileUploads

Multiple fileuploads can be enabled using the multiple attribute. This way multiple files can be selected and uploaded together.

```
<p:fileUpload fileUploadListener="#{controller.handleFileUpload}"  
    multiple="true" />
```



Auto Upload

Default behavior requires users to trigger the upload process, you can change this way by setting auto to true. Auto uploads are triggered as soon as files are selected from the dialog.

```
<p:fileUpload fileUploadListener="#{controller.handleFileUpload}"
    auto="true" />
```

Partial Page Update

After the fileUpload process completes you can use the PrimeFaces PPR to update any component on the page. FileUpload is equipped with the update attribute for this purpose. Following example displays a “File Uploaded” message using the growl component after file upload.

```
<p:fileUpload fileUploadListener="#{controller.handleFileUpload}"
    multiple="true" update="messages">
</p:fileUpload>

<p:growl id="messages" />
```

```
public class Controller {

    public void handleFileUpload(FileUploadEvent event) {
        //add facesmessage to display with growl
        //application code
    }
}
```

File Filters

Users can be restricted to only select the file types you've configured, for example a file filter defined on *.jpg will only allow selecting jpg files. Several different file filters can be configured for a single fileUpload component.

```
<p:fileUpload fileUploadListener="#{controller.handleFileUpload}"
```

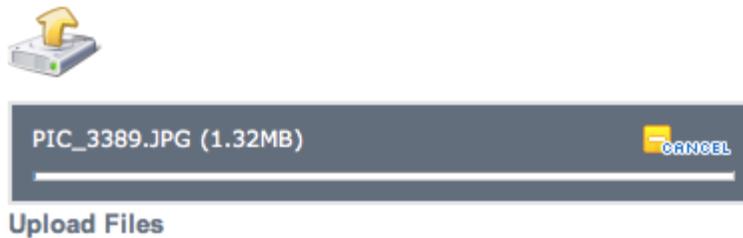
Size Limit

Most of the time you might need to restrict the file upload size, this is as simple as setting the sizeLimit configuration. Following fileUpload limits the size to 10000 bytes for each file.

```
<p:fileUpload fileUploadListener="#{controller.handleFileUpload}"
    limitSize="10000" />
```

Skinning FileUpload

FileUpload is a highly customizable component in terms of skinning. Best way to show this is start with an example. After skinning the fileUpload will look like;



```
<p:fileUpload widgetVar="uploader"
    fileUploadListener="#{fileUploadController.handleFileUpload}"
    height="48" width="48" image="/images/browse.png"
    cancelImage="/images/cancel.png" customUI="true"/>

<h:outputLink value="#" title="Upload" onclick="uploader.upload();">
    Upload Files
</h:outputLink>
```

The image of the browse button is customized using the image attribute and the image for cancel button is configured with cancellImage attribute. Note that when you use a custom image for the browse button set the height and width properties to be same as the image size.

Another important feature is the customUI. Since fileUpload is a composite component, we made the UI flexible enough to customize it for your own requirements. When customUI is set to true, default upload and cancel links are not rendered and it's up to you to handle these events if you want using the client side api. There're two simple methods upload() and clear() that you can use to plug-in your own UI.

Filter Configuration

FileUpload filter's default settings can be configured with init parameters. Two configuration options exist, threshold size and temporary file upload location.

Parameter Name	Description
thresholdSize	Maximum file size in bytes to keep uploaded files in memory. If a file exceeds this limit, it'll be temporarily written to disk.

Parameter Name	Description
uploadDirectory	Disk repository path to keep temporary files that exceeds the threshold size. By default it is System.getProperty ("java.io.tmpdir")

An example configuration below defined thresholdSize to be 50kb and uploads to user's temporary folder.

```
<filter>
    <filter-name>PrimeFaces FileUpload Filter</filter-name>
    <filter-class>
        org.primefaces.webapp.filter.FileUploadFilter
    </filter-class>
    <init-param>
        <paramater-name>thresholdSize</parameter-name>
        <parameter-value>51200</parameter-value>
    </init-param>
    <init-param>
        <paramater-name>uploadDirectory</parameter-name>
        <parameter-value>/Users/jagatai/temp</parameter-value>
    </init-param>
</filter>
```

3.25 GraphicImage

PrimeFaces GraphicImage extends standard JSF graphic image component with the ability of displaying binary data like an inputstream. Main use cases of GraphicImage is to make displaying images stored in database or on-the-fly images easier. Legacy way to do this is to come up with a Servlet that does the streaming, GraphicImage does all the hard work without the need of a Servlet.

Info

Tag	graphicImage
Tag Class	org.primefaces.component.graphicimage.GraphicImageTag
Component Class	org.primefaces.component.graphicimage.GraphicImage
Component Type	org.primefaces.component.GraphicImage
Component Family	org.primefaces.component
Renderer Type	org.primefaces.component.GraphicImageRenderer
Renderer Class	org.primefaces.component.graphicimage.GraphicImageRenderer

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Binary data to stream or context relative path.
alt	null	String	Alternate text for the image
url	null	String	Alias to value attribute
width	null	String	Width of the image
height	null	String	Height of the image
title	null	String	Title of the image
dir	null	String	Direction of the text displayed
lang	null	String	Language code

Name	Default	Type	Description
ismap	FALSE	Boolean	Specifies to use a server-side image map
usemap	null	String	Name of the client side map
style	null	String	Style of the image
styleClass	null	String	Style class of the image
onclick	null	String	onclick dom event handler
ondblclick	null	String	ondblclick dom event handler
onkeydown	null	String	onkeydown dom event handler
onkeypress	null	String	onkeypress dom event handler
onkeyup	null	String	onkeyup dom event handler
onmousedown	null	String	onmousedown dom event handler
onmousemove	null	String	onmousemove dom event handler
onmouseout	null	String	onmouseout dom event handler
onmouseover	null	String	onmouseover dom event handler
onmouseup	null	String	onmouseup dom event handler

Getting started with GraphicImage

GraphicImage requires a *org.primefaces.model.StreamedContent* content as it's value. StreamedContent is an interface and PrimeFaces provides a ready implementation called *DefaultStreamedContent*. Following examples loads an image from the classpath.

```
<p:graphicImage value="#{dynamicImageController.image}" />
```

```
public class DynamicImageController {
    private StreamedContent image;
    //getters&setters

    public DynamicImageController() {
        InputStream stream = this.getClass().getResourceAsStream("barcalogo.jpg");
        image = new DefaultStreamedContent(stream, "image/jpeg");
    }
}
```

DefaultStreamedContent gets an inputstream as the first parameter and mime type as the second. Please see the javadocs if you require more information.

In a real life application, you can create the inputstream after reading the image from the database. For example `java.sql.ResultSet` API has the `getBinaryStream()` method to read blob files stored in database.

Displaying Charts with JFreeChart

StreamedContent is a powerful API that can display images created on-the-fly as well. Here's an example that generates a chart with JFreeChart and displays it with `p:graphicImage`.

```
public class BackingBean {

    private StreamedContent chartImage;

    public BackingBean() {
        try {
            JFreeChart jfreechart = ChartFactory.createPieChart
("Turkish Cities", createDataset(), true, true, false);
            File chartFile = new File("dynamichart");
            ChartUtilities.saveChartAsPNG(chartFile, jfreechart, 375,
300);
            chartImage = new DefaultStreamedContent(new FileInputStream
(chartFile), "image/png");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

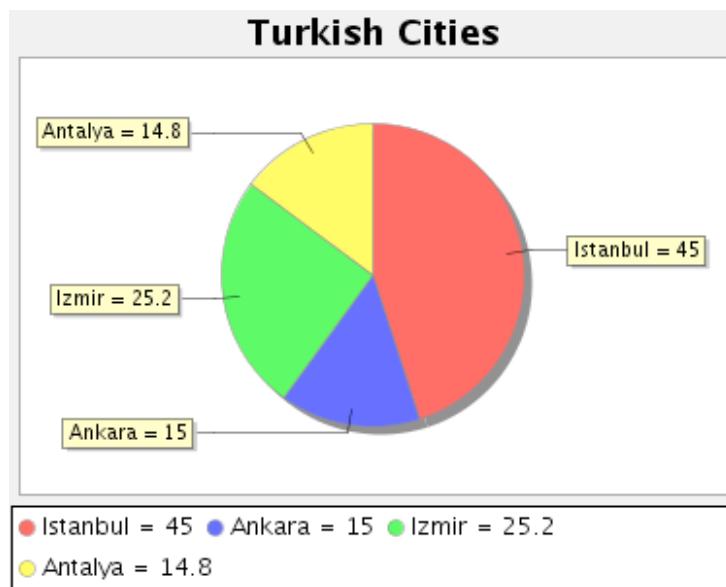
    private PieDataset createDataset() {
        DefaultPieDataset dataset = new DefaultPieDataset();
        dataset.setValue("Istanbul", new Double(45.0));
        dataset.setValue("Ankara", new Double(15.0));
        dataset.setValue("Izmir", new Double(25.2));
        dataset.setValue("Antalya", new Double(14.8));

        return dataset;
    }

    //getters and setters
}
```

`<p:graphicImage value="#{backingBean.chartImage}" />`

Basically p:graphicImage makes any JSF chart component using JFreechart obsolete and lets you to avoid wrappers to take full advantage of JFreechart API.



Displaying a Barcode

Similar to the chart example, a barcode can be generated as well. This sample uses barbecue project for the barcode API.

```
public class BackingBean {

    private StreamedContent barcode;

    public BackingBean() {
        try {
            File barcodeFile = new File("dynamicbarcode");
            BarcodeImageHandler.saveJPEG(BarcodeFactory.createCode128
("PRIMEFACES"), barcodeFile);
            barcode = new DefaultStreamedContent(new FileInputStream
(barcodeFile), "image/jpeg");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    //getters and setters
}
```

```
<p:graphicImage value="#{BackingBean.barcode}" />
```



Passing Parameters

Behind the scenes, dynamic images are generated by a different request whose format is defined initially by the `graphicImage`. Suppose you want to generate different images depending on a request parameter. Problem is the request parameter can only be available at initial load of page containing the `graphicImage`, you'd lose the value of the parameter for the actual request that generates the image. To solve this, you can pass request parameters to the `graphicImage` via `f:param` tags, as a result the actual request rendering the image can have access to these values.

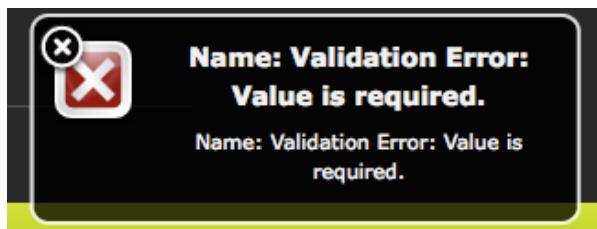
Displaying Regular Images

`GraphicImage` also keeps the behavior in case to display non dynamic images as well like the standard `h:graphicImage`.

```
<p:graphicImage value="barcalogo.jpg" />
```

3.26 Growl

Growl is based on the Mac's growl notification widget and used to display FacesMessages from the server side. Growl is simply a replacement of h:messages.



Info

Tag	growl
Tag Class	org.primefaces.component.growl.GrowlTag
Component Class	org.primefaces.component.growl.Growl
Component Type	org.primefaces.component.Growl
Component Family	org.primefaces.component
Renderer Type	org.primefaces.component.GrowlRenderer
Renderer Class	org.primefaces.component.growl.GrowlRenderer

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
sticky	FALSE	boolean	Specifies if the message should stay instead of hidden automatically.
showSummary	TRUE	boolean	Specifies if the summary of message should be displayed.
showDetail	FALSE	boolean	Specifies if the detail of message should be displayed.

Name	Default	Type	Description
globalOnly	FALSE	boolean	When true, only facesmessages without clientids are displayed.
life	3000	integer	Duration in milliseconds to display non-sticky messages.
warnIcon	null	String	Image of the warning messages.
infoIcon	null	String	Image of the info messages.
errorIcon	null	String	Image of the error messages.
fatalIcon	null	String	Image of the fatal messages.

Getting Started with Growl

Growl is a replacement of h:messages and usage is very similar indeed. Simply place growl anywhere on your page, since messages are displayed as an overlay, the location of growl in JSF page does not matter.

```
<p:growl />
```

Lifetime of messages

By default each message will be displayed for 3000 ms and then hidden. A message can be made sticky meaning it'll never be hidden automatically.

```
<p:growl sticky="true"/>
```

If growl is not working in sticky mode, it's also possible to tune the duration of displaying messages. Following growl will display the messages for 5 seconds.

```
<p:growl life="5000"/>
```

Growl with Ajax

If you need to display messages with growl after an ajax request you just need to update it just like a regular component.

```
<p:growl id="messages"/>
```

```
<p:commandButton value="Submit" update="messages" />
```

3.27 HotKey

HotKey is a generic key binding component that can bind any formation of keys to javascript event handlers or ajax calls.

Info

Tag	hotkey
Tag Class	org.primefaces.component.hotkey.HotKeyTag
Component Class	org.primefaces.component.hotkey.HotKey
Component Type	org.primefaces.component.HotKey
Component Family	org.primefaces.component
Renderer Type	org.primefaces.component.HotKeyRenderer
Renderer Class	org.primefaces.component.hotkey.HotKeyRenderer

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component.
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
bind	null	String	The Key binding.
handler	null	String	Javascript event handler to be executed when the key binding is pressed.
action	null	javax.el.MethodExpression	A method expression that'd be processed in the partial request caused by uiajax.
actionListener	null	javax.faces.event.ActionListener	An actionlistener that'd be processed in the partial request caused by uiajax.
immediate	FALSE	boolean	Boolean value that determines the phaseld, when true actions are processed at apply_request_values, when false at invoke_application phase.

Name	Default	Type	Description
partialSubmit	FALSE	boolean	Flag for partial process, when set to true, parent form will be submitted but only the component the ajax event is attached to will be processed by JSF
update	null	String	Client side id of the component(s) to be updated after async partial submit request
onstart	null	String	Javascript code to execute before ajax request is begins.
oncomplete	null	String	Javascript code to execute after ajax request is completed.

Getting Started with HotKey

HotKey is used in two ways, either on client side with the event handler or with ajax support. Simples example would be;

```
<p:hotkey bind="a" handler="alert('Pressed a');" />
```

When this hotkey is on page, pressing the a key will alert the 'Pressed key a' text.

Key combinations

Most of the time you'd need key combinations rather than a single key.

```
<p:hotkey bind="ctrl+s" handler="alert('Pressed ctrl+s');" />
```

```
<p:hotkey bind="ctrl+shift+s" handler="alert('Pressed ctrl+shift+s')"/>
```

Integration

Here's an example demonstrating how to integrate hotkeys with a client side api. Using left and right keys will switch the images displayed via the p:imageSwitch component.

```
<p:hotkey bind="left" handler="switcher.previous();"/>
<p:hotkey bind="right" handler="switcher.next();"/>

<p:imageSwitch widgetVar="switcher">
    //content
</p:imageSwitch>
```

Ajax Support

Ajax is a built-in feature of hotKeys meaning you can do ajax calls with key combinations. Following form can be submitted with the *ctrl+shift+s* combination.

```
<h:form prependId="false">

    <p:hotkey bind="ctrl+shift+s" update="display"
        actionListener="# {hotkeyController.action}"/>

    <h:panelGrid columns="2" style="margin-bottom:10px">
        <h:outputLabel for="firstname" value="Firstname:" />
        <h:inputText id="firstname" value="#{pprBean.firstname}" />
    </h:panelGrid>

    <h:outputText id="dsplay" value="Hello: #{pprBean.firstname}"
        rendered="#{not empty pprBean.firstname}"/>

</h:form>
```

Note that hotkey must be nested inside a form to use the ajax support. We're also planning to add built-in hotkey support for p:commandButton and p:commandLink since hotkeys are a common use case for command components.

3.28 IdleMonitor

IdleMonitor watches users' actions on a page and notify several callbacks in case they go idle or active again.

Source

```

01. <view plain copy to clipboard print ?>
02. <p:idleMonitor timeout="10000">
03.   <p:dialog header="What's happening?" 
04.     " widgetVar="idleDialog" modal="true" fixedCenter="true" close="false"
05.       width="400px">
06.         <h:outputText value="Dude, are you there?" />
07.       </p:dialog>

```

Tag	idleMonitor
Tag Class	org.primefaces.component.idlemonitor.IdleMonitorTag
Component Class	org.primefaces.component.idlemonitor.IdleMonitor
Component Type	org.primefaces.component.IdleMonitor
Component Family	org.primefaces.component
Renderer Type	org.primefaces.component.IdleMonitorRenderer
Renderer Class	org.primefaces.component.idlemonitor.IdleMonitor

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
timeout	300000	int	Time to wait in milliseconds until deciding if the user is idle. Default is 5 minutes.
onidle	null	String	Javascript event to execute when user goes idle

Name	Default	Type	Description
onactive	null	String	Javascript event to execute when user goes active
idleListener	null	javax.el.Method Expression	Server side event to be called in case user goes idle
update	null	String	Client side id of the component(s) to be updated after async partial submit request

Getting Started with IdleMonitor

To begin with, you can listen to events that are called when a user goes idle or becomes active again. Example below displays a warning dialog onidle and hides it back when user moves the mouse or uses the keyboard.

```
<p:idleMonitor onidle="idleDialog.show();" onactive="idleDialog.hide();"/>

<p:dialog header="What's happening?" widgetVar="idleDialog" modal="true"
    fixedCenter="true" close="false" width="400px" visible="true">
    <h:outputText value="Dude, are you there?" />
</p:dialog>
```

Controlling Timeout

By default, idleMonitor waits for 5 minutes (300000 ms) until triggering the onidle event. You can customize this duration with the timeout attribute.

IdleListener

Most of the time you may need to be notified on server side as well about IdleEvents so that necessary actions like invalidating the session or logging can be done. For this purpose use the idleListeners that are notified with ajax. A conventional idleEvent is passed as parameter to the idleListener.

```
<p:idleMonitor idleListener="#{idleMonitorController.handleIdle}"/>
```

HandleIdle is a simple method that's defined in idleMonitorController bean.

```
public void handleIdle(IdleEvent event) {
    //Invalidate user
}
```

AJAX Update

IdleMonitor uses PrimeFaces PPR to update the dom with the server response after an idleListener is notified. Example below adds a message and updates an outputText.

```
<h:form prependId="false">
    <p:idleMonitor idleListener="#{idleMonitorController.handleIdle}"
        update="message"/>

    <h:outputText id="message" value="#{idleMonitorController.msg}" />
</h:form>
```

```
public class IdleMonitorController {

    private String msg;

    public String getMsg() {
        return msg;
    }

    public void setMsg(String msg) {
        this.msg = msg;
    }

    public void idleListener(IdleEvent event) {
        msg = "Message from server: Your session is closed";

        //invalidate session
    }
}
```

Note: An idleMonitor must be enclosed in a form if an idleListener is defined.

3.29 ImageCropper

ImageCropper allows cropping a certain region of an image. A new image is created containing the cropped area and assigned to a CroppedImage instanced on the server side.



Info

Tag	<code>imageCropper</code>
Tag Class	<code>org.primefaces.component.imagecropper.ImageCropperTag</code>
Component Class	<code>org.primefaces.component. imagecropper.ImageCropper</code>
Component Type	<code>org.primefaces.component.ImageCropper</code>
Component Family	<code>org.primefaces.component</code>
Renderer Type	<code>org.primefaces.component.ImageCropperRenderer</code>
Renderer Class	<code>org.primefaces.component.imagecropper.ImageCropperRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component

Name	Default	Type	Description
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Value of the component than can be either an EL expression of a literal text
converter	null	Converter/ String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id
immediate	FALSE	boolean	Boolean value that specifies the lifecycle phase the valueChangeEvents should be processed, when true the events will be fired at "apply request values", if immediate is set to false, valueChange Events are fired in "process validations" phase
required	FALSE	boolean	Marks component as required
validator	null	MethodBindi ng	A method binding expression that refers to a method validationg the input
valueChangeListener	null	ValueChang eListener	A method binding expression that refers to a method for handling a valuchangeevent
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.
image	null	String	Context relative path to the image.
widgetVar	null	String	Javascript variable name of the wrapped widget

Getting started with the ImageCropper

Image to be cropped is provided via the *image* attribute. ImageCropper is an input component and the cropped area of the original image is used to create a new image, this new image can be accessed on the server side jsf backing bean by setting the *value* attribute of the image cropper.

Assuming the image is at %WEBAPP_ROOT%/campnou.jpg

```
<p:imageCropper value="#{myBean.croppedImage}" image="/campnou.jpg" />
```

```
public class MyBean {
    private CroppedImage croppedImage;

    public CroppedImage getCroppedImage() {
        return croppedImage;
    }

    public void setCroppedImage(CroppedImage croppedImage) {
        this.croppedImage = croppedImage;
    }
}
```

CroppedImage is a PrimeFaces api and contains handy information about the crop process. Following table describes CroppedImage properties.

Property	Type	Description
originalFileName	String	Name of the original file that's cropped
bytes	byte[]	Contents of the cropped area as a byte array
left	int	Left coordinate
right	int	Right coordinate
width	int	Width of the cropped image
height	int	Height of the cropped image

Probably most important property is the bytes since it contains the byte[] representation of the cropped area, an example that saves the cropped part to a folder in web server is described below.

```
<p:imageCropper value="#{myBean.croppedImage}"
    image="/campnou.jpg">
</p:imageCropper>

<h:commandButton value="Crop" action="#{myBean.crop}" />
```

```

public class ImageCropperBean {

    private CroppedImage croppedImage;

    public CroppedImage getCroppedImage() {
        return croppedImage;
    }

    public void setCroppedImage(CroppedImage croppedImage) {
        this.croppedImage = croppedImage;
    }

    public String crop() {
        ServletContext servletContext = (ServletContext)
FacesContext.getCurrentInstance().getExternalContext().getContext();
        String newFileName = servletContext.getRealPath("") +
File.separator + "ui" + File.separator + "barca" + File.separator+
croppedImage.getOriginalFileName() + "cropped.jpg";

        FileImageOutputStream imageOutput;
        try {
            imageOutput = new FileImageOutputStream(new File
(newFileName));
            imageOutput.write(croppedImage.getBytes(), 0,
croppedImage.getBytes().length);
            imageOutput.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }
}

```

External Images

ImageCropper has the ability to crop external images as well.

```

<p:imageCropper value="#{myBean.croppedImage}"
    image="http://primefaces.prime.com.tr/en/images/schema.png">
</p:imageCropper>

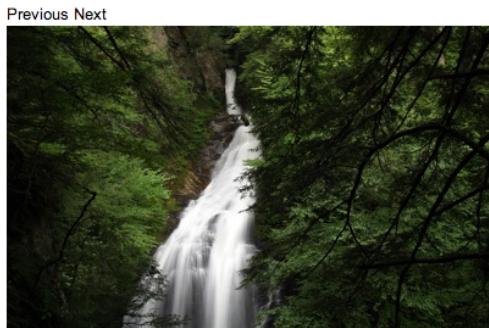
```

Context Relative Path

For local images, ImageCropper always requires the image path to be context relative. So to accomplish this simply just add slash ("path/to/image.png") and imagecropper will recognize it at %WEBAPP_ROOT%/path/to/image.png. Action url relative local images are not supported.

3.30 ImageSwitch

ImageSwitch component is used to enable switching between a set of images with some nice effects. ImageSwitch also provides a simple client side api for flexibility.



Info

Tag	<code>imageSwitch</code>
Tag Class	<code>org.primefaces.component.imageswitch.ImageSwitchTag</code>
Component Class	<code>org.primefaces.component.imageswitch.ImageSwitch</code>
Component Type	<code>org.primefaces.component.ImageSwitch</code>
Component Family	<code>org.primefaces.component</code>
Renderer Type	<code>org.primefaces.component.ImageSwitchRenderer</code>
Renderer Class	<code>org.primefaces.component.imageswitch.ImageSwitchRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
<code>effect</code>	null	String	Name of the effect for transition.
<code>speed</code>	500	int	Speed of the effect in milliseconds.
<code>slideshowSpeed</code>	3000	int	Slideshow speed in milliseconds.
<code>slideshowAuto</code>	TRUE	boolean	Starts slideshow automatically on page load.

Getting started with ImageSwitch

ImageSwitch component needs a set of images to display. Provide the image collection as a set of children components.

```
<p:imageSwitch effect="FlyIn" widgetVar="imageswitch">
    <p:graphicImage value="/images/nature1.jpg" />
    <p:graphicImage value="/images/nature2.jpg" />
    <p:graphicImage value="/images/nature3.jpg" />
    <p:graphicImage value="/images/nature4.jpg" />
</p:imageSwitch>
```

You need to use the ImageSwitch client side api to trigger the transitions. Example below uses two span elements to navigate between the images.

```
<span onclick="imageswitch.previous();">Previous</span>
<span onclick="imageswitch.next();">Next</span>
```

Client Side API

Method	Description
void previous()	Switches to previous image.
void next()	Switches to next image.
void startSlideshow();	Manually starts a slideshow.
void stopSlideshow();	Manually stops a slideshow.

Use the widgetVar to get the variable name of the client side widget.

Effect Speed

The speed is considered in terms of milliseconds and specified via the speed attribute.

```
<p:imageSwitch effect="FlipOut" speed="150" widgetVar="imageswitch" >
    //set of images
</p:imageSwitch>
```

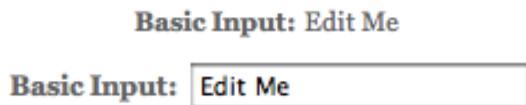
List of Effects

ImageSwitch supports a wide range of transition effects. Following is the full list, note that values are case sensitive.

FadeIn, FlyIn, FlyOut, FlipIn, FlipOut, ScrollIn, ScrollOut, SingleDoor, DoubleDoor

3.31 Inplace

Inplace provides easy inplace editing and inline content display. Inplace consists of two members, display element is the initial clickable label and inline element is the hidden content that'll be displayed when display element is toggled.



Info

Tag	<code>inplace</code>
Tag Class	<code>org.primefaces.component.inplace.InplaceTag</code>
Component Class	<code>org.primefaces.component.inplace.Inplace</code>
Component Type	<code>org.primefaces.component.Inplace</code>
Component Family	<code>org.primefaces.component</code>
Renderer Type	<code>org.primefaces.component.InplaceRenderer</code>
Renderer Class	<code>org.primefaces.component.inplace.InplaceRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
<code>label</code>	null	String	Label to be shown in display mode.
<code>effect</code>	fade	String	Effect to be used when toggling.
<code>effectSpeed</code>	normal	String	Speed of the effect.
<code>disabled</code>	FALSE	boolean	Prevents hidden content to be shown.
<code>widgetVar</code>	null	String	Javascript variable name of the client side object.

Getting started with Inplace

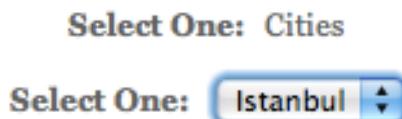
The inline component needs to be a child of inplace.

```
<p:inplace>
    <h:inputText value="Edit me" />
</p:inplace>
```

Custom Labels

By default inplace displays it's first child's value as the label, you can customize it via the label attribute.

```
<h:outputText value="Select One" />
<p:inplace label="Cities">
    <h:selectOneMenu>
        <f:selectItem itemLabel="Istanbul" itemValue="Istanbul" />
        <f:selectItem itemLabel="Ankara" itemValue="Ankara" />
    </h:selectOneMenu>
</p:inplace>
```



Effects

Default effect is fadeIn and fadeOut meaning display element will fadeOut and inline content will be shown with fadeOut effect. Other possible effect is 'slide', also effect speed can be tuned with values 'slow', 'normal' and 'fast'.

```
<p:inplace label="Show Image" effect="slide" effectSpeed="fast">
    <p:graphicImage value="/images/nature1.jpg" />
</p:inplace>
```

Skinning Inplace

Style Class	Applies
.pf-inplace-highlight	Display element when hovered.
.pf-inplace-display	Display element.
.pf-inplace-display-disabled	Disabled display element.
.pf-inplace-content	Inline content.

3.32 InputMask

InputMask forces an input to fit in a defined mask template.

Date:	10/01/2009
Phone:	(213) 421-3423
Phone with Ext:	(645) 645-7447 x65474
taxId:	21-3214231
SSN:	534-53-4264
Product Key:	nk-231-h432

Info

Tag	inputMask
Tag Class	org.primefaces.component.inputmask.InputMaskTag
Component Class	org.primefaces.component.inputmask.InputMask
Component Type	org.primefaces.component.InputMask
Component Family	org.primefaces.component
Renderer Type	org.primefaces.component.InputMaskRenderer
Renderer Class	org.primefaces.component.inputmask.InputMask

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
mask	null	Integer	Mask template
placeHolder	null	String	PlaceHolder in mask template.
value	null	Object	Value of the component than can be either an EL expression or a literal text

Name	Default	Type	Description
converter	null	Converter/ String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id
immediate	FALSE	boolean	Boolean value that specifies the lifecycle phase the valueChangeEvents should be processed, when true the events will be fired at "apply request values", if immediate is set to false, valueChange Events are fired in "process validations" phase
required	FALSE	boolean	Marks component as required
validator	null	MethodBinding	A method binding expression that refers to a method validationg the input
valueChangeListener	null	ValueChangeListener	A method binding expression that refers to a method for handling a valuchangeevent
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.
accesskey	null	String	Html accesskey attribute
alt	null	String	Html alt attribute
dir	null	String	Html dir attribute
disabled	FALSE	Boolean	Html disabled attribute
lang	null	String	Html lang attribute
maxlength	null	Integer	Html maxlength attribute
onblur	null	String	Html onblur attribute
onchange	null	String	Html onchange attribute
onclick	null	String	Html onclick attribute
ondblclick	null	String	Html ondblclick attribute
onfocus	null	String	Html onfocus attribute
onkeydown	null	String	Html onkeydown attribute
onkeypress	null	String	Html onkeypress attribute

Name	Default	Type	Description
onkeyup	null	String	Html onkeyup attribute
onmousedown	null	String	Html onmousedown attribute
onmousemove	null	String	Html onmousemove attribute
onmouseout	null	String	Html onmouseout attribute
onmouseover	null	String	Html onmouseover attribute
onmouseup	null	String	Html onmouseup attribute
readonly	FALSE	Boolean	Html readonly attribute
size	null	Integer	Html size attribute
style	null	String	Html style attribute
styleClass	null	String	Html styleClass attribute
tabindex	null	Integer	Html tabindex attribute
title	null	String	Html title attribute

Getting Started with InputMask

InputMask is actually an extended h:inputText and usage is very similar. InputMask below enforces input to be in 99/99/9999 date format.

```
<p:inputMask value="#{bean.field}" mask="99/99/9999" />
```

Mask Examples

```
<h:outputText value="Phone: " />
<p:inputMask value="#{maskController.phone}" mask="(999) 999-9999"/>

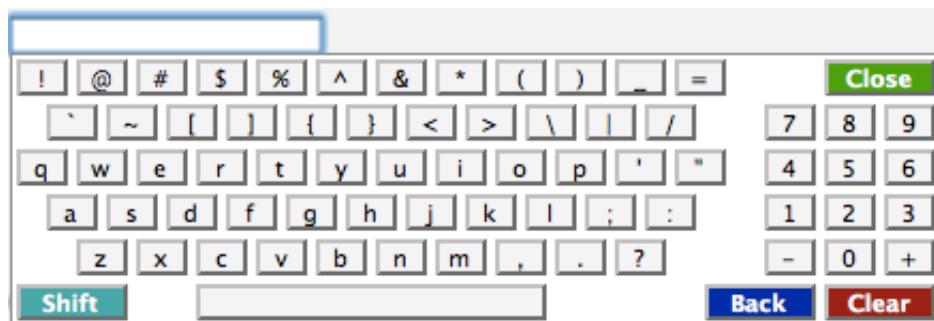
<h:outputText value="Phone with Ext: " />
<p:inputMask value="#{maskController.phoneExt}" mask="(999) 999-9999?
x99999"/>

<h:outputText value="SSN: " />
<p:inputMask value="#{maskController.ssn}" mask="999-99-9999"/>

<h:outputText value="Product Key: " />
<p:inputMask value="#{maskController.productKey}" mask="a*-999-a999"/>
```

3.33 Keyboard

Keyboard is an input component that uses a virtual keyboard to provide the input. Important features are the customizable layouts and skinning capabilities.



Info

Tag	<code>keyboard</code>
Tag Class	<code>org.primefaces.component.keyboard.KeyboardTag</code>
Component Class	<code>org.primefaces.component.keyboard.Keyboard</code>
Component Type	<code>org.primefaces.component.Keyboard</code>
Component Family	<code>org.primefaces.component</code>
Renderer Type	<code>org.primefaces.component.KeyboardRenderer</code>
Renderer Class	<code>org.primefaces.component.keyboard.KeyboardRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
<code>value</code>	null	Object	Value of the component than can be either an EL expression or a literal text

Name	Default	Type	Description
converter	null	Converter /String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id
immediate	FALSE	boolean	Boolean value that specifies the lifecycle phase the valueChangeEvents should be processed, when true the events will be fired at "apply request values", if immediate is set to false, valueChange Events are fired in "process validations" phase
required	FALSE	boolean	Marks component as required
validator	null	MethodBinding	A method binding expression that refers to a method validationg the input
valueChangeLister	null	ValueChangeListen er	A method binding expression that refers to a method for handling a valuchangeevent
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.
accesskey	null	String	Html accesskey attribute
alt	null	String	Html alt attribute
dir	null	String	Html dir attribute
disabled	FALSE	Boolean	Html disabled attribute
lang	null	String	Html lang attribute
maxlength	null	Integer	Html maxlength attribute
onblur	null	String	Html onblur attribute
onchange	null	String	Html onchange attribute
onclick	null	String	Html onclick attribute
ondblclick	null	String	Html ondblclick attribute
onfocus	null	String	Html onfocus attribute
onkeydown	null	String	Html onkeydown attribute
onkeypress	null	String	Html onkeypress attribute
onkeyup	null	String	Html onkeyup attribute

Name	Default	Type	Description
onmousedown	null	String	Html onmousedown attribute
onmousemove	null	String	Html onmousemove attribute
onmouseout	null	String	Html onmouseout attribute
onmouseover	null	String	Html onmouseover attribute
onmouseup	null	String	Html onmouseup attribute
readonly	FALSE	Boolean	Html readonly attribute
size	null	Integer	Html size attribute
style	null	String	Html style attribute
styleClass	null	String	Html styleClass attribute
tabindex	null	Integer	Html tabindex attribute
title	null	String	Html title attribute
password	FALSE	boolean	Makes the input a password field.
showMode	focus	String	Specifies the showMode, 'focus', 'button', 'both'
buttonImage	null	String	Image for the button.
buttonImageOnly	FALSE	boolean	When set to true only image of the button would be displayed.
effect	fadeIn	String	Effect of the display animation.
effectDuration	null	String	Length of the display animation.
layout	qwerty	String	Built-in layout of the keyboard.
layoutTemplate	null	String	Template of the custom layout.
keypadOnly	focus	boolean	Specifies displaying a keypad instead of a keyboard.
promptLabel	null	String	Label of the prompt text.
closeLabel	null	String	Label of the close key.
clearLabel	null	String	Label of the clear key.
backspaceLabel	null	String	Label of the backspace key.

Getting Started with Keyboard

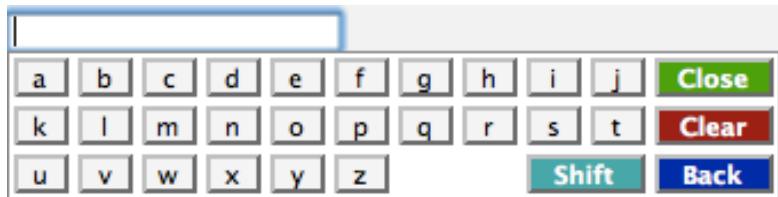
Keyboard is used just like a simple inputText, by default when the input gets the focus a keyboard is displayed.

```
<p:keyboard value="#{bean.value}" />
```

Built-in Layouts

There're a couple of built-in keyboard layouts these are 'qwerty', 'qwertyBasic' and 'alphabetic'. For example keyboard below has the alphabetic layout.

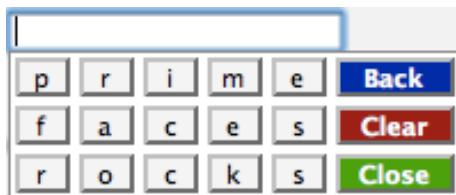
```
<p:keyboard value="#{bean.value}" layout="alphabetic"/>
```



Custom Layouts

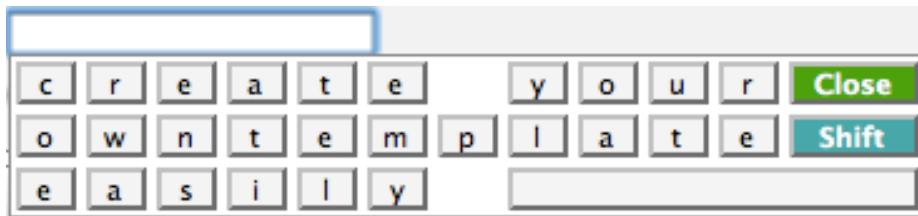
Keyboard has a very flexible layout mechanism allowing you to come up with your own layout.

```
<p:keyboard value="#{bean.value}"
    layout="custom"
    layoutTemplate="prime-back,faces-clear,rocks-close"/>
```



Another example;

```
<p:keyboard value="#{bean.value}"
    layout="custom"
    layoutTemplate="create-space-your-close,owntemplate-shift,easily-space-spacebar"/>
```



A layout template consists of built-in keys and your own keys. Following is the list of all built-in keys.

- back
- clear
- close
- shift
- spacebar
- space
- halfspace

All other text in a layout is realized as separate keys so “prime” would create 5 keys as “p” “r” “i” “m” “e”. Use dash to separate each member in layout and use commas to create a new row.

Keypad

By default keyboard displays whole keys, if you only need the numbers use the keypad mode.

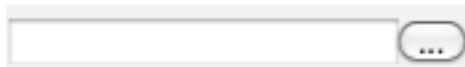
```
<p:keyboard value="#{bean.value}" />
```



ShowMode

There're a couple of different ways to display the keyboard, by default keyboard is shown once input field receives the focus. This is customized using the showMode feature which accept values ‘focus’, ‘button’, ‘both’. Keyboard below displays a button next to the input field, when the button is clicked the keyboard is shown.

```
<p:keyboard value="#{bean.value}" showMode="button"/>
```



Button can also be customized using the buttonImage and buttonImageOnly attributes.

```
<p:keyboard value="#{bean.value}" buttonImage="key.png"
    buttonImageOnly="true"/>
```



Skinning Keyboard

Skinning keyboard is achieved with CSS. Following are three different skinning examples.

Aqua

```
<p:keyboard value="#{bean.value}" styleClass="aqua"/>
```

```
#keypad-div.aqua {
    background: #6699CC;
    border: 1px solid #CCCCFF;
    -moz-border-radius: 4px;
    -webkit-border-radius: 4px;
}

.aqua .keypad-key {
    width: 20px;
    border: 1px solid #CCCCFF;
    -moz-border-radius: 4px;
    -webkit-border-radius: 4px;
    color: #FFFFFF;
    background: #99CCFF;
}

.aqua .keypad-key-down {
    background: #8c8;
}

.aqua .keypad-clear {
    letter-spacing: -3px;
}

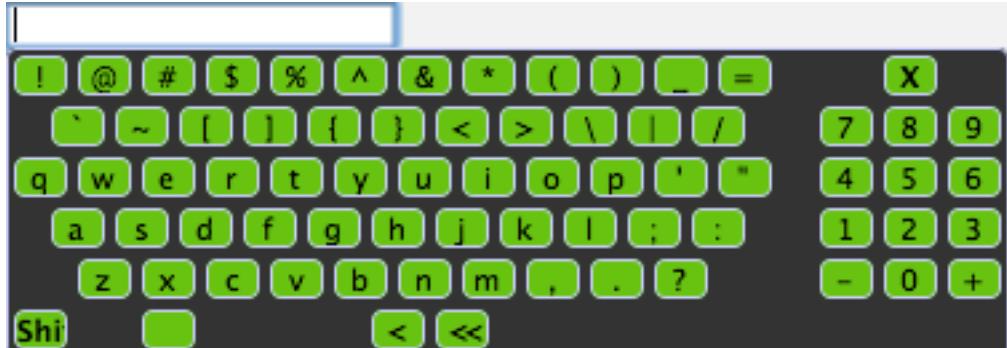
.aqua .keypad-space {
    width: 20px;
}
```



Homebrew

```
<p:keyboard value="#{bean.value}" styleClass="homebrew"/>
```

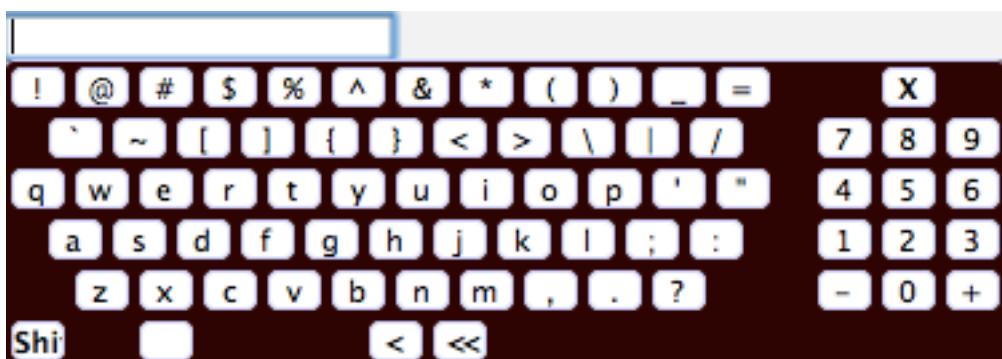
```
#keypad-div.homebrew {
    background: #333333;
    border: 1px solid #CCCCFF;
    -moz-border-radius: 4px;
    -webkit-border-radius: 4px;
}
.homebrew .keypad-key {
    width: 20px;
    border: 1px solid #CCCCFF;
    -moz-border-radius: 4px;
    -webkit-border-radius: 4px;
    color: #000;
    background: #33CC00;
}
.homebrew .keypad-key-down {
    background: #8c8;
}
.homebrew .keypad-clear {
    letter-spacing: -3px;
}
.homebrew .keypad-space {
    width: 20px;
}
```



Brownie

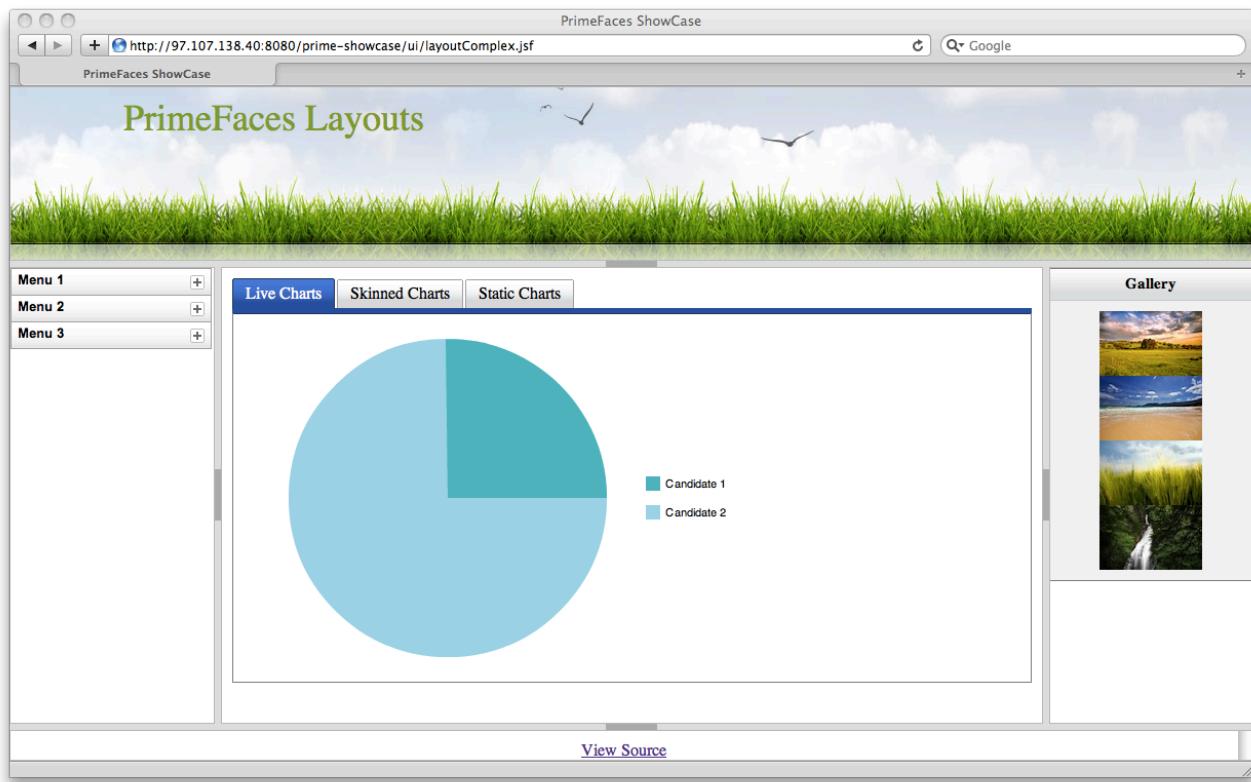
```
<p:keyboard value="#{bean.value}" styleClass="brownie"/>
```

```
#keypad-div.brownie {
    background: #330000;
    border: 1px solid #CCCCFF;
    -moz-border-radius: 4px;
    -webkit-border-radius: 4px;
}
.brownie .keypad-key {
    width: 20px;
    border: 1px solid #CCCCFF;
    -moz-border-radius: 4px;
    -webkit-border-radius: 4px;
    color: #000;
    background: #FFFFFF;
}
.brownie .keypad-key-down {
    background: #8c8;
}
.brownie .keypad-clear {
    letter-spacing: -3px;
}
.brownie .keypad-space {
    width: 20px;
}
```



3.34 Layout

Layout component features a highly customizable borderLayout model making it very easy to create complex layouts even if you're not familiar with web design.



Info

Tag	layout
Tag Class	org.primefaces.component.layout.LayoutTag
Component Class	org.primefaces.component.layout.Layout
Component Type	org.primefaces.component.Layout
Component Family	org.primefaces.component
Renderer Type	org.primefaces.component.LayoutRenderer
Renderer Class	org.primefaces.component.layout.LayoutRenderer

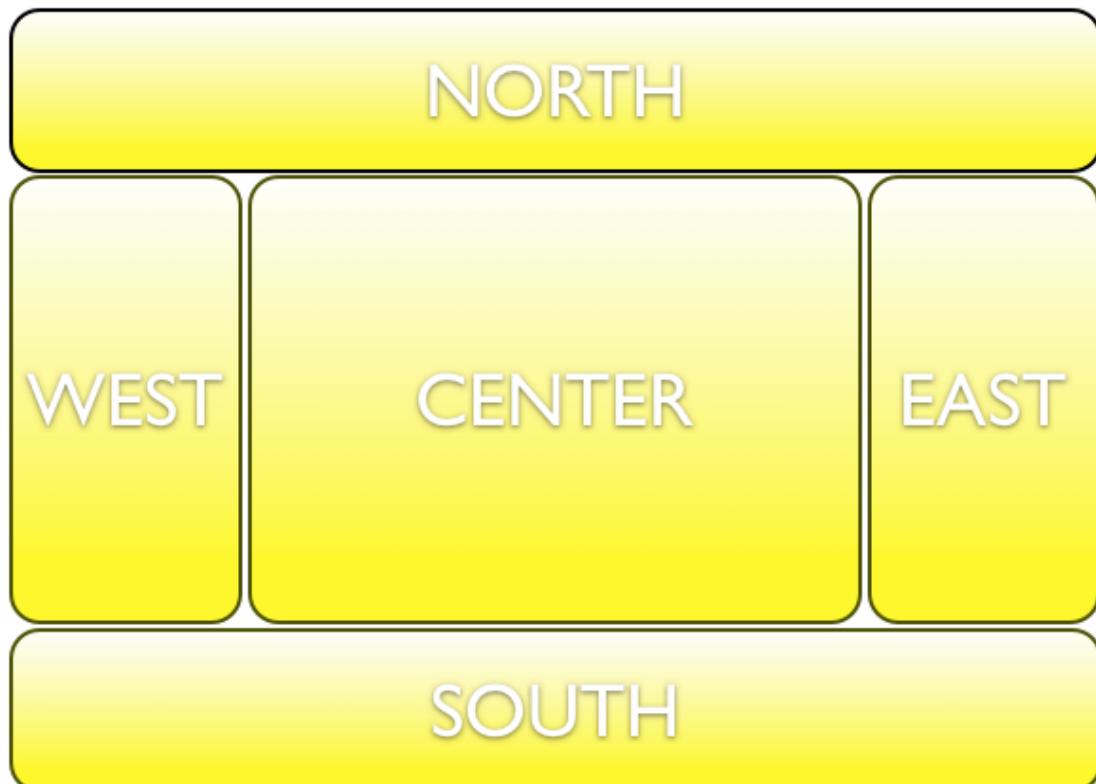
Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component

Name	Default	Type	Description
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
widgetVar	null	String	Javascript variable name of the wrapped widget
fullPage	FALSE	boolean	Specifies whether layout should span all page or not.
style	null	String	Style to apply to container element, this is only applicable to element based layouts.
styleClass	null	String	Style class to apply to container element, this is only applicable to element based layouts.

Getting started with Layout

Layout is based on a borderLayout model that consists of 5 different layout units which are north, west, center, east and south. This model is visualized in the schema below;

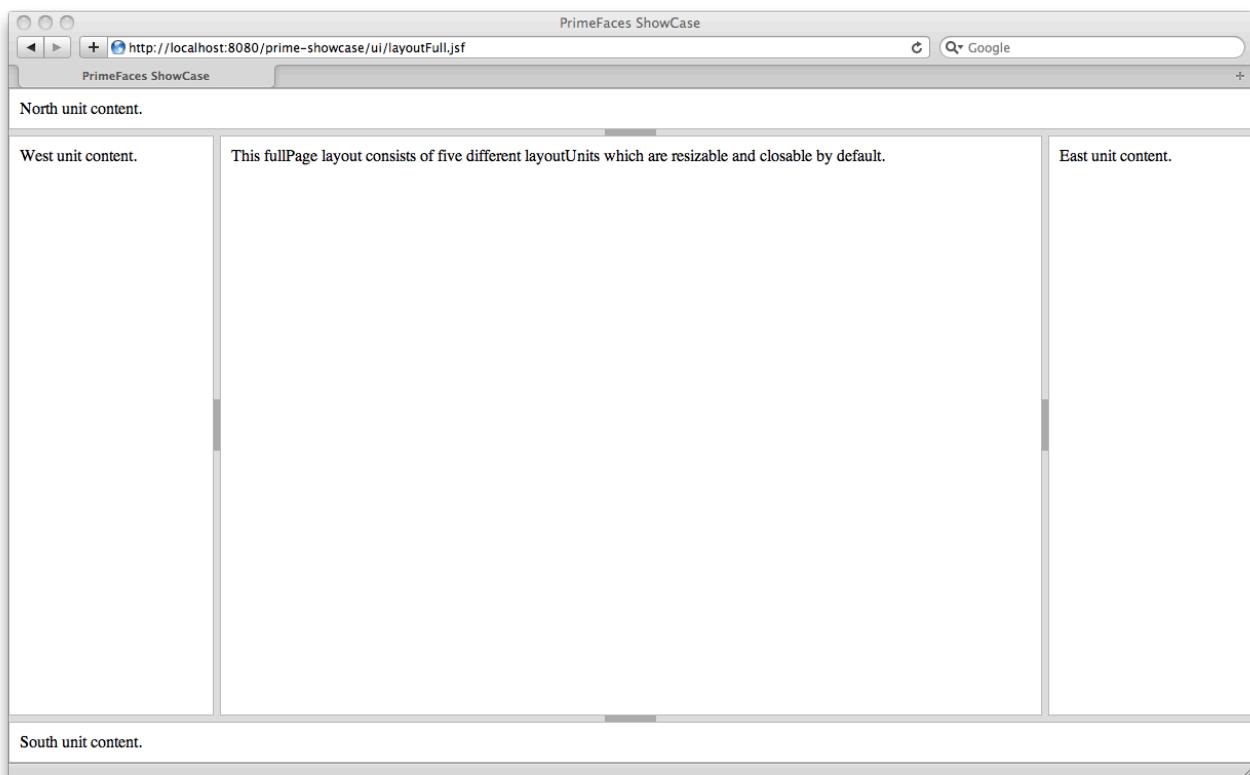


Full Page Layout

Layout has two modes, you can either use it for a full page layout or for a specific region in your page. This setting is controlled with the `fullPage` attribute which is false by default.

The regions in a layout are defined by `layoutUnits`, following is a simple full page layout with all possible units. Note that you can place any content in each layout unit.

```
<p:layout fullPage="true">
    <p:layoutUnit position="top" header="TOP" height="50">
        <h:outputText value="Top content." />
    </p:layoutUnit>
    <p:layoutUnit position="bottom" header="BOTTOM" height="100">
        <h:outputText value="Bottom content." />
    </p:layoutUnit>
    <p:layoutUnit position="left" header="LEFT" width="300">
        <h:outputText value="Left content" />
    </p:layoutUnit>
    <p:layoutUnit position="right" header="RIGHT" width="200">
        <h:outputText value="Right Content" />
    </p:layoutUnit>
    <p:layoutUnit position="center" header="CENTER">
        <h:outputText value="Center Content" />
    </p:layoutUnit>
</p:layout>
```



Layout unit position

There're five different regions for a layout unit to be placed these are;

- north
- south
- east
- west
- center

Dimensions

Except center layoutUnit, other layout units must have dimensions defined. size attributes is used to set the default of unit. For north and south units, size corresponds to height and for east and west units size means width. minSize and maxSize is to define limits of resizable units.

Element based layout

Another powerful feature of layout is that, you can use it anywhere in your page even you're not using it for the whole page. This is the default case actually so just not define fullPage attribute or set it to false.

Layout example below demonstrates the usage of a layout withing a specific page region.

```
<p:layout height="400">
    <p:layoutUnit position="top" height="50">
        <h:outputText value="Top Content" />
    </p:layoutUnit>

    <p:layoutUnit position="bottom" height="50">
        <h:outputText value="Bottom Content" />
    </p:layoutUnit>

    <p:layoutUnit position="left" width="100">
        <h:outputText value="Left Content" />
    </p:layoutUnit>

    <p:layoutUnit position="center">
        <h:outputText value="Center Content" />
    </p:layoutUnit>
</p:layout>
```

Nested Layouts

For even more complex requirements, layouts can be nested as well.

```
<p:layout fullPage="true">

    <p:layoutUnit position="north">
        <h:outputText value="Outer north unit content." />
    </p:layoutUnit>

    <p:layoutUnit position="south">
        <h:outputText value="Outer south unit content." />
    </p:layoutUnit>

    <p:layoutUnit position="west">
        <h:outputText value="Outer west unit content." />
    </p:layoutUnit>

    <p:layoutUnit position="east">
        <h:outputText value="Outer east unit content." />
    </p:layoutUnit>

    <p:layoutUnit position="center">
        <p:layout>
            <p:layoutUnit position="north">
                <h:outputText value="Middle north unit content." />
            </p:layoutUnit>

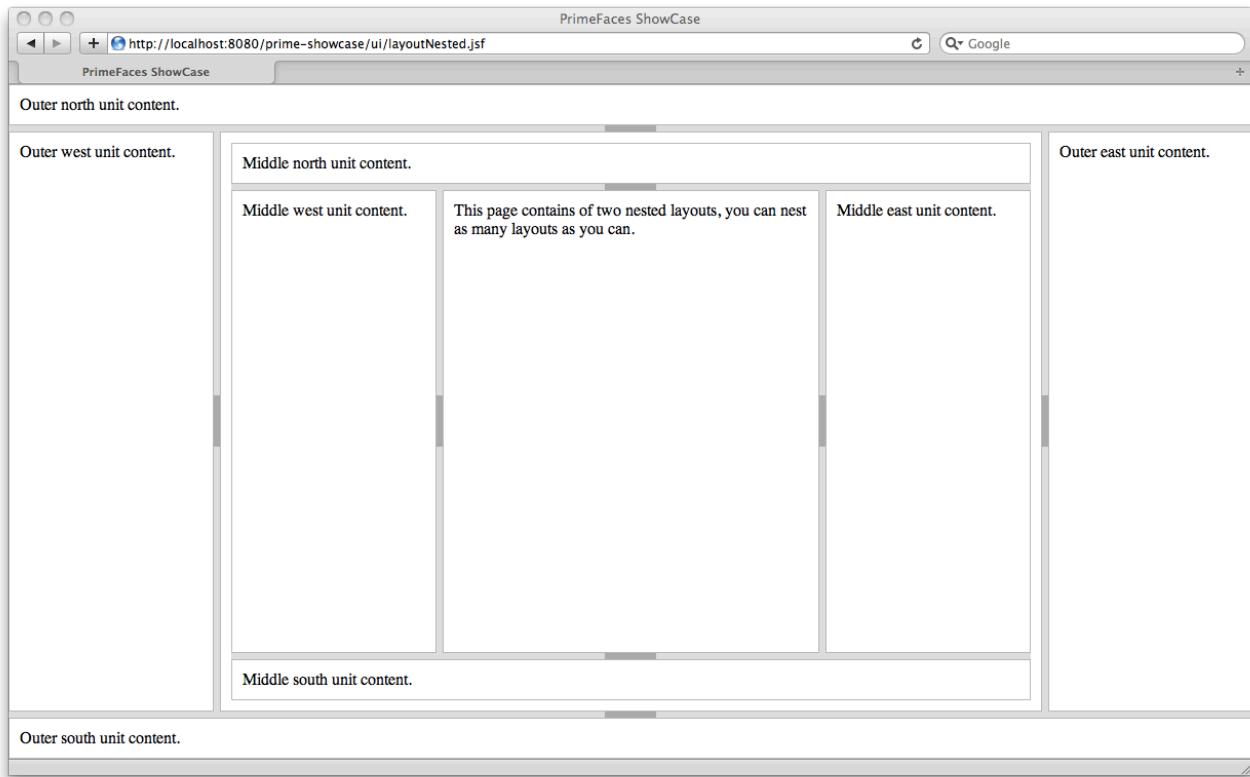
            <p:layoutUnit position="south">
                <h:outputText value="Middle south unit content." />
            </p:layoutUnit>

            <p:layoutUnit position="west">
                <h:outputText value="Middle west unit content." />
            </p:layoutUnit>

            <p:layoutUnit position="east">
                <h:outputText value="Middle east unit content." />
            </p:layoutUnit>

            <p:layoutUnit position="center">
                <h:outputText value="Middle center unit content."/>
            </p:layoutUnit>
        </p:layout>
    </p:layoutUnit>

</p:layout>
```

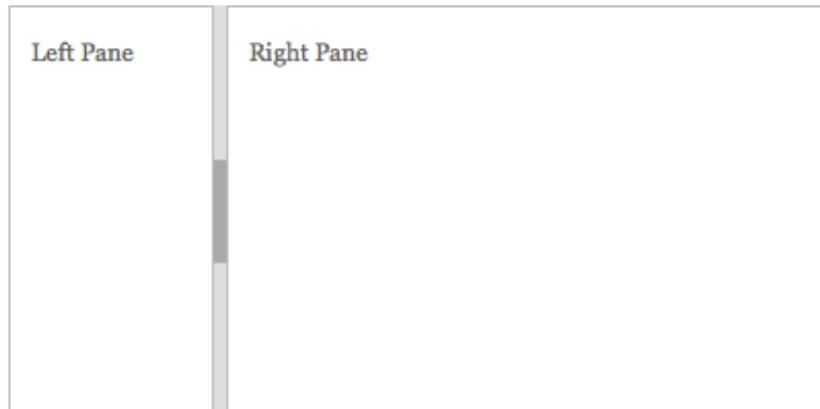


Element Based Layouts

Following layout shows how easy it is to create a horizontal split panel implementation with `p:layout`. Note that this is an example of an element based layout.

```
<p:layout style="width:400px;height:200px">
    <p:layoutUnit position="west" size="100">
        <h:outputText value="Left Pane" />
    </p:layoutUnit>

    <p:layoutUnit position="center">
        <h:outputText value="Right Pane" />
    </p:layoutUnit>
</p:layout>
```



Skinning Layout

Style Class	Applies
.pf-layout-pane	Each layout unit container
.pf-layout-pane-{location}	Location specific layout unit container
.pf-layout-resizer	Resizer element
.pf-layout-toggler	Toggler element to show/hide unit.

{location} could be ‘north’, ‘south’, ‘east’, ‘west’ and ‘center’.

3.35 LayoutUnit

LayoutUnit represents a region in the border layout model of the Layout component. Please see the layout component section for more information.

Info

Tag	<code>layoutUnit</code>
Tag Class	<code>org.primefaces.component.layout.LayoutUnitTag</code>
Component Class	<code>org.primefaces.component.layout.LayoutUnit</code>
Component Type	<code>org.primefaces.component.LayoutUnit</code>
Component Family	<code>org.primefaces.component</code>

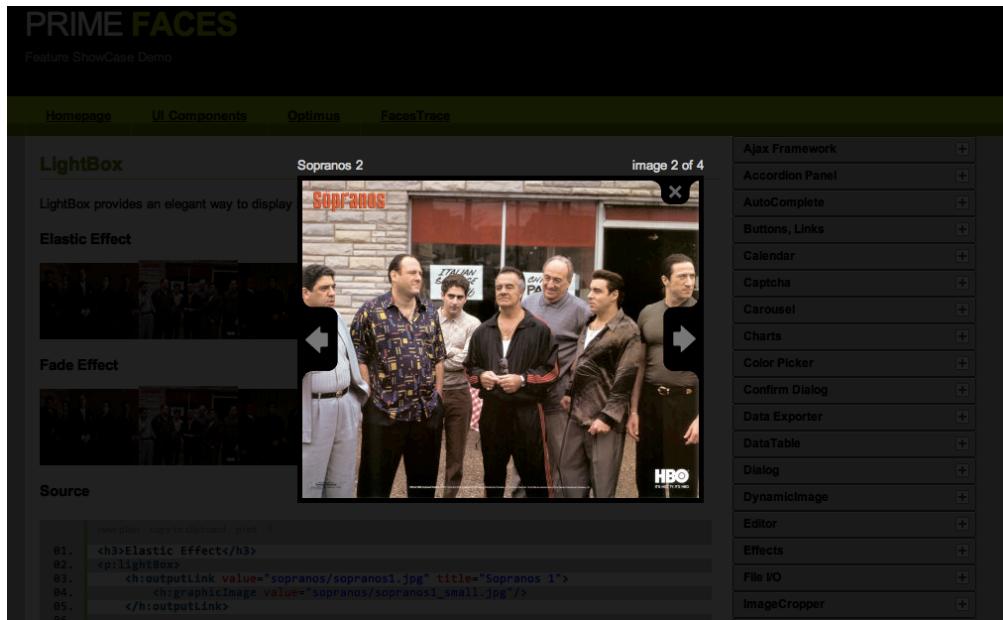
Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
<code>position</code>	null	String	Position of the unit, can be 'north', 'west', 'center', 'east', 'south'.
<code>size</code>	null	String	Size of the unit in pixels.
<code>minSize</code>	50	Integer	Minimum size of a resizable unit, 0 is unlimited.
<code>maxSize</code>	0	Integer	Maximum size of a resizable unit, 0 is unlimited.
<code>resizable</code>	TRUE	boolean	Makes the unit resizable.
<code>spacingOpen</code>	6	Integer	Spacing between adjacent units when open.
<code>spacingClosed</code>	6	Integer	Spacing between adjacent units when closed.
<code>closable</code>	TRUE	boolean	Makes unit closable.
<code>slidable</code>	TRUE	boolean	Makes unit slidable.
<code>style</code>	null	String	Style to apply to main container element.
<code>styleClass</code>	null	String	Style class to apply to main container element.

Name	Default	Type	Description
closed	FALSE	boolean	Unit is closed by default when set to true.
effect	null	String	Name of the effect of toggle animation.
effectSpeed	null	String	Speed of the effect of toggle animation.

3.36 LightBox

Lightbox features a powerful overlay that can display images, multimedia content, other JSF components and external urls.



Info

Tag	lightBox
Tag Class	org.primefaces.component.lightbox.LightBoxTag
Component Class	org.primefaces.component.lightbox.LightBox
Component Type	org.primefaces.component.LightBox
Component Family	org.primefaces.component
Renderer Type	org.primefaces.component.LightBoxRenderer
Renderer Class	org.primefaces.component.lightbox.LightBoxRenderer

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.

Name	Default	Type	Description
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
style	null	String	Style of the container element not the overlay element.
styleClass	null	String	Style class of the container element not the overlay element.
widgetVar	null	String	Javascript variable name of the client side widget
transition	elastic	String	Name of the transition effect. Valid values are 'elastic', 'fade' and 'none'.
speed	350	int	Speed of the transition effect in milliseconds.
width	null	String	Width of the overlay.
height	null	String	Height of the overlay.
iframe	FALSE	boolean	Specifies an iframe to display an external url in overlay.
opacity	0.85	double	Level of overlay opacity between 0 and 1.
visible	FALSE	boolean	Displays lightbox without requiring any user interaction by default.
slideshow	FALSE	boolean	Displays lightbox without requiring any user interaction by default.
slideshowSpeed	2500	int	Speed for slideshow in milliseconds.
slideshowStartText	null	String	Label of slideshow start text.
slideshowStopText	null	String	Label of slideshow stop text.
slideshowAuto	TRUE	boolean	Starts slideshow automatically.
currentTemplate	null	String	Text template for current image display like "1 of 3". Default is "{current} of {total}".
overlayClose	TRUE	boolean	When true clicking outside of overlay will close lightbox.

Images

The images displayed in the lightBox need to be nested as child outputLink components. Following lightBox is displayed when any of the links are clicked.

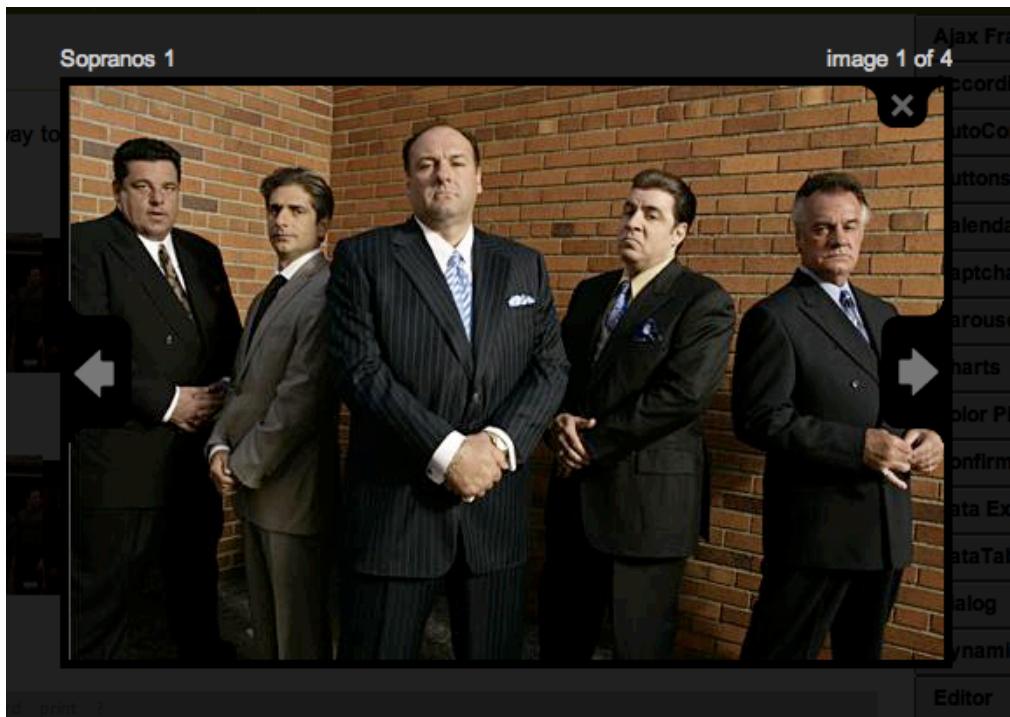
```
<p:lightbox>
    <h:outputLink value="sopranos/sopranos1.jpg" title="Sopranos 1">
        <h:graphicImage value="sopranos/sopranos1_small.jpg"/>
    </h:outputLink>

    <h:outputLink value="sopranos/sopranos2.jpg" title="Sopranos 2">
        <h:graphicImage value="sopranos/sopranos2_small.jpg" />
    </h:outputLink>

    <h:outputLink value="sopranos/sopranos3.jpg" title="Sopranos 3">
        <h:graphicImage value="sopranos/sopranos3_small.jpg"/>
    </h:outputLink>

    <h:outputLink value="sopranos/sopranos4.jpg" title="Sopranos 4">
        <h:graphicImage value="sopranos/sopranos4_small.jpg"/>
    </h:outputLink>
</p:lightbox>
```

Output of this lightbox is;

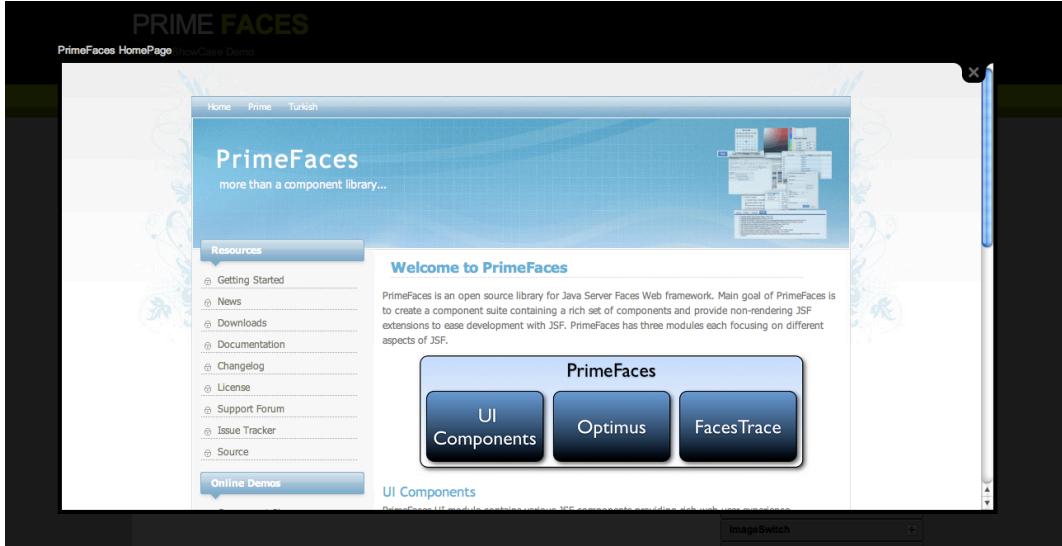


Iframe Mode

LightBox also has the ability to display iframes inside the page overlay, following lightbox displays the PrimeFaces homepage when the link inside is clicked.

```
<p:lightBox iframe="true" width="80%" height="80%">
    <h:outputLink value="http://primefaces.prime.com.tr/en"
        title="PrimeFaces HomePage">
        <h:outputText value="PrimeFaces HomePage"/>
    </h:outputLink>
</p:lightBox>
```

Clicking the outputLink will display PrimeFaces homepage within an iframe.

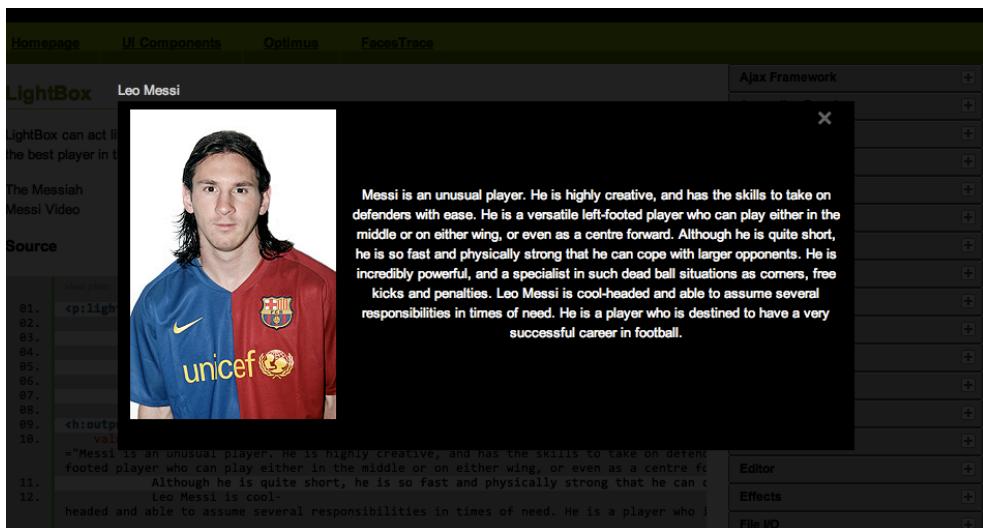


Inline Mode

Inline mode acts like a modal panel, you can display other JSF content on the page using the lightbox overlay. Simply place your overlay content in the “inline” facet. Clicking the link in the example below will display the panelGrid contents in overlay.

```
<p:lightBox width="50%" height="50%">
    <h:outputLink value="#" title="Leo Messi" >
        <h:outputText value="The Messiah"/>
    </h:outputLink>

    <f:facet name="inline">
        <h:panelGrid columns="2">
            <h:graphicImage value="barca/messi.jpg" />
        <h:outputText style="color:#FFFFFF"
            value="Messi is an unusual player....." />
        </h:panelGrid>
    </f:facet>
</p:lightBox>
```



Skinning LightBox

style and styleClass attributes effect the parent dom element containing the outputLink components. These classes do not effect the overlay. There'll be more customization options for skinning the overlay and built-in themes in future releases.

SlideShow

If you want to use lightbox images as a slideshow, turn slideshow setting to true.

```
<p:lightBox slideshow="true" slideshowSpeed="2000"
    slideshowStartText="Start" slideshowStopText="Stop">
    <h:outputLink value="sopranos/sopranos1.jpg" title="Sopranos 1">
        <h:graphicImage value="sopranos/sopranos1_small.jpg"/>
    </h:outputLink>

    <h:outputLink value="sopranos/sopranos2.jpg" title="Sopranos 2">
        <h:graphicImage value="sopranos/sopranos2_small.jpg" />
    </h:outputLink>
```

Tips and Tricks

There's a known issue with lightbox that causes to page scroll down each time an image is shown. Workaround is to change the doctype to;

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
```

3.37 LinkButton

LinkButton is a goButton implementation that is used to redirect to a URL.



Info

Tag	<code>linkButton</code>
Tag Class	<code>org.primefaces.component.linkbutton.LinkButtonTag</code>
Component Class	<code>org.primefaces.component.linkbutton.LinkButton</code>
Component Type	<code>org.primefaces.component.LinkButton</code>
Component Family	<code>org.primefaces.component</code>
Renderer Type	<code>org.primefaces.component.LinkButtonRenderer</code>
Renderer Class	<code>org.primefaces.component.linkbutton.LinkButtonRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
<code>value</code>	null	String	Label for the link button
<code>converter</code>	null	Converter/String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id
<code>widgetVar</code>	null	String	Id for the button object defined as a YUI button for to be accessed outside.
<code>href</code>	null	String	Href value for the link button used for navigating.
<code>target</code>	null	String	Html anchor target attribute, valid values are “_blank”, “_top”, “_parent”, “_self”

Name	Default	Type	Description
style	null	String	Style to be applied on the button element
styleClass	null	String	StyleClass to be applied on the button element
onblur	null	String	onblur dom event handler
onchange	null	String	onchange dom event handler
onclick	null	String	onclick dom event handler
ondblclick	null	String	ondblclick dom event handler
onfocus	null	String	onfocus dom event handler
onkeydown	null	String	onkeydown dom event handler
onkeypress	null	String	onkeypress dom event handler
onkeyup	null	String	onkeyup dom event handler
onmousedown	null	String	onmousedown dom event handler
onmousemove	null	String	onmousemove dom event handler
onmouseout	null	String	onmouseout dom event handler
onmouseover	null	String	onmouseover dom event handler
onmouseup	null	String	onmouseup dom event handler
onselect	null	String	onselect dom event handler

Getting started with LinkButton

LinkButton component requires a link(href) to navigate

```
<p:linkButton value="Barca" href="http://www.fcbarcelona.com" />
```

Skinning LinkButton

Please check button component for styling link button.

3.38 Media

Media component is used for embedding multimedia content such as videos and music to JSF views. Media renders `<object />` or `<embed />` html tags depending on the user client.

Info

Tag	<code>media</code>
Tag Class	<code>org.primefaces.component.media.MediaTag</code>
Component Class	<code>org.primefaces.component.media.Media</code>
Component Type	<code>org.primefaces.component.Media</code>
Component Family	<code>org.primefaces.component</code>
Renderer Type	<code>org.primefaces.component.MediaRenderer</code>
Renderer Class	<code>org.primefaces.component.media.MediaRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component.
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean.
<code>value</code>	null	String	URL referring to the media source to play.
<code>player</code>	null	String	Type of the player, possible values are "quicktime", "windows", "flash", "real".
<code>width</code>	null	String	Width of the player.
<code>height</code>	null	String	Height of the player.
<code>style</code>	null	String	Style of the player.
<code>styleClass</code>	null	String	StyleClass of the player.

Getting started with Media

In its simplest form media component requires a source to play, this is defined using the value attribute.

```
<p:media value="/media/ria_with_primefaces.mov" />
```

Player Types

By default, players are identified using the value extension so for instance mov files will be played by quicktime player. You can customize which player to use with the player attribute.

```
<p:media value="http://www.youtube.com/v/ABCDEFGH" player="flash"/>
```

Following is the supported players and file types.

Player	Types
windows	asx, asf, avi, wma, wmv
quicktime	aif, aiff, aac, au, bmp, gsm, mov, mid, midi, mpg, mpeg, mp4, m4a, psd, qt, qtif, qif, qt, snd, tif, tiff, wav, 3g2, 3pg
flash	flv, mp3, swf
real	ra, ram, rm, rpm, rv, smi, smil

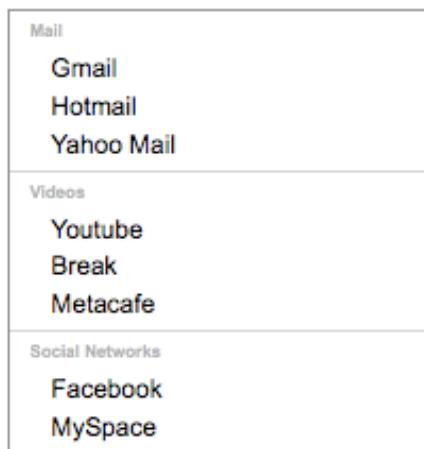
Parameters

Different proprietary players might have different configuration parameters, these can be specified using f:param tags.

```
<p:media value="/media/ria_with_primefaces.mov">
    <f:param name="param1" value="value1" />
    <f:param name="param2" value="value2" />
</p:media>
```

3.39 Menu

Menu is a navigation component with various customized modes like multi tiers, overlay and nested menus.



Info

Tag	<code>menu</code>
Tag Class	<code>org.primefaces.component.menu.MenuTag</code>
Component Class	<code>org.primefaces.component.menu.Menu</code>
Component Type	<code>org.primefaces.component.Menu</code>
Component Family	<code>org.primefaces.component</code>
Renderer Type	<code>org.primefaces.component.MenuRenderer</code>
Renderer Class	<code>org.primefaces.component.menu.MenuRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component.
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean.
<code>visible</code>	FALSE	boolean	Sets menu's visibility. Only <i>applicable</i> to dynamic positioned menus.

Name	Default	Type	Description
x	null	int	Sets the menu's left absolute coordinate. Only applicable to dynamic positioned menus.
y	null	int	Sets the menu's top absolute coordinate. Only applicable to dynamic positioned menus.
fixedCenter	FALSE	boolean	Boolean value that specifies whether the component should be automatically centered in the viewport on window scroll and resize. Only applicable to dynamic positioned menus.
constraintToViewPort	TRUE	FALSE	Boolean indicating if the Menu will try to remain inside the boundaries of the size of viewport. Only applicable to <i>dynamic</i> positioned menus.
position	static	String	Sets the way menu is placed on the page, when "static" menu is displayed in the normal flow, when set to "dynamic" menu is not on the normal flow allowing overlaying. Default value is "static".
clickToHide	TRUE	boolean	Sets the behavior when outside of the menu is clicked. Only applicable to <i>dynamic</i> positioned menus.
keepOpen	FALSE	boolean	Sets the behavior when the menu is clicked. Only applicable to <i>dynamic</i> positioned menus.
tiered	FALSE	boolean	Sets the tiered mode, when set to true menu will be rendered in different tiers.
effect	FADE	String	Sets the effect for the menu display, default value is FADE. Possible values are "FADE", "SLIDE", "NONE". Use "NONE" to disable animation at all.
effectDuration	0.25	double	Sets the effect duration in seconds.
autoSubmenuDisplay	TRUE	boolean	When set to true, submenus are displayed on mouseover of a menuitem.
showDelay	250	int	Sets the duration in milliseconds before a submenu is displayed.
hideDelay	0	int	Sets the duration in milliseconds before a menu is hidden.
submenuHideDelay	250	int	Sets the duration in milliseconds before a submenu is hidden.
context	null	String	Position of the menu.
widgetVar	null	String	Javascript variable name of the wrapped widget.

Getting started with the Menu

A menu is composed of submenus and menuitems.

```
<p:menu>
    <p:submenu title="Mail">
        <p:menuitem label="Gmail" url="http://www.google.com" />
        <p:menuitem label="Hotmail" url="http://www.hotmail.com" />
        <p:menuitem label="Yahoo Mail" url="http://mail.yahoo.com" />
    </p:submenu>

    <p:submenu title="Videos">
        <p:menuitem label="Youtube" url="http://www.youtube.com" />
        <p:menuitem label="Break" url="http://www.break.com" />
        <p:menuitem label="Metacafe" url="http://www.metacafe.com" />
    </p:submenu>

    <p:submenu title="Social Networks">
        <p:menuitem label="Facebook" url="http://www.facebook.com" />
        <p:menuitem label="MySpace" url="http://www.myspace.com" />
    </p:submenu>
</p:menu>
```

Dynamic Positioning

Menu can be positioned on a page in two ways; “static” and “dynamic”. By default it’s static meaning the menu is in normal page flow. In contrast dynamic menus is not on the normal flow of the page allowing overlaying of other elements. A dynamic menu can be positioned on the page using the x,y or fixedCenter attributes. X and Y attributes defined the top and left coordinates of the menu. Another alternative is by setting fixedCenter to true, this way menu would be positioned at the center of page.

A dynamic menu is not visible by default and a little javascript is required to hide and show the menu.

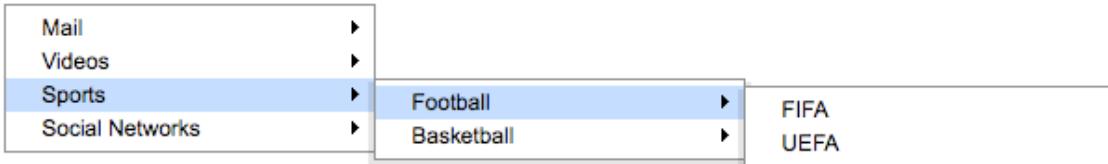
```
<p:menu position="dynamic" widgetVar="myMenu">
    ...submenus and menuitems
</p:menu>

<a href="#" onclick="myMenu.show()">Show</a>
<a href="#" onclick="myMenu.hide()">Hide</a>
```

MultiTiered Menus

By default each submenu is displayed at a single tier, menu also supports nested submenus, *tiered* attribute needs to set to true to enable this features.

```
<p:menu tiered="true">  
    ...submenus and menuitems  
</p:menu>
```



Effects

Menu has a built-in animation to display when displaying hiding itself and it's submenus. This animation is customizable using attributes like effect, effectDuration, showDelay and more. Full list is described at the attributes table.

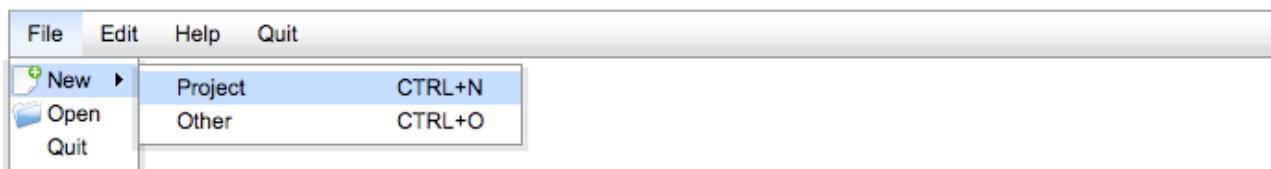
Skinning

Full list of CSS Selectors

<http://developer.yahoo.com/yui/menu/#skinref>

3.40 Menubar

Menubar is similar to the menu and provides a horizontal navigation component.



Info

Tag	<code>menubar</code>
Tag Class	<code>org.primefaces.component.menubar.MenuBarTag</code>
Component Class	<code>org.primefaces.component.menubar.MenuBar</code>
Component Type	<code>org.primefaces.component.MenuBar</code>
Component Family	<code>org.primefaces.component</code>
Renderer Type	<code>org.primefaces.component.MenuBarRenderer</code>
Renderer Class	<code>org.primefaces.component.menubar.MenuBarRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component.
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean.
<code>effect</code>	FADE	String	Sets the effect for the menu display, default value is FADE. Possible values are "FADE", "SLIDE", "NONE". Use "NONE" to disable animation at all.
<code>effectDuration</code>	0.25	double	Sets the effect duration in seconds.
<code>autoSubmenuDisplay</code>	FALSE	boolean	When set to true, submenus are displayed on mouseover of a menuitem.
<code>widgetVar</code>	null	String	Javascript variable name of the wrapped widget.

Getting started with Menubar

Just like the menu, menubar requires submenus and menuitems as child components to compose the menubar.

```
<p:menubar>
    <p:submenu label="Mail">
        <p:menuitem label="Gmail" url="http://www.google.com" />
        <p:menuitem label="Hotmail" url="http://www.hotmail.com" />
        <p:menuitem label="Yahoo Mail" url="http://mail.yahoo.com" />
    </p:submenu>
    <p:submenu label="Videos">
        <p:menuitem label="Youtube" url="http://www.youtube.com" />
        <p:menuitem label="Break" url="http://www.break.com" />
    </p:submenu>
</p:menubar>
```

Nested Menus

To create a menubar with a higher depth, nest submenus in parent submenus.

```
<p:menubar>
    <p:submenu label="File">
        <p:submenu label="New">
            <p:menuitem label="Project" url="#" />
            <p:menuitem label="Other" url="#" />
        </p:submenu>
        <p:menuitem label="Open" url="#" /></p:menuitem>
        <p:menuitem label="Quit" url="#" /></p:menuitem>
    </p:submenu>

    <p:submenu label="Edit">
        <p:menuitem label="Undo" url="#" /></p:menuitem>
        <p:menuitem label="Redo" url="#" /></p:menuitem>
    </p:submenu>

    <p:submenu label="Help">
        <p:menuitem label="Contents" url="#" />
        <p:submenu label="Search">
            <p:submenu label="Text">
                <p:menuitem label="Workspace" url="#" />
            </p:submenu>
            <p:menuitem label="File" url="#" />
        </p:submenu>
    </p:submenu>
    <p:submenu label="Quit" url="#" />
</p:menubar>
```

Effects

Menu has a built-in animation to display when displaying hiding itself and it's submenus. This animation is customizable using attributes like effect, effectDuration, showDelay and more. Full list is described at the attributes table.

Skinning

Menubar is based on the YUI menubar which is highly customizable using CSS selectors.

```
.yui-skin-sam .yuimenubar ul {
    background: url(..../design/nav.gif);
}
.yui-skin-sam .yuimenuitemlabel {
    background: #FFFFCC;
}
.yui-skin-sam .yuimenubaritemlabel {
    background: url(..../design/nav.gif);color: #333333;
}
```



Full list of CSS Selectors is available at:

<http://developer.yahoo.com/yui/menu/#skinref>

Styling attributes

Well defined CSS selectors of menubar are perfect for global skinning of menus in your application, however PrimeFaces extension style attributes also provide a way to style a particular menubar.

Icon of a menuitem

Use style or styleClass attributes to assign an icon to a particular menuitem. Following menuitem will have an undo icon displayed on the left.

```
.undo {
    background: url(..../images/undo.png) no-repeat 2%;
}
```

```
<p:menuItem label="undo" styleClass="undo" />
```

Similarly submenu labels' can have icons as well using the same approach. PrimeFaces showcase application demonstrates a customized menubar with icons, helpTexts that you can use as a reference.

3.41 MenuItem

MenuItem is nested in a submenu component and represents a navigation item.

Info

Tag	<code>menuItem</code>
Tag Class	<code>org.primefaces.component.menuitem.MenuItemTag</code>
Component Class	<code>org.primefaces.component.menuitem.MenuItem</code>
Component Type	<code>org.primefaces.component.MenuItem</code>
Component Family	<code>org.primefaces.component</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component.
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean.
<code>label</code>	null	String	Label to be displayed.
<code>url</code>	null	String	URL that will be navigated to when the menuitem is clicked.
<code>target</code>	null	String	Target element of the menuitem's anchor element
<code>helpText</code>	null	String	Help text of the menuitem, can be used for keyboard shortcuts.
<code>onClick</code>	null	String	Javascript onclick event.
<code>style</code>	null	String	Style of the menuitem label.
<code>styleClass</code>	null	String	StyleClass of the menuitem label.

Getting started with MenuItem

Please see Menu section to find out how menuitem is used with the menu.

3.42 Message

Message is a pre-skinned extended version of the standard JSF message component.



Info

Tag	message
Tag Class	org.primefaces.component.message.MessageTag
Component Class	org.primefaces.component.message.Message
Component Type	org.primefaces.component.Message
Component Family	org.primefaces.component
Renderer Type	org.primefaces.component.MessageRenderer
Renderer Class	org.primefaces.component.message.MessageRenderer

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component.
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean.
showSummary	FALSE	boolean	Specifies if the summary of the FacesMessage should be displayed.
showDetail	TRUE	boolean	Specifies if the detail of the FacesMessage should be displayed.
for	null	String	Id of the component whose messages to display.

Getting started with Message

Message usage is exactly same as standard message.

```
<h:inputText id="txt" value="#{bean.text}" />
<p:message for="txt" />
```

Skinning Message

Full list of CSS selectors of message is as follows;

Style Class	Applies
pf-message-{severity}	Container element of the message
pf-message-{severity}-summary	Summary text
pf-message-{severity}-info	Detail text

{severity} can be 'info', 'error', 'warn' and error.

3.43 Messages

Messages is a pre-skinned extended version of the standard JSF messages component.



Sample info message PrimeFaces rocks!



Sample warn message Watch out for PrimeFaces!



Sample error message PrimeFaces makes no mistakes



Sample fatal message Fatal Error in System

Info

Tag	messages
Tag Class	org.primefaces.component.messages.MessagesTag
Component Class	org.primefaces.component.messages.Messages
Component Type	org.primefaces.component.Messages
Component Family	org.primefaces.component
Renderer Type	org.primefaces.component.MessagesRenderer
Renderer Class	org.primefaces.component.messages.MessagesRenderer

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component.
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean.
showSummary	FALSE	boolean	Specifies if the summary of the FacesMessages should be displayed.
showDetail	TRUE	boolean	Specifies if the detail of the FacesMessages should be displayed.

Name	Default	Type	Description
globalOnly	FALSE	String	When true, only facesmessages with no clientIds are displayed.

Getting started with Message

Message usage is exactly same as standard messages.

```
<p:messages />
```

Skinning Message

Full list of CSS selectors of message is as follows;

Style Class	Applies
pf-messages-{severity}	Container element of the message
pf-messages-{severity}-summary	Summary text
pf-messages-{severity}-info	Detail text
pf-messages-{severity}-icon	Icon of the message.

{severity} can be ‘info’, ‘error’, ‘warn’ and error.

3.44 OutputPanel

OutputPanel is a display only element that's useful in various cases such as adding placeholders to a page.

Info

Tag	outputPanel
Tag Class	org.primefaces.component.outputpanel.OutputPanelTag
Component Class	org.primefaces.component.outputpanel.OutputPanel
Component Type	org.primefaces.component.OutputPanel
Component Family	org.primefaces.component
Renderer Type	org.primefaces.component.OutputPanelRenderer
Renderer Class	org.primefaces.component.output.OutputPanelRenderer

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
style	null	String	Style of the html container element
styleClass	null	String	StyleClass of the html container element

AjaxRendered

Due to the nature of ajax, it is much simpler to update an existing element on page rather than inserting a new element to the dom. When a JSF component is not rendered, no markup is rendered so for components with conditional rendering regular PPR mechanism may not work since the markup to update on page does not exist. OutputPanel is useful in this case.

Suppose the rendered condition on bean is false when page is loaded initially and search method on bean sets the condition to be true meaning datatable will be rendered after a page submit. The problem is although partial output is generated, the markup on page cannot be updated since it doesn't exist.

```
<p:dataTable id="tbl" rendered="#{bean.condition}" ...>
    //columns
</p:dataTable>

<p:commandButton update="tbl" actionListener="#{bean.search}" />
```

Solution is to use the outputPanel as a placeHolder.

```
<p:outputPanel id="out">
    <p:dataTable id="tbl" rendered="#{bean.condition}" ...>
        //columns
    </p:dataTable>
</p:outputPanel>

<p:commandButton update="out" actionListeler="#{bean.list}" />
```

Note that you won't need an outputPanel if commandButton has no update attribute specified, in this case parent form will be updated partially implicitly making an outputPanel use obsolete.

Skinning OutputPanel

style and styleClass attributes are used to skin the outputPanel which in turn renders a simple span element. Following outputPanel displays a block container which is also used in the drag&drop example to specify a droppable area.

```
.slot {
    background:#FF9900;
    width:64px; height:96px;
    display:block;
}
```

```
<p:outputPanel styleClass="slot"><p:outputPanel>
```



3.45 Panel

Grouping content with a header is a common requirement in an application and panel component aims to simplify this use case. Panel is also easy to skin and provides a toggle feature for its contents.

About Barca
People behind PrimeFaces are hardcore F.C. Barcelona fans and many examples in the demo applications reflect this. Despite of this fact PrimeFaces library is open source and free to use, that means Real Madrid fans are also welcomed to use the library :)

Info

Tag	panel
Tag Class	org.primefaces.component.panel.PanelTag
Component Class	org.primefaces.component.panel.Panel
Component Type	org.primefaces.component.Panel
Component Family	org.primefaces.component
Renderer Type	org.primefaces.component.PanelRenderer
Renderer Class	org.primefaces.component.panel.PanelRenderer

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
header	null	String	Header text
footer	null	String	Footer text
toggleable	FALSE	boolean	Makes panel toggleable, places an icon for user interaction
toggleSpeed	1000	int	Speed of toggling in milliseconds
style	null	String	Style of the panel

Name	Default	Type	Description
styleClass	null	String	Style class of the panel
collapsed	FALSE	boolean	Renders a toggleable panel as collapsed.

Getting started with Panel

Panel is a grouping component and placed as a parent of it's content.

```
<p:panel header="Header Text">
    Child components here...
</p:panel>
```

Header and Footer

Header and Footer texts can be provided by *header* and *footer* attributes.

```
<p:panel header="Header Text" footer="Footer Text">
    Child components here...
</p:panel>
```

Toggle Panel

Panel contents can be toggled with a slide effect using the toggleable feature. Toggling is turned off by default and toggleable needs to be set to true to enable it.

```
<p:panel header="Header Text" toggleable="true">
    Child components here...
</p:panel>
```

A toggleable panel looks like the following, see the icon on the top right corner.



When toggled panel contents will slide up to the header part. Currently toggling happens with a slide effect and we plan to add more built-in effect in the future .A toggle panel will look like below.

About Barca

By default toggling happens in 1000 milliseconds, this can be tuned by the *toggleSpeed* attribute.

Skinning Panel

There are two ways to change the look and feel of a panel. One way is to use the global skinning selectors and other way is to apply styles per component. Following table lists the skinning selectors for panel.

Class	Applies
.pf-panel	Main panel container element
.pf-panel-hd	Header container
.pf-panel-bd	Content container
.pf-panel-ft	Footer container
.pf-panel-toggler-expanded	Expanded toggler
.pf-panel-toggler-collapsed	Collapsed toggler

Basically, container elements are simple division elements. Example below uses these selectors to change the look of the panel.

```
.pf-panel, .pf-panel-hd, .pf-panel-bd, .pf-panel-ft {
    border-color:#99FF00;
}
.pf-panel-hd {
    background: url(..../images/head_green.png);
}
.pf-panel-bd {
    background: #FFFFCC;
}
.pf-panel-toggler-expanded, .pf-panel-toggler-collapsed{
    top:3px;
    width:16px;
    height:16px;
}
.pf-panel-toggler-expanded {
    background: url(..../images/toggle_green_expanded.png) no-repeat;
}
.pf-panel-toggler-collapsed {
    background: url(..../images/toggle_greenCollapsed.png) no-repeat;
}
```

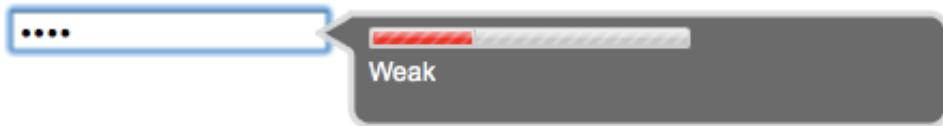
About Barca

People behind PrimeFaces are hardcore F.C. Barcelona fans and many examples in the demo applications reflect this. Despite of this fact PrimeFaces library is open source and free to use, that means Real Madrid fans are also welcomed to use the library :)

style and styleClass attributes apply to the main container element like *.pf-panel*.

3.46 Password Strength

Password Strength provides a visual feedback regarding password complexity.



Info

Tag	<code>password</code>
Tag Class	<code>org.primefaces.component.password.PasswordTag</code>
Component Class	<code>org.primefaces.component.password.Password</code>
Component Type	<code>org.primefaces.component.Password</code>
Component Family	<code>org.primefaces.component</code>
Renderer Type	<code>org.primefaces.component.PasswordRenderer</code>
Renderer Class	<code>org.primefaces.component.password.PasswordRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
<code>value</code>	null	Object	Value of the component than can be either an EL expression of a literal text
<code>converter</code>	null	Converter/ String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id

Name	Default	Type	Description
immediate	FALSE	boolean	Boolean value that specifies the lifecycle phase the valueChangeEvents should be processed, when true the events will be fired at "apply request values", if immediate is set to false, valueChange Events are fired in "process validations" phase
required	FALSE	boolean	Marks component as required
validator	null	MethodBinding	A method binding expression that refers to a method validationg the input
valueChangeListener	null	ValueChangeListene	A method binding expression that refers to a method for handling a valuchangeevent
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.
accesskey	null	String	Html accesskey attribute
alt	null	String	Html alt attribute
dir	null	String	Html dir attribute
disabled	FALSE	Boolean	Html disabled attribute
lang	null	String	Html lang attribute
maxlength	null	Integer	Html maxlength attribute
onblur	null	String	Html onblur attribute
onchange	null	String	Html onchange attribute
onclick	null	String	Html onclick attribute
ondblclick	null	String	Html ondblclick attribute
onfocus	null	String	Html onfocus attribute
onkeydown	null	String	Html onkeydown attribute
onkeypress	null	String	Html onkeypress attribute
onkeyup	null	String	Html onkeyup attribute
onmousedown	null	String	Html onmousedown attribute
onmousemove	null	String	Html onmousemove attribute

Name	Default	Type	Description
onmouseout	null	String	Html onmouseout attribute
onmouseover	null	String	Html onmouseover attribute
onmouseup	null	String	Html onmouseup attribute
readonly	FALSE	Boolean	Html readonly attribute
size	null	Integer	Html size attribute
style	null	String	Html style attribute
styleClass	null	String	Html styleClass attribute
tabindex	null	Integer	Html tabindex attribute
title	null	String	Html title attribute
minLength	8	Integer	Minimum length of a strong password
inline	FALSE	boolean	Displays feedback inline rather than using a popup.
promptLabel	Please enter a password	String	Label of prompt.
level	1	Integer	Level of security.
weakLabel	Weak	String	Label of weak password.
goodLabel	Good	String	Label of good password.
strongLabel	String	String	Label of strong password.
onshow	null	String	Javascript event handler to be executed when password strength indicator is shown.
onhide	null	String	Javascript event handler to be executed when password strength indicator is hidden.
widgetVar	null	String	Javascript variable name of the client side Password strength object.

Getting Started with Password

Password is an input component and used just like a standard input text.

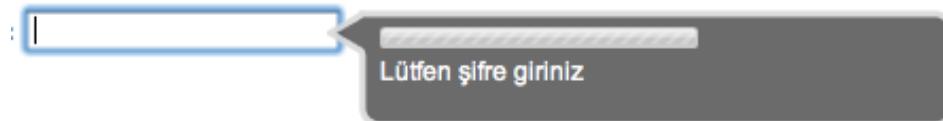
```
<p:password value="#{mybean.password}" />
```

```
public class MyBean {  
  
    private String password;  
  
    public String getPassword() { return password; }  
    public void setPassword(String password) { this.password = password; }  
}
```

I18N

Although all labels are in English by default, you can provide custom labels as well. Following password gives feedback in Turkish.

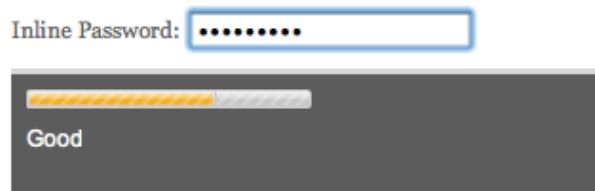
```
<p:password value="#{mybean.password}" promptLabel="Lütfen şifre giriniz"  
weakLabel="Zayıf" goodLabel="Orta seviye" strongLabel="Güçlü" />
```



Inline Strength Indicator

By default strength indicator is shown in an overlay, if you prefer an inline indicator just enable inline mode.

```
<p:password value="#{mybean.password}" inline="true"/>
```



Custom Animations

Using onshow and onhide callbacks, you can create your own animation as well.

```
<p:password value="#{mybean.password}" inline="true"  
onshow="fadein" onhide="fadeout"/>
```

This examples uses jQuery api for fadeIn and fadeOut effects. Each callback takes two parameters; input and container. input is the actual input element of password and container is the strength indicator element.

```
<script type="text/javascript">
    function fadein(input, container) {
        container.fadeIn("slow");
    }

    function fadeout(input, container) {
        container.fadeOut("slow");
    }
</script>
```

Skinning Password

Skinning selectors for password is as follows;

Name	Applies
.jpassword	Container element of strength indicator.
.jpassword-meter	Visual bar of strength indicator.
.jpassword-info	Feedback text of strength indicator.

3.47 PickList

PickList is used for transferring data between two different collections.



Info

Tag	<code>pickList</code>
Tag Class	<code>org.primefaces.component.picklist.PanelTag</code>
Component Class	<code>org.primefaces.component.picklist.Panel</code>
Component Type	<code>org.primefaces.component.PickList</code>
Component Family	<code>org.primefaces.component</code>
Renderer Type	<code>org.primefaces.component.PickListRenderer</code>
Renderer Class	<code>org.primefaces.component.picklist.PickListRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
<code>value</code>	null	Object	Value of the component than can be either an EL expression or a literal text
<code>converter</code>	null	Converter/String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id

Name	Default	Type	Description
immediate	FALSE	boolean	Boolean value that specifies the lifecycle phase the valueChangeEvents should be processed, when true the events will be fired at "apply request values", if immediate is set to false, valueChange Events are fired in "process validations" phase
required	FALSE	boolean	Marks component as required
validator	null	MethodBinding	A method binding expression that refers to a method validationg the input
valueChangeListener	null	ValueChangeListener	A method binding expression that refers to a method for handling a valuchangeevent
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.
var	null	String	Name of the iterator.
itemLabel	null	String	Label of an item.
itemValue	null	Object	Value of an item.
style	null	String	Style of the main container.
styleClass	null	String	Style class of the main container.
widgetVar	null	String	Javascript variable name of the client side PickList object.

Getting started with PickList

You need to create custom model called org.primefaces.model.picklist.DualListModel to use PickList. As the name suggests it consists of two lists, one is the source list and the other is the target. As the first example we'll create a DualListModel that contains basic Strings.

```

public class PickListBean {

    private DualListModel<String> cities;

    public PickListBean() {
        List<String> source = new ArrayList<String>();
        List<String> target = new ArrayList<String>();

        citiesSource.add("Istanbul");
        citiesSource.add("Ankara");
        citiesSource.add("Izmir");
        citiesSource.add("Antalya");
        citiesSource.add("Bursa");

        cities = new DualListModel<String>(citiesSource, citiesTarget);
    }

    public DualListModel<String> getCities() {
        return cities;
    }

    public void setCities(DualListModel<String> cities) {
        this.cities = cities;
    }
}

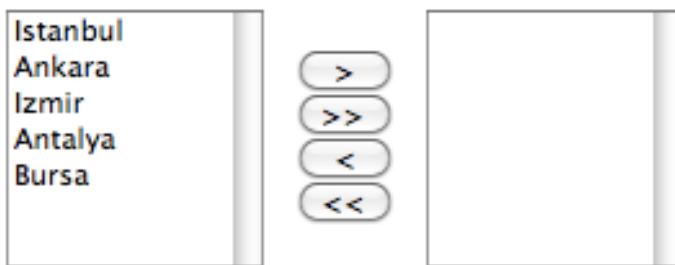
```

And bind the cities dual list to the picklist;

```

<p:pickList value="#{pickListBean.cities}" var="city"
            itemLabel="#{city}" itemValue="#{city}">

```



When you submit the form containing the pickList, the data model will be populated with the new values and you can access these values with DualListModel.getSource() and DualListModel.getTarget() api.

Complex Pojos

Most of the time you would deal with complex pojos rather than primitive types like String. This use case is no different except the addition of a converter. Following pickList displays a list of players(name, age ...).

```

public class PickListBean {

    private DualListModel<Player> players;

    public PickListBean() {
        //Players
        List<Player> source = new ArrayList<Player>();
        List<Player> target = new ArrayList<Player>();

        source.add(new Player("Messi", 10));
        source.add(new Player("Ibrahimovic", 9));
        source.add(new Player("Henry", 14));
        source.add(new Player("Iniesta", 8));
        source.add(new Player("Xavi", 6));
        source.add(new Player("Puyol", 5));

        players = new DualListModel<Player>(source, target);
    }

    public DualListModel<Player> getPlayers() {
        return players;
    }
    public void setPlayers(DualListModel<Player> players) {
        this.players = players;
    }
}

```

```

<p:pickList value="#{pickListBean.players}" var="player"
            itemLabel="#{player.name}" itemValue="#{player}" converter="player">

```

Customizing Controls

PickList is a composite component and as other PrimeFaces composite components, pickList provides a customizable UI. Using the facet based approach you can customize which controls would be displayed and how.

```

<p:pickList value="#{pickListBean.players}" var="player"
            itemLabel="#{player.name}" itemValue="#{player}" converter="player">

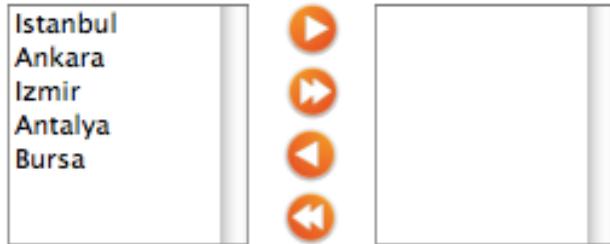
    <f:facet name="add">
        <p:graphicImage value="/images/picklist/add.png"/>
    </f:facet>
    <f:facet name="addAll">
        <p:graphicImage value="/images/picklist/addall.png"/>
    </f:facet>

```

```

<f:facet name="remove">
    <p:graphicImage value="/images/picklist/remove.png"/>
</f:facet>
<f:facet name="removeAll">
    <p:graphicImage value="/images/picklist/removeall.png"/>
</f:facet>
</p:pickList>

```



Skinning PickList

In addition to the customized controls, there're a couple of css selectors applying to picklist .

Class	Applies
.pf-picklist-source	Source listbox
.pf-picklist-target	Target listbox
.pf-picklist-control	Container for a picklist control (add, remove eg.)

PickList is located inside an html table container element which can be styled using style-styleClass attributes.

3.48 Poll

Poll is an ajax component that has the ability to send periodical ajax requests and execute actionlisteners on JSF backing beans.

Info

Tag	poll
Tag Class	<code>org.primefaces.component.poll.PollTag</code>
Component Class	<code>org.primefaces.component.poll.Poll</code>
Component Type	<code>org.primefaces.component.Poll</code>
Component Family	<code>org.primefaces.component</code>
Renderer Type	<code>org.primefaces.component.PollRenderer</code>
Renderer Class	<code>org.primefaces.component.poll.PollRenderer</code>

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component.
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean.
interval	2	Integer	Interval in seconds to do periodic ajax requests.
update	null	String	Client side id(s) of the component(s) to be updated to be updated after asyn partial submit request.
action	null	<code>javax.el.MethodExpression</code>	A method expression that'd be processed in the partial request caused by uiajax.
actionListener	null	<code>javax.faces.event.ActionListener</code>	An actionlistener that'd be processed in the partial request caused by uiajax.
immediate	FALSE	boolean	Boolean value that determines the phaseld, when true actions are processed at apply_request_values, when false at invoke_application phase.

Name	Default	Type	Description
partialSubmit	FALSE	boolean	Flag for partial process, when set to true, parent form will be submitted but only the component the ajax event is attached to will be processed by JSF
onstart	null	String	Javascript code to execute before ajax request is begins.
oncomplete	null	String	Javascript code to execute after ajax request is completed.
widgetVar	null	String	Javascript variable name of the poll object.

Getting started with Poll

Poll below invokes increment method on CounterBean every 2 seconds and txt_count is updated with the new value of the count variable. Note that poll must be nested inside a form.

```
<h:form prependId="false">
    <h:outputText id="txt_count" value="#{counterBean.count}" />

    <p:poll actionListener="#{counterBean.increment}"
        update="txt_count" />
</h:form>
```

```
public class CounterBean {

    private int count;

    public int getCount() {return count;}
    public void setCount(int count) {this.count = count;}

    public void increment(ActionEvent actionEvent) {
        count++;
    }
}
```

Tuning timing

By default the periodic interval is 2 seconds, this can be changed with the interval attribute. Following poll works per 5 seconds.

```
<h:outputText id="txt" value="#{counterBean.count}" />  
<p:poll interval="5" actionListener="#{counterBean.increment}"  
update="txt" />
```

Start and Stop

Poll can be started manually, handy widgetVar attribute is once again comes for help.

```
<h:form prependId="false">  
    <h:outputText id="txt_count" value="#{counterBean.count}" />  
  
    <p:poll interval="5" actionListener="#{counterBean.increment}"  
           update="txt_count" widgetVar="myPoll"/>  
  
    <a href="#" onclick="myPoll.start();">Start</a>  
    <a href="#" onclick="myPoll.stop();">Stop</a>  
  
</h:form>
```

3.49 Printer

Printer allows sending a specific JSF component to the printer, not the whole page.

Info

Tag	<code>printer</code>
Tag Class	<code>org.primefaces.component.printer.PrinterTag</code>
Component Class	<code>org.primefaces.component.printer.Printer</code>
Component Type	<code>org.primefaces.component.Printer</code>
Component Family	<code>org.primefaces.component</code>
Renderer Type	<code>org.primefaces.component.PrinterRenderer</code>
Renderer Class	<code>org.primefaces.component.printer.PrinterRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
<code>target</code>	null	String	Server side id of a JSF component to print.

Getting started with the Printer

Printer is attached to any action component like a button or a link. Printer below allows printing only the outputText, not the whole page.

```
<h:commandButton id="btn" value="Print">
    <p:printer target="output" />
</h:commandButton>

<h:outputText id="output" value="PrimeFaces Rocks!" />
```

Following printer prints an image on the page.

```
<h:outputLink id="lnk" value="#">  
    <p:printer target="image" />  
    <h:outputText value="Print Image" />  
</h:outputLink>  
  
<p:graphicImage id="image" value="/images/nature1.jpg" />
```

3.50 Push

Push component creates an agent that creates a channel between the server and the client.

Info

Tag	<code>push</code>
Tag Class	<code>org.primefaces.component.push.PushTag</code>
Component Class	<code>org.primefaces.component.push.Push</code>
Component Type	<code>org.primefaces.component.Push</code>
Component Family	<code>org.primefaces.component</code>
Renderer Type	<code>org.primefaces.component.PushRenderer</code>
Renderer Class	<code>org.primefaces.component.push.PushRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
<code>channel</code>	null	Object	Unique channel name of the connection between subscriber and the server.
<code>onpublish</code>	null	Object	Javascript event handler that is process when the server publishes data.

Getting started with the Push

See chapter 6, “Ajax Push/Comet” for detailed information.

3.51 Rating

Rating component features a star based rating system. Rating can be used as a plain input component or with ajax RateListeners.



Info

Tag	<code>rating</code>
Tag Class	<code>org.primefaces.component.rating.RatingTag</code>
Component Class	<code>org.primefaces.component.rating.Rating</code>
Component Type	<code>org.primefaces.component.Rating</code>
Component Family	<code>org.primefaces.component</code>
Renderer Type	<code>org.primefaces.component.RatingRenderer</code>
Renderer Class	<code>org.primefaces.component.rating.RatingRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
<code>value</code>	null	Object	Value of the component than can be either an EL expression of a literal text
<code>converter</code>	null	Converter/String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id

Name	Default	Type	Description
immediate	FALSE	boolean	Boolean value that specifies the lifecycle phase the valueChangeEvents should be processed, when true the events will be fired at "apply request values", if immediate is set to false, valueChange Events are fired in "process validations" phase
required	FALSE	boolean	Marks component as required
validator	null	MethodBinding	A method binding expression that refers to a method validationg the input
valueChangeListener	null	ValueChangeListener	A method binding expression that refers to a method for handling a valuchangeevent
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.
stars	5	int	Number of stars to display
rateListener	null	javax.el.Method Expression	A server side listener to process a RateEvent
update	null	String	Client side id of the component(s) to be updated after async partial submit request
disabled	FALSE	boolean	Disabled user interaction

Getting Started with Rating

Rating is an input component that takes a double variable as it's value.

```
public class RatingController {  
  
    private double rating;  
    //Getters and Setters  
}
```

```
<p:rating value="#{ratingController.rating}" />
```

When the enclosing form is submitted value of the rating will be assigned to the rating variable.

Number of Stars

Default number of stars is 5, if you need less or more stars use the stars attribute. Following rating consists of 10 stars.

```
<p:rating value="#{ratingController.rating}" stars="10"/>
```



Display Value Only

In cases where you only want to use the rating component to display the rating value and disallow user interaction, set disabled to true.

Ajax RateListeners

In order to respond to rate events instantly rather than waiting for the user to submit the form, use the RateListener feature which sends an RateEvent via an ajax request. On server side you can listen these RateEvent by defining RateListeners as MethodExpressions.

Rating below responds to a rate event instantly and updates the message component whose value is provided by the defined RateListener.

```
<p:rating rateListener="#{ratingController.handleRate}" update="msg"/>
<h:outputText id="msg" value="#{ratingController.message}" />
```

```
public class RatingController {
    private String message;

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }

    public void handleRate(RateEvent rateEvent) {
        message = "You rated:" + rateEvent.getRating();
    }
}
```

3.52 RemoteCommand

RemoteCommand provides a way to execute JSF backing bean methods directly from javascript.

Info

Tag	remoteCommand
Tag Class	org.primefaces.component.remotecontrol.RemoteCommandTag
Component Class	org.primefaces.component.remotecontrol.RemoteCommand
Component Type	org.primefaces.component.RemoteCommand
Component Family	org.primefaces.component
Renderer Type	org.primefaces.component.RemoteCommandRenderer
Renderer Class	org.primefaces.component.remotecontrol.RemoteCommandRenderer

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component.
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
event	null	String	Javascript event to attach the uiajax. Examples are "blur, keyup, click, etc"
action	null	javax.el.MethodExpression	A method expression that'd be processed in the partial request caused by uiajax.
actionListener	null	javax.faces.event.ActionListener	An actionlistener that'd be processed in the partial request caused by uiajax.
immediate	FALSE	boolean	Boolean value that determines the phaseId, when true actions are processed at apply_request_values, when false at invoke_application phase.
name	null	String	Name of the command

Name	Default	Type	Description
update	null	String	Client side id of the component(s) to be updated after async partial submit request
onstart	null	String	Javascript code to execute before ajax request is begins.
oncomplete	null	String	Javascript code to execute after ajax request is completed.

Getting started with RemoteCommand

RemoteCommand is used by invoking the command from your javascript function.

```
<p:remoteCommand name="increment" actionListener="#{counter.increment}"
    out="count" />
<h:outputText id="count" value="#{counter.count}" />
```

```
<script type="text/javascript">
function customfunction() {
    //custom code
    increment();           //makes a remote call
}
</script>
```

That's it whenever you execute your custom javascript function(eg customfunction()), a remote call will be made and output text is updated.

Note that remoteCommand must be nested inside a form.

3.53 Resizable

PrimeFaces features a resizable component that has the ability to make a JSF component resizable. Resizable can be used on various components like resize an input fields, panels, menus, images and more.

Info

Tag	resizable
Tag Class	org.primefaces.component.resizable.ResizableTag
Component Class	org.primefaces.component.resizable.Resizable
Component Type	org.primefaces.component.Resizable
Component Family	org.primefaces.component
Renderer Type	org.primefaces.component.ResizableRenderer
Renderer Class	org.primefaces.component.resizable.ResizableRenderer

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
widgetVar	null	String	Javascript variable name of the wrapped widget
proxy	FALSE	boolean	Displays a proxy when resizing
status	FALSE	boolean	Shows the height and width of the resizing component
handles	null	boolean	Handles to use, any combination of 't', 'b', 'r', 'l', 'bl', 'br', 'tl', 'tr' is valid, shortcut "all" enables all handlers.
ghost	FALSE	String	Displays a ghost effect
knobHandles	FALSE	boolean	Displays smaller handles

Name	Default	Type	Description
animate	FALSE	String	Controls animation
effect	null	String	Effect for animation
animateDuration	0.75	double	Duration of animation
minWidth	null	Integer	Minimum width to resize
maxWidth	null	Integer	Maximum width to resize
minHeight	null	Integer	Minimum height to resize
maxHeight	null	Integer	Maximum height to resize

Getting started with resizable

To make a component resizable, just add p:resizable as a child to a parent component that needs to be resized;

```
<h:graphicImage id="img" value="/ui/barca/campnou.jpg">
    <p:resizable />
</h:graphicImage>
```

That's it now an image can be resized by the user if he/she wants to see more detail :) Another common use case is the input fields, if users need more space for a textarea, make it resizable by;

```
<h:inputTextarea id="area" value="Resize me if you need more space">
    <p:resizable />
</h:inputTextarea>
```

Note: It's important for components that're resized to have an assigned id because some components do not render their clientId's if you don't give them an id explicitly.

Animations

Other than plain resize handling, animations and effects are also supported.

```
<h:inputTextarea id="area" value="Resize me!!!">
    <p:resizable proxy="true" animate="true" effect="bounceOut"/>
</h:inputTextarea>
```

Effect names

- backBoth
- backIn
- backOut
- bounceBoth
- bounceln
- bounceOut
- easeBoth
- easeBothStrong
- easeln
- easeInStrong
- easeNone
- easeOut
- easeOutStrong
- elasticBoth
- elasticln
- elasticOut

Note: Effect names are case sensitive and incorrect usage may result in javascript errors

Boundaries

To prevent overlapping with other elements on page, boundaries need to be specified. There're 4 attributes for this minWidth, maxWidth, minHeight and maxHeight. The valid values for these attributes are numbers in terms of pixels.

```
<h:inputTextarea id="area" value="Resize me!!!">
    <p:resizable maxWidth="200" maxHeight="200"/>
</h:inputTextarea>
```

3.54 Resource

Resource component enables resources like javascript and css bundled with PrimeFaces to be added to a page.

Info

Tag	<code>resource</code>
Tag Class	<code>org.primefaces.component.resource.ResourceTag</code>
Component Class	<code>org.primefaces.component.resource.Resource</code>
Component Type	<code>org.primefaces.component.Resource</code>
Component Family	<code>org.primefaces.component</code>
Renderer Type	<code>org.primefaces.component.ResourceRenderer</code>
Renderer Class	<code>org.primefaces.component.resource.ResourceRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
<code>name</code>	null	String	Path of the resource

Getting started with resource

The best place to locate p:resource is inside head tag. Following resource adds jquery bundled with Primefaces to the page.

```
<head>
  <p:resource name="/jquery/jquery.js" />
</head>
```

3.55 Resources

Resources component renders all script and link tags necessary for PrimeFaces component to work.

Info

Tag	<code>resources</code>
Tag Class	<code>org.primefaces.component.resources.ResourcesTag</code>
Component Class	<code>org.primefaces.component.resources.Resources</code>
Component Type	<code>org.primefaces.component.Resources</code>
Component Family	<code>org.primefaces.component</code>
Renderer Type	<code>org.primefaces.component.ResourcesRenderer</code>
Renderer Class	<code>org.primefaces.component.resources.ResourcesRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
<code>exclude</code>	null	String	Comma separated list of resources to be excluded.

Getting started with resources

The best place to locate p:resources is inside head tag.

```
<head>
  <p:resources />
</head>
```

Resources Section contains more information about Resource Handling and how to exclude particular resources from adding to the page.

3.56 Slider

Slider component enhances a simple input text to allow providing the input value via a slider. Slider can work both in horizontal and vertical mode.



Info

Tag	<code>slider</code>
Tag Class	<code>org.primefaces.component.slider.SliderTag</code>
Component Class	<code>org.primefaces.component.slider.Slider</code>
Component Type	<code>org.primefaces.component.Slider</code>
Component Family	<code>org.primefaces.component</code>
Renderer Type	<code>org.primefaces.component.SliderRenderer</code>
Renderer Class	<code>org.primefaces.component.slider.SliderRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
<code>for</code>	null	String	Id of the input text that the slider will be used for
<code>minValue</code>	null	int	Minimum value of the slider
<code>maxValue</code>	null	int	Maximum value of the slider
<code>thumbImage</code>	null	String	Image of the slider thumb
<code>tickMarks</code>	1	int	Fixed pixel increments that the slider move in

Name	Default	Type	Description
animate	TRUE	boolean	Boolean value to enable/disable the animated move when background of slider is clicked
type	horizontal	String	Sets the type of the slider, “horizontal” or “vertical”.
size	200	int	Width of the slider background, change this if you use a background with a width different than 200.

Getting started with slider

Slider requires an input text component to work with, **for** attribute is used to set the id of the input text component whose input will be provided by the slider.

```
public class SliderController {  
  
    private int number;  
  
    public String getNumber() {  
        return number;  
    }  
  
    public void setNumber(int number) {  
        this.number = number;  
    }  
}
```

```
<h:inputText id="number" value="#{sliderController.number}" />  
  
<p:slider for="number" minValue="0" maxValue="200"/>
```

Vertical Slider

By default slider works horizontally, vertical sliding is also supported and can be set using the **type** attribute.

```
<h:inputText id="number" value="#{sliderController.number}" />  
  
<p:slider for="number" type="vertical" minValue="0" maxValue="200"/>
```

Output of this would slider be;



Tick Marks

Tick marks defines the interval between each point during sliding. Default value is one. Following example restricts the values slider can provide to multiples of ten.

```
<h:inputText id="number" value="#{sliderController.number}" />
<p:slider for="number" tickMarks="10" minValue="0" maxValue="200"/>
```

Animation

Sliding is animated by default, if you want to turn it off animate attribute set the *animate* attribute to false.

Boundaries

Maximum and minimum boundaries for the sliding is defined using minValue and maxValue attributes. Following slider can slide between -100 and +100.

```
<h:inputText id="number" value="#{sliderController.number}" />
<p:slider for="number" minValue="-100" maxValue="100"/>
```

3.57 Spinner

Spinner is a an input component to provide a numerical input via two buttons that control the actual value.



Info

Tag	<code>spinner</code>
Tag Class	<code>org.primefaces.component.spinner.SpinnerTag</code>
Component Class	<code>org.primefaces.component.spinner.Spinner</code>
Component Type	<code>org.primefaces.component.Spinner</code>
Component Family	<code>org.primefaces.component</code>
Renderer Type	<code>org.primefaces.component.SpinnerRenderer</code>
Renderer Class	<code>org.primefaces.component.spinner.SpinnerRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
<code>value</code>	null	Object	Value of the component than can be either an EL expression or a literal text
<code>converter</code>	null	Converter/String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id

Name	Default	Type	Description
immediate	FALSE	boolean	Boolean value that specifies the lifecycle phase the valueChangeEvents should be processed, when true the events will be fired at "apply request values", if immediate is set to false, valueChange Events are fired in "process validations" phase
required	FALSE	boolean	Marks component as required
validator	null	MethodBinding	A method binding expression that refers to a method validationg the input
valueChangeListener	null	ValueChangeListene	A method binding expression that refers to a method for handling a valuchangeevent
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.
stepFactor	1	double	Stepping factor for each increment and decrement
accesskey	null	String	Html accesskey attribute
alt	null	String	Html alt attribute
dir	null	String	Html dir attribute
disabled	FALSE	Boolean	Html disabled attribute
lang	null	String	Html lang attribute
maxlength	null	Integer	Html maxlength attribute
onblur	null	String	Html onblur attribute
onchange	null	String	Html onchange attribute
onclick	null	String	Html onclick attribute
ondblclick	null	String	Html ondblclick attribute
onfocus	null	String	Html onfocus attribute
onkeydown	null	String	Html onkeydown attribute
onkeypress	null	String	Html onkeypress attribute
onkeyup	null	String	Html onkeyup attribute

Name	Default	Type	Description
onmousedown	null	String	Html onmousedown attribute
onmousemove	null	String	Html onmousemove attribute
onmouseout	null	String	Html onmouseout attribute
onmouseover	null	String	Html onmouseover attribute
onmouseup	null	String	Html onmouseup attribute
readonly	FALSE	Boolean	Html readonly attribute
size	null	Integer	Html size attribute
style	null	String	Html style attribute
styleClass	null	String	Html styleClass attribute
tabindex	null	Integer	Html tabindex attribute
title	null	String	Html title attribute

Getting Started with Spinner

Spinner is an input component and used just like a standard input text.

```
<p:spinner value="#{spinnerController.number}" />
```

```
public class SpinnerController {

    private int number;

    public String getNumber() {
        return number;
    }

    public void setNumber(int number) {
        this.number = number;
    }
}
```

Step Factor

Other than integers, spinner also support doubles so the fractional part can be controlled with spinner as well. For doubles use the stepFactor attribute to specify stepping amount. Following example uses a stepFactor 0.25.

```
<p:spinner value="#{spinnerController.number}" stepFactor="0.25"/>
```

```
public class SpinnerController {  
  
    private double number;  
  
    public Double getNumber() {  
        return number;  
    }  
  
    public void setNumber(Double number) {  
        this.number = number;  
    }  
}
```

Output of this spinner would be;



In case an increment happens, value is incremented by 0.25.



3.58 Submenu

Submenu is nested in a menu component and represents a navigation group.

Info

Tag	<code>submenu</code>
Tag Class	<code>org.primefaces.component.submenu.SubmenuTag</code>
Component Class	<code>org.primefaces.component.submenu.Submenu</code>
Component Type	<code>org.primefaces.component.Submenu</code>
Component Family	<code>org.primefaces.component</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component.
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean.
<code>label</code>	null	String	Label of the submenu header.
<code>url</code>	null	String	Url to be navigated to when the label is clicked.
<code>style</code>	null	String	Style of the menuitem label.
<code>styleClass</code>	null	String	StyleClass of the menuitem label.

Getting started with Submenu

Please see Menu section to find out how submenu is used with the menu.

3.59 Stack

Stack is a navigation component that mimics the stacks feature in Mac OS X.



Info

Tag	<code>stack</code>
Tag Class	<code>org.primefaces.component.stack.StackTag</code>
Component Class	<code>org.primefaces.component.stack.Stack</code>
Component Type	<code>org.primefaces.component.Stack</code>
Component Family	<code>org.primefaces.component</code>
Renderer Type	<code>org.primefaces.component.StackRenderer</code>
Renderer Class	<code>org.primefaces.component.stack.StackRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component.
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean.
<code>icon</code>	null	String	An optional image to contain stacked items.

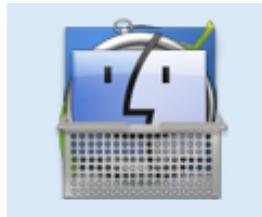
Name	Default	Type	Description
openSpeed	300	String	Speed of the animation when opening the stack.
closeSpeed	300	Integer	Speed of the animation when opening the stack.
widgetVar	null	String	Javascript variable name of the client side widget.

Getting started with Stack

Each item in the stack is represented with stackItems. Stack below has five stack items with different icons and labels.

```
<p:stack icon="/images/stack/stack.png">
  <p:stackItem label="Aperture" icon="/images/stack/aperture.png" url="#" />
  <p:stackItem label="Photoshop" icon="/images/stack/photoshop.png" url="#" />
  <p:stackItem label="Coda" icon="/images/stack/coda.png" url="#" />
  <p:stackItem label="Safari" icon="/images/stack/safari.png" url="#" />
  <p:stackItem label="Finder" icon="/images/stack/finder.png" url="#" />
</p:stack>
```

Initially stack will be rendered in collapsed mode;



Location

Stack is a fixed positioned element and location can be change via css. There's one important css selector for stack called `.pf-stack`. Override this style to change the location of stack.

```
.pf-stack {
  bottom: 28px;
  right: 40px;
}
```

3.60 StackItem

StackItem is used by the stack component to represent each navigation item in stack..

Info

Tag	stackItem
Tag Class	org.primefaces.component.stack.StackItemTag
Component Class	org.primefaces.component.stack.StackItem
Component Type	org.primefaces.component.Stack
Component Family	org.primefaces.component

Attributes

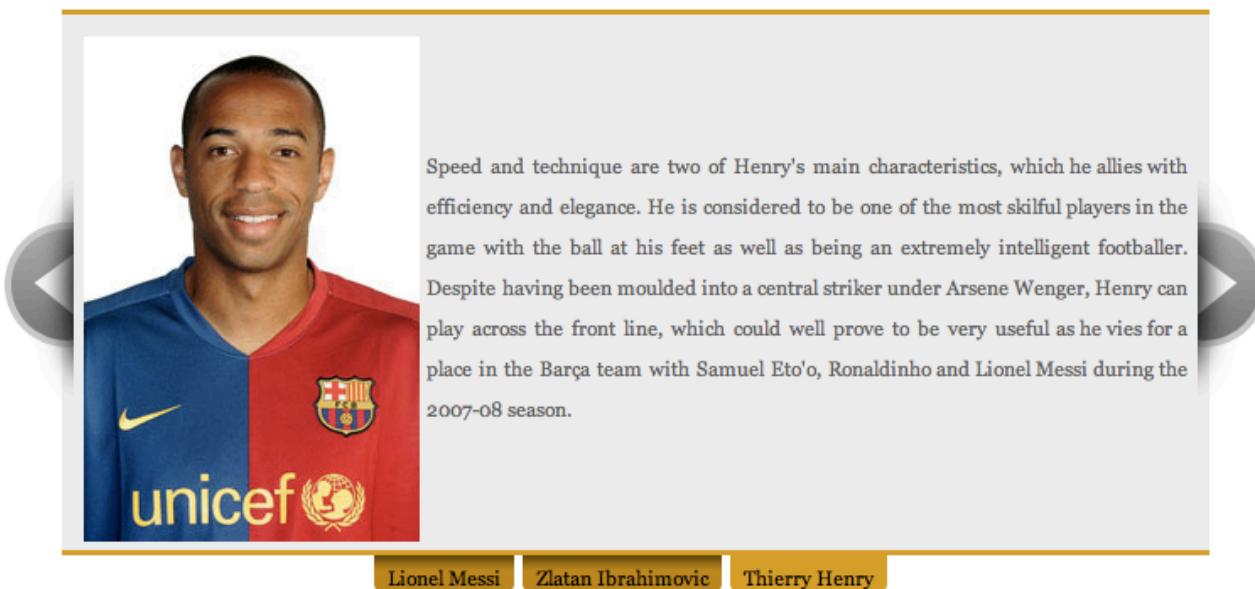
Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
label	null	String	Label of the item.
onclick	null	String	Onclick handler.
icon	null	String	Icon to be displayed.
url	null	String	URL to be used for navigation.

Getting started with Stack

Please see STack component section to find out how stackItem is used.

3.61 TabSlider

TabSlider is similar to tabView but display tabs using an easing slide effect.



Info

Tag	tabSlider
Tag Class	org.primefaces.component.tabslider.TabSliderTag
Component Class	org.primefaces.component.tabslider.TabSlider
Component Type	org.primefaces.component.TabSlider
Component Family	org.primefaces.component
Renderer Type	org.primefaces.component.TabSliderRenderer
Renderer Class	org.primefaces.component.tabslider.TabSliderRenderer

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean

Name	Default	Type	Description
widgetVar	null	String	Javascript variable name of client side widget.
activeIndex	1	Integer	Index of the active tab, count starts from 1.
effect	easeInOutExpo	String	Easing animation type used for slide effect.
effectDuration	600	Integer	Duration of effect in milliseconds.
navigator	TRUE	boolean	Enables tab name navigation.
style	null	String	Style to apply to main container element.
styleClass	null	String	StyleClass to apply to main container element.

Getting started with the TabSlider

TabSlider requires one or more child tab components to display.

```
<p:tabSlider>
    <p:tab title="Tab One">
        <h:outputText value="Lorem" />
    </p:tab>
    <p:tab title="Tab Two">
        <h:outputText value="Ipsum" />
    </p:tab>
    <p:tab title="Tab Three">
        <h:outputText value="Dolor" />
    </p:tab>
</p:tabSlider>
```

Effect Types

TabSlider uses jquery easing plugin for various easing effects, full list of effects is available at;

<http://gsgd.co.uk/sandbox/jquery/easing/>

3.62 TabView

TabView is a container component that displays its content in tabs.



Info

Tag	tabView
Tag Class	org.primefaces.component.tabview.TabViewTag
Component Class	org.primefaces.component.tabview.TabView
Component Type	org.primefaces.component.TabView
Component Family	org.primefaces.component
Renderer Type	org.primefaces.component.TabViewRenderer
Renderer Class	org.primefaces.component.tabview.TabViewRenderer

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
activeIndex	0	int	Index of the active tab
orientation	null	String	Defines how the tabs should oriented, valid values are top bottom left right
contentTransition	FALSE	boolean	Applies a transition effect during changing the tabs
widgetVar	null	String	Javascript variable name of the wrapped widget
toggleMode	client	String	Specifies the toggleMode, valid values are "client"(default) or "async"

Name	Default	Type	Description
cache	TRUE	boolean	When tab contents are lazy loaded by ajax toggleMode, caching only retrieves the tab contents once and subsequent toggles of a cached tab does not communicate with server. If caching is turned off, tab contents are refetched from server each time tab is clicked. This setting is true by default

Getting started with the TabView

TabView requires one or more child tab components to display.

```
<p:tabView>
    <p:tab title="Tab One">
        <h:outputText value="Lorem" />
    </p:tab>
    <p:tab title="Tab Two">
        <h:outputText value="Ipsum" />
    </p:tab>
    <p:tab title="Tab Three">
        <h:outputText value="Dolor" />
    </p:tab>
</p:tabView>
```

Orientation

There are four types of orientation tabview can display, default is the most common type; "top". Other valid values are "left", "right" and bottom.

```
<p:tabView orientation="bottom">
    <p:tab title="Tab One">
        <h:outputText value="Lorem" />
    </p:tab>
    <p:tab title="Tab Two">
        <h:outputText value="Ipsum" />
    </p:tab>
    <p:tab title="Tab Three">
        <h:outputText value="Dolor" />
    </p:tab>
</p:tabView>
```

For example, bottom orientation setting would give look like the following;



Content Transition

When the active tab changes, animation can be enabled by setting *contentTransition* attribute to true.

Lazy Loaded Tabs with Ajax

There're two toggleModes in tabview, **client** (default) and **async**. In client mode, all tab contents are rendered to the client, on the other hand in async mode, only the active tab contents are rendered and inactive tab contents are not. When an inactive tab header is clicked, tabview makes an ajax request, fetches the tab contents and displays them. Async mode is handy in reducing page size and performance since inactive tabs are lazy loaded.

To enable, lazy loaded tabs use *toggleMode="async"* setting.

```
<p:tabView toggleMode="async">
    <p:tab title="Tab One">
        <h:outputText value="Lorem" />
    </p:tab>
    <p:tab title="Tab Two">
        <h:outputText value="Ipsum" />
    </p:tab>
    <p:tab title="Tab Three">
        <h:outputText value="Dolor" />
    </p:tab>
</p:tabView>
```

Content Caching

Dynamically loaded tabs cache their contents by default, by doing so, reactivating a tab doesn't result in an ajax request since contents are cached. If you want to get content of a tab each time a tab is clicked, turn off caching by *cache="false"*.

```
<p:tabView toggleMode="async" cache="false">
    //contents
</p:tabView>
```

Skinning

TabView can be easily styled using CSS selectors, markup rendered by tabview is in following format.

```
<div id="mytabview" class="yui-navset">
    <ul class="yui-nav">
        <li class="selected"><a href="#tab1"><em>Label One</em></a></li>
        <li><a href="#tab2"><em>Label Two</em></a></li>
        <li><a href="#tab3"><em>Label Three</em></a></li>
    </ul>
    <div class="yui-content">
        <div><p>Lorem</p></div>
        <div><p>Ipsum</p></div>
        <div><p>Dolor</p></div>
    </div>
</div>
```

Given this fact an example skinning would be;

```
.yui-skin-sam .yui-navset .yui-nav{
    border-color: #33CC00;
}

.yui-skin-sam .yui-navset .yui-nav a {
    background: #99FF66;
}

.yui-skin-sam .yui-navset .yui-nav a:hover{
    background: #99FF00;
}

.yui-skin-sam .yui-navset .yui-nav .selected a,
.yui-skin-sam .yui-navset .yui-nav .selected a:focus,
.yui-skin-sam .yui-navset .yui-nav .selected a:hover{
    background: #33CC00;
}

.yui-skin-sam .yui-navset .yui-content{
    background: #FFFFCC;
    border-color:#33CC00;
}
```

Result of this skinning is;

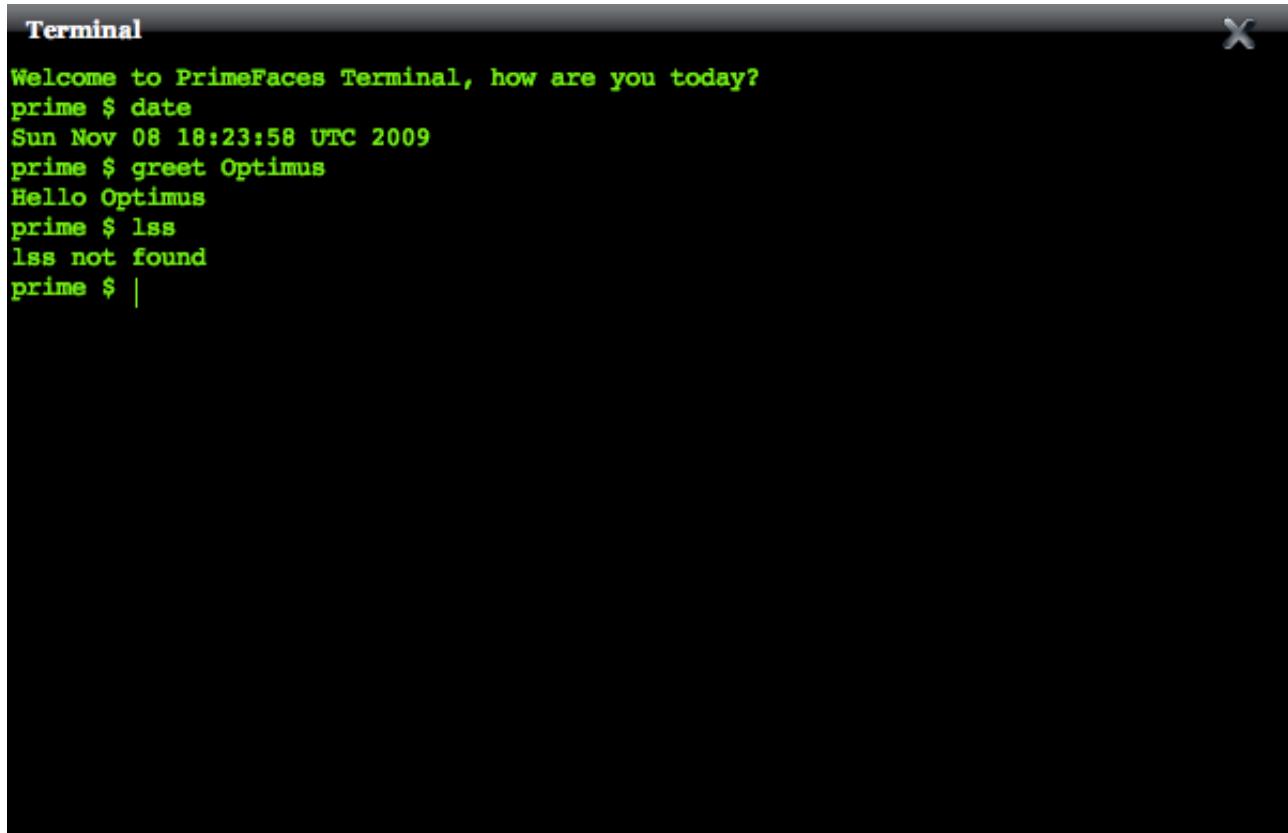


In detail, since default skin is yui-skin-sam all selectors should be prefixed by "yui-skin-sam". Table below contains most important selectors that would suffice to override when skinning tabview. In addition <http://developer.yahoo.com/yui/examples/tabview/skinning.html> page also contains detailed description of tabview skinning.

Selector	Effects
. yui-skin-sam . yui-navset	Main tabview container, this selector should be used for width or etc.
. yui-skin-sam . yui-navset . yui-nav	Container for the tab headers, border between tabcontent and tab headers are specified here
. yui-skin-sam . yui-navset . yui-nav a	Inactive Tabs
. yui-skin-sam . yui-navset . yui-nav . selected a	Active Tab
. yui-skin-sam . yui-navset . yui-content	Current tab content

3.63 Terminal

Terminal is an ajax powered web based terminal that brings desktop terminals to JSF.



```
Welcome to PrimeFaces Terminal, how are you today?  
prime $ date  
Sun Nov 08 18:23:58 UTC 2009  
prime $ greet Optimus  
Hello Optimus  
prime $ lss  
lss not found  
prime $ |
```

Info

Tag	terminal
Tag Class	org.primefaces.component.terminal.TerminalTag
Component Class	org.primefaces.component.terminal.Terminal
Component Type	org.primefaces.component.Terminal
Component Family	org.primefaces.component
Renderer Type	org.primefaces.component.TerminalRenderer
Renderer Class	org.primefaces.component.terminal.TerminalRenderer

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
width	null	String	Width of the terminal
height	null	String	Height of the terminal
welcomeMessage	null	String	Welcome message to be displayed on initial terminal load.
prompt	prime \$	String	Primary prompt text.
commandHandler	null	javax.el.MethodExpression	Method to be called with arguments to process.
widgetVar	null	String	Javascript variable name of the wrapped widget

Getting started with the Terminal

A command handler is necessary to interpret commands entered in terminal.

```
<p:terminal commandHandler="#{terminalBean.handleCommand}" />
```

```
public class TerminalBean {

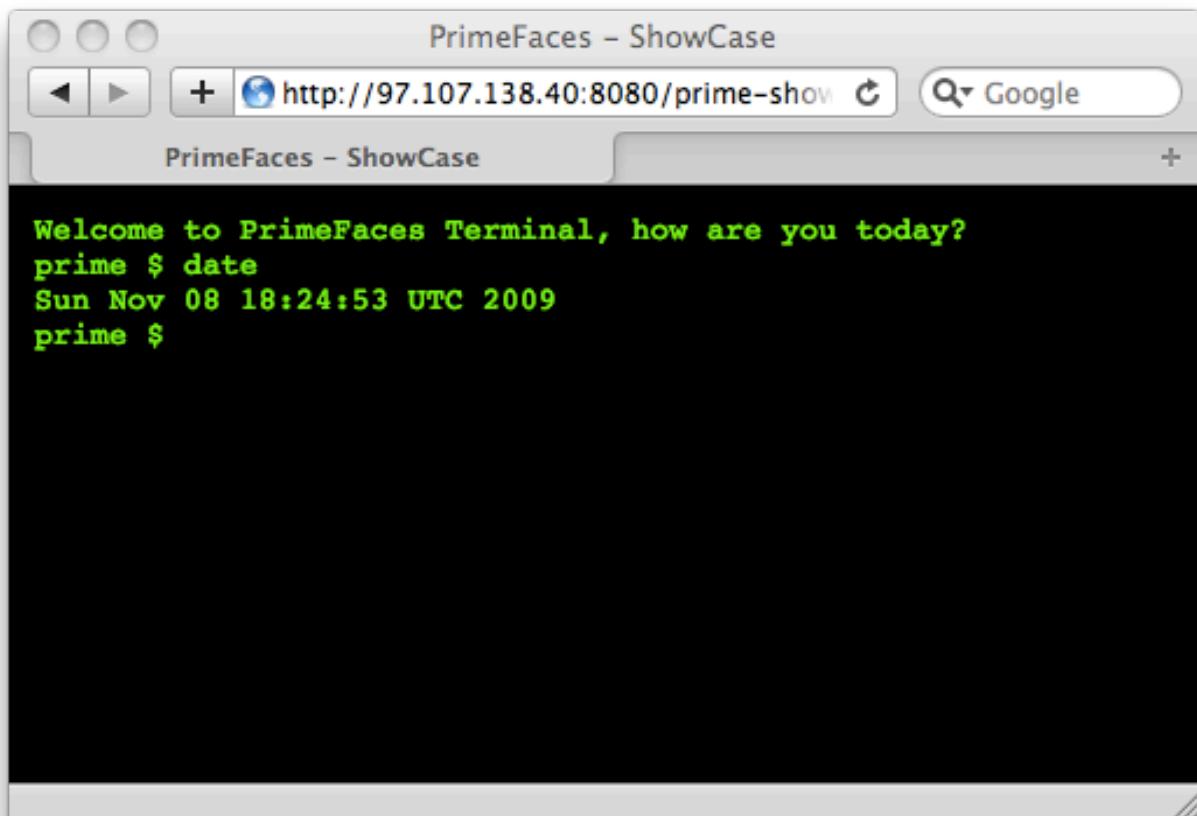
    public String handleCommand(String command, String[] params) {
        if(command.equals("greet"))
            return "Hello " + params[0];
        else if(command.equals("date"))
            return new Date().toString();
        else
            return command + " not found";
    }
}
```

Whenever a command is sent to the server, handleCommand method is invoked with the command name and the command arguments as a String array.

Full Screen Terminal

Setting width and height to 100% and placing the terminal as a direct child of body element is enough to create a full page terminal.

```
<body>
    <p:terminal width="100%" height="100%" />
</body>
```



3.64 Tooltip

Tooltip goes beyond the legacy html title attribute by providing custom effects, events, html content and skinning support.

[PrimeFaces Home](#)



Info

Tag	<code>tooltip</code>
Tag Class	<code>org.primefaces.component.tooltip.TooltipTag</code>
Component Class	<code>org.primefaces.component.tooltip.Tooltip</code>
Component Type	<code>org.primefaces.component.Tooltip</code>
Component Family	<code>org.primefaces.component</code>
Renderer Type	<code>org.primefaces.component.TooltipRenderer</code>
Renderer Class	<code>org.primefaces.component.tooltip.TooltipRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
<code>value</code>	null	Object	Value of the component than can be either an EL expression or a literal text
<code>converter</code>	null	Converter/String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id
<code>widgetVar</code>	null	String	Javascript variable name of client side tooltip object.

Name	Default	Type	Description
global	FALSE	boolean	A global tooltip converts each title attribute to a tooltip.
targetPosition	bottomRight	String	The corner of the target element to position the tooltip by.
position	topLeft	String	The corner of the tooltip to position the target's position.
showEvent	mouseover	String	Event displaying the tooltip.
showDelay	140	Integer	Delay time for displaying the tooltip.
showEffect	fade	String	Effect to be used for displaying.
showEffectLength	100	Integer	Time in milliseconds to display the effect.
hideEvent	mouseout	String	Event hiding the tooltip.
hideDelay	0	Integer	Delay time for hiding the tooltip.
hideEffect	fade	String	Effect to be used for hiding.
hideEffectLength	100	Integer	Time in milliseconds to process the hide effect.
theme	blue	String	Name of the built-in theme.

Getting started with the Tooltip

Tooltip is used by nesting it as a child of its target. Tooltip below sets a tooltip on the input field.

```
<h:inputSecret id="pwd" value="#{myBean.password}">
    <p:tooltip value="Password must contain only numbers"/>
</h:inputSecret>
```

Global Tooltip

One powerful feature of tooltip is using title attributes of other JSF components to create the tooltips, in this case you only need to place one tooltip to your page. This would also perform better compared to being nested as a child for each target.

```
<p:tooltip global="true" />
```

Effects

Showing and Hiding of tooltip along with the effect durations can be customized easily..

```
<h:inputSecret id="pwd" value="#{myBean.password}">
    <p:tooltip value="Password must contain only numbers"
        showEffect="slide" hideEffect="slide"
        showEffectLength="2000" hideEffectLength="2000"/>
</h:inputSecret>
```

Events

A tooltip is shown on mouseover event and hidden when mouse is out. If you need to change this behaviour use the showEvent and hideEvent feature. Tooltip below is displayed when the input gets the focus and hidden with onblur.

```
<h:inputSecret id="pwd" value="#{myBean.password}">
    <p:tooltip value="Password must contain only numbers"
        showEvent="focus" hideEvent="blue"/>
</h:inputSecret>
```

Delays

There're sensable defaults for each delay to display the tooltips and these can be configured easily as follows;

```
<h:inputSecret id="pwd" value="#{myBean.password}">
    <p:tooltip value="Password must contain only numbers"
        showDelay="2000" hideDelay="2000"/>
</h:inputSecret>
```

Tooltip above waits for 2 seconds to show and hide itself.

Html Content

Another powerful feature of tooltip is the ability to display custom content as a tooltip not just plain texts. An example is as follows;

```
<h:outputLink id="lnk" value="#">
    <h:outputText value="PrimeFaces Home" />
    <p:tooltip>
        <p:graphicImage value="/images/prime_logo.png" />
        <h:outputText value="Visit PrimeFaces Home" />
    </p:tooltip>
</h:outputLink>
```

Skinning Tooltip

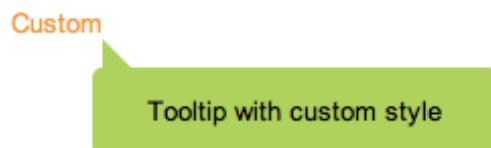
Tooltip supports built-in themes, default theme is blue. Here's the list of supported themes.

- blue
- cream
- dark
- green
- light
- red

If you need to create your own style rather than using the built-on ones, use the style configuration. Just like styling the charts provide your options with a custom javascript object.

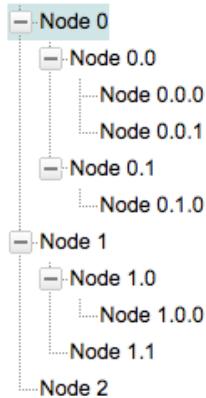
```
<script type="text/javascript">
var custom = {
    width: 200,
    padding: 5,
    background: '#A2D959',
    color: 'black',
    textAlign: 'center',
    border: {
        width: 7,
        radius: 5,
        color: '#A2D959'
    },
    tip: 'topLeft',
    name: 'dark'
};
</script>
```

```
<h:outputLink id="lnk" value="#">
    <h:outputText value="Custom" />
    <p:tooltip value="Tooltip with custom style" style="custom"/>
</h:outputLink>
```



3.65 Tree

Tree is a component that is commonly used for displaying hierarchical data or creating site navigations.



Info

Tag	tree
Tag Class	org.primefaces.component.tree.TreeTag
Component Class	org.primefaces.component.tree.Tree
Component Type	org.primefaces.component.Tooltip
Component Family	org.primefaces.component
Renderer Type	org.primefaces.component.TreeRenderer
Renderer Class	org.primefaces.component.tree.TreeRenderer

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean

Name	Default	Type	Description
value	null	Object	Value of the component, basically a TreeNode instance
toggleMode	async	String	Specifies the toggleMode, valid values are "client" or "async"(default)
expandAnim	null	String	Animation to be displayed on node expand, valid values are "FADE_IN" or "FADE_OUT"
collapseAnim	null	String	Animation to be displayed on node collapse, valid values are "FADE_IN" or "FADE_OUT"
nodeSelectListener	null	javax.el.MethodExpression	Method expression to listen node select events
nodeExpandListener	null	javax.el.MethodExpression	Method expression to listen node expand events
nodeCollapseListener	null	javax.el.MethodExpression	Method expression to listen node collapse events
cache	TRUE	boolean	Specifies caching on dynamically loaded nodes. When set to true expanded nodes will be kept in memory.
widgetVar	null	String	Javascript variable name of the wrapped widget
onNodeClick	null	String	Javascript event to process when a tree node is clicked.
expanded	FALSE	boolean	When set to true, all nodes will be displayed as expanded on initial page load.

Getting started with the Tree

Tree is populated with a org.primefaces.model.tree.TreeNode instance which corresponds to the root. TreeNode has a hierarchical data structure and represents the data to be populated in tree.

```

private TreeNode root;

public TreeBean() {
    root = new TreeNode("Root");
    TreeNode node0 = new TreeNode("Node 0");
    TreeNode node1 = new TreeNode("Node 1");
    TreeNode node2 = new TreeNode("Node 2");

    TreeNode node00 = new TreeNode("Node 0.0");
    TreeNode node01 = new TreeNode("Node 0.1");

    TreeNode node10 = new TreeNode("Node 1.0");
    TreeNode node11 = new TreeNode("Node 1.1");
    node00.addChild(new TreeNode("Node 0.0.0"));
    node00.addChild(new TreeNode("Node 0.0.1"));
    node01.addChild(new TreeNode("Node 0.1.0"));

    node10.addChild(new TreeNode("Node 1.0.0"));

    node0.addChild(node00);
    node0.addChild(node01);

    node1.addChild(node10);
    node1.addChild(node11);

    root.addChild(node0);
    root.addChild(node1);
    root.addChild(node2);
}

```

```

public TreeNode getModel() {
    return root;
}

```

Once model is instantiated via TreeNodes, bind the model to the tree as the value

```
<p:tree value="#{treeBean.model}"></p:tree>
```

Toggling Mode

Tree supports two toggling modes; **client** and **async**.

Client

When toggling is set to client all the treenodes in model are sent to the client and tree is created, this mode is suitable for relatively small datasets and provides fast user

interaction. On the otherhand it's not suitable for large data since all the data is sent to the client.

Async

Async mode uses ajax to populate and fetch the treemodel, compared to the client toggling, async mode has the advantage to deal with large data because only the child nodes of the root node is sent to the client and whole tree is lazily populated. When a node is expanded, tree gets the children nodes of the expanded nodes and sends them to the client for display.

Event Handling

Tree is an interactive component, it can notify listeners about certain events. There're currently three events supported, node select, expand and collapse. For example when a node is expanded and a server side nodeExpandListener is defined on tree, the particular java method is executed with the NodeExpandEvent. Following tree has three listeners;

```
<p:tree value="#{treeBean.model}" toggleMode="async"
    nodeSelectListener="#{treeBean.onNodeSelect}"
    nodeExpandListener="#{treeBean.onNodeExpand}"
    nodeCollapseListener="#{treeBean.onNodeCollapse} />
```

The listeners are simple methodexpressions,

```
public void onNodeSelect(NodeSelectEvent event) {
    String node = event.getTreeNode().getData().toString();
    logger.info("Selected:" + node);
}

public void onNodeExpand(NodeExpandEvent event) {
    String node = event.getTreeNode().getData().toString();
    logger.info("Expanded:" + node);
}

public void onNodeCollapse(NodeCollapseEvent event) {
    String node = event.getTreeNode().getData().toString();
    logger.info("Collapsed:" + node);
}
```

Event listeners are mostly useful for dealing with huge amount of data. The idea for implementing such a use case would be providing only the root and child nodes to the tree, use event listeners to get the selected node and add new nodes to that particular tree at runtime. Please see the TreeNode api documentation for more information.

Node Caching

When caching is turned on(default), dynamically loaded nodes will be kept in memory so re-expanding a node will not trigger a server side request. In case it's set to false, collapsing the node will remove the children and expanding it later causes the children nodes to be fetched from server again.

When caching is turned on collapse and expand events are not notified on the server side.

Animations

Expand and Collapse operations can be animated using expandAnim and collapseAnim. There're two valid values for these attributes, FADE_IN and FADE_OUT.

```
<p:tree value="#{treeBean.model}"
        expandAnim="FADE_IN"
        collapseAnim="FADE_OUT"/>
```

Handling Node Click

If you need to execute custom javascript when a treenode is clicked, use the *onNodeClick* attribute. Your javascript method will be processed with passing an object containing node information as a parameter.

```
<p:tree onNodeClick="handleNodeClick" />
```

```
function handleNodeClick(event, node) {
    alert("You clicked:" + node.data);
}
```

Expand by default

If you need all nodes to be displayed as expanded on initial page load, set the expanded setting to true.

Skinning Tree

Treeview has certain css selectors for nodes, for more information please see;

<http://developer.yahoo.com/yui/treeview/#style>

As an example following tree displays it's nodes with folder icons. This example is also available in YUI website.

```

/* first or middle sibling, no children */
.ygtvtn { background: url(tn.gif) 0 0 no-repeat; width:17px; height:22px; }

/* first or middle sibling, collapsable */
.ygtvtm { background: url(tm.gif) 0 0 no-repeat; width:34px; height:22px;
cursor:pointer }

/* first or middle sibling, collapsable, hover */
.ygtvtmh { background: url(tmh.gif) 0 0 no-repeat; width:34px; height:22px;
cursor:pointer }

/* first or middle sibling, expandable */
.ygtvtp { background: url(tp.gif) 0 0 no-repeat; width:34px; height:22px;
cursor:pointer }

/* first or middle sibling, expandable, hover */
.ygtvtpmh { background: url(tpmh.gif) 0 0 no-repeat; width:34px; height:22px;
cursor:pointer }

/* last sibling, no children */
.ygtvln { background: url(ln.gif) 0 0 no-repeat; width:17px; height:22px; }

/* Last sibling, collapsable */
.ygtvlm { background: url(lm.gif) 0 0 no-repeat; width:34px; height:22px;
cursor:pointer }

/* Last sibling, collapsable, hover */
.ygtvlmh { background: url(lmh.gif) 0 0 no-repeat; width:34px; height:22px;
cursor:pointer }

/* Last sibling, expandable */
.ygtvlp { background: url(lp.gif) 0 0 no-repeat; width:34px; height:22px;
cursor:pointer }

/* Last sibling, expandable, hover */
.ygtvlph { background: url(lph.gif) 0 0 no-repeat; width:34px; height:22px;
cursor:pointer }

/* Loading icon */
.ygtvloading { background: url/loading.gif) 0 0 no-repeat; width:16px;
height:22px; }

/* the style for the empty cells that are used for rendering the depth
 * of the node */
.ygtvdepthcell { background: url(vline.gif) 0 0 no-repeat; width:17px;
height:22px;
}

```

```
.ygtvblankdepthcell { width:17px; height:22px; }

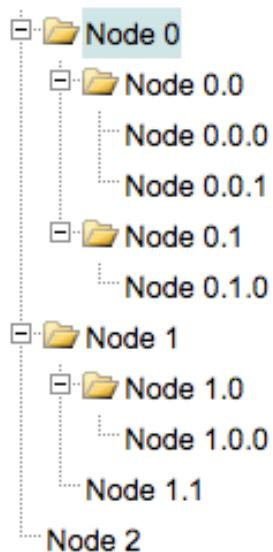
/* the style of the div around each node */
.ygtvitem { }

.ygtvitem table{
    margin-bottom:0;
}
.ygtvitem td {
    border:none;padding:0;
}

/* the style of the div around each node's collection of children */
.ygtvchildren { }
* html .ygtvchildren { height:1%; }

/* the style of the text label in ygTextNode */
.ygtvlabel, .ygtvlabel:link, .ygtvlabel:visited, .ygtvlabel:hover {
    margin-left:2px;
    text-decoration: none;
}
```

Result of this skin would be;



3.66 UIAjax

UIAjax is a generic component that can enable ajax communication on any JSF component. UIAjax is attached to a javascript event of it's parent.

UIAjax Classes

Tag	ajax
Tag Class	org.primefaces.component.uiajax.UIAjaxTag
Component Class	org.primefaces.component.uiajax.UIAjax
Component Type	org.primefaces.component.UIAjax
Component Family	org.primefaces.component
Renderer Type	org.primefaces.component.UIAjaxRenderer
Renderer Class	org.primefaces.component.uiajax.UIAjaxRenderer

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component.
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
event	null	String	Javascript event to attach the uiajax. Examples are "blur, keyup, click, etc"
action	null	javax.el.MethodExpression	A method expression that'd be processed in the partial request caused by uiajax.
actionListener	null	javax.faces.event.ActionListener	An actionlistener that'd be processed in the partial request caused by uiajax.
immediate	FALSE	boolean	Boolean value that determines the phasesId, when true actions are processed at apply_request_values, when false at invoke_application phase.
partialSubmit	FALSE	boolean	Flag for partial process, when set to true, parent form will be submitted but only the component the ajax event is attached to will be processed by JSF

Name	Default	Type	Description
update	null	String	Client side id of the component(s) to be updated after async partial submit request
onstart	null	String	Javascript code to execute before ajax request is begins.
oncomplete	null	String	Javascript code to execute after ajax request is completed.

Getting started with UIAjax

UIAjax needs a parent and a javascript event of it's parent to work with.

```
<h:inputText id="firstname" value="#{bean.firstname}">
    <p:ajax event="keyup" update="out" />
</h:inputText>

<h:outputText id="out" value="#{bean.firstname}" />
```

In this example, each time the user types and releases the key, an ajax request is sent to the server and normal JSF lifecycle is executed. When the response is received uiajax partially updates the output text with id "out".

Update is not mandatory, if you do not provide the update attribute, by default p:ajax updates it's parent form.

Note: Id attribute needs to be present because JSF implementations behave differently. While mojarra does not render the clientId of the inputText if id is not provided, myfaces does. UIAjax requires the clientId of it's parent at the rendered output.

UIAjax and ActionEvents

UIAjax extends from UICommand, this means it can execute action methods and actionListeners defined in a JSF backing bean. Following example executes an actionlistener each time keyup event occurs and counts the number of keyups.

```
<h:inputText id="counter">
    <p:ajax event="keyup" update="out" actionListener="#
    {counterBean.increment}"/>
</h:inputText>
<h:outputText id="out" value="#{counterBean.count}" />
```

```

public class CounterBean {
    private int count;
    public int getCount() {return count;}
    public void setCount(int count) {this.count = count;}

    public void increment(ActionEvent actionEvent) {
        count++;
    }
}

```

UIAjax and Validations

A tricky example of uiajax is validations to mimic client side validation. Following example validates the inputtext on blur event of the input text being validated.

```

<h:form prependId="false">
    <p:panel id="panel" header="New Person">
        <h:messages />
        <h:outputText value="5 characters minimum" />

        <h:panelGrid columns="3">
            <h:outputLabel for="firstname" value="Firstname: *" />
            <h:inputText id="firstname" value="#{pprBean.firstname}" >
                <f:validateLength minimum="5" />
                <p:ajax event="blur" update="panel" />
            </h:inputText>
            <h:message for="firstname" />
        </h:panelGrid>
    </p:panel>
</h:form>

```

Partial Validation and Model Update

By default whole parent form is submitted and processed by an ajax request caused by uiajax. In some cases if there're other input components with validations in the form, the uiajax component would not be processed completely since jsf lifecycle would jump to render response after validation failure.

UIAjax has a partialSubmit feature to avoid this scenario, it's default by false and when enabled only the ajaxified component would get processed by JSF. As an example, the view below has two input fields, one of the fields is required and other is ajaxified with the ajax attribute.

```
<h:form prependId="false">
    <h:outputText value="Required Firstname : " />
    <h:inputText id="firstname" value="#{pprBean.firstname}" required="true"/>
        <h:outputText id="out1" value="#{pprBean.firstname}" />

        <h:outputText value="Surname(Request on blur): " />
        <h:inputText id="surname" value="#{pprBean.surname}">
            <p:ajax event="blur" update="out2" partialSubmit="true"/>
        </h:inputText>
        <h:outputText id="out2" value="#{pprBean.surname}" />
</h:form>
```

When onblur event fires on surname field, on the server side JSF will only process (validate, update model) surname field and the validation of first name would not be triggered.

3.67 Watermark

Watermark displays a hint on an input field describing what the field is for.

Info

Tag	watermark
Tag Class	org.primefaces.component.watermark.WatermarkTag
Component Class	org.primefaces.component.watermark.Watermark
Component Type	org.primefaces.component.Watermark
Component Family	org.primefaces.component
Renderer Type	org.primefaces.component.WatermarkRenderer
Renderer Class	org.primefaces.component.watermark.WatermarkRenderer

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component.
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	0	int	Text of watermark.

Getting started with Watermark

Watermark is nested inside an input text.

```
<h:inputText value="#{bean.searchKeyword}">
    <p:watermark value="Search with a keyword" />
</h:inputText>
```

Form Submissions

Watermark is set as the text of an input field which shouldn't be sent to the server when an enclosing form is submitted. This would result in updating bean properties with watermark values. Watermark component is clever enough to handle this case, by default in non-ajax form submissions, watermarks are cleared. However ajax submissions required a little manual effort.

```
<h:inputText value="#{bean.searchKeyword}">
    <p:watermark value="Search with a keyword" />
</h:inputText>

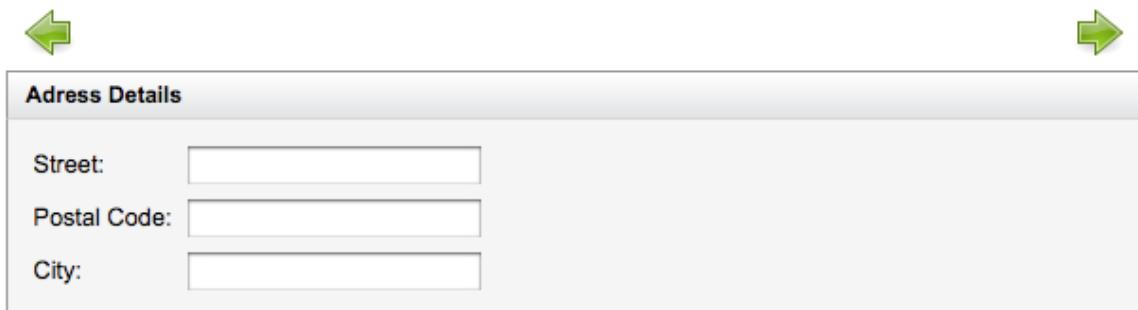
<h:commandButton value="Submit" />
<p:commandButton value="Submit" onclick="PrimeFaces.cleanWatermarks()" 
oncomplete="PrimeFaces.showWatermarks()" />
```

Skinning Watermark

There's only one css style class applying watermark which is '.pf-watermark', you can override this class to bring in your own style.

3.68 Wizard

Wizard provides a an ajax enhanced UI to implement conversations/workflows easily in a single page. Wizard consists of several child tab components where each tab represents a step in the process.



Info

Tag	wizard
Tag Class	org.primefaces.component.wizard.WizardTag
Component Class	org.primefaces.component.wizard.Wizard
Component Type	org.primefaces.component.Wizard
Component Family	org.primefaces.component
Renderer Type	org.primefaces.component.WizardRenderer
Renderer Class	org.primefaces.component.wizard.WizardRenderer

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component.
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
step	0	int	Index of the current step in flow.
effect	slide	String	Effect to be used for switching between steps; "slide" or "toggle".

Name	Default	Type	Description
width	400	int	Width of the viewport in pixels.
height	400	int	Height of the viewport in pixels.
style	null	String	Style of the main wizard container element.
styleClass	null	String	Style class of the main wizard container element.
speed	500	int	Speed of the animation in ms.
customUI	FALSE	boolean	When customUI is turned on, navigational controls won't be rendered and page authors can create their own ui using client side api

Getting started with Wizard

Each step in the flow is represented with a tab. As an example following wizard is used to create a new user in a total of 4 steps where last step is for confirmation of the information provided in first 3 steps.

To begin with create your backing bean, it's important that the bean must live across multiple requests so avoid a request scope bean. Optimal scope for wizard is viewScope which is built-in with JSF 2.0. For JSF 1.2 libraries like PrimeFaces optimus, Seam, MyFaces orchestra provides this scope.

```
public class UserWizard {

    private User user = new User();

    public User getUser() {return user;}
    public void setUser(User user) {this.user = user;}

    public void save(ActionEvent actionEvent) {
        //Persist user
        FacesMessage msg = new FacesMessage("Successful",
            "Welcome :" + user.getFirstname());
        FacesContext.getCurrentInstance().addMessage(null, msg);
    }
}
```

User is a simple pojo with properties such as firstname, lastname, email and etc. Following wizard requires 3 steps to get the user data; Personal Details, Address Details and Contact Details. Note that last tab contains read-only data for confirmation and the submit button.

```

<p:wizard height="200" width="600">
    <p:tab>
        <p:panel header="Personal Details">

            <h:messages errorClass="error"/>

            <h:panelGrid columns="2">
                <h:outputText value="Firstname: *" />
                <h:inputText value="#{userWizard.user.firstname}" required="true"/>

                <h:outputText value="Lastname: *" />
                <h:inputText value="#{userWizard.user.lastname}" required="true"/>

                <h:outputText value="Age: " />
                <h:inputText value="#{userWizard.user.age}" />
            </h:panelGrid>
        </p:panel>
    </p:tab>

    <p:tab>
        <p:panel header="Address Details">

            <h:messages errorClass="error"/>

            <h:panelGrid columns="2" columnClasses="label, value">
                <h:outputText value="Street: " />
                <h:inputText value="#{userWizard.user.street}" />

                <h:outputText value="Postal Code: " />
                <h:inputText value="#{userWizard.user.postalCode}" />

                <h:outputText value="City: " />
                <h:inputText value="#{userWizard.user.city}" />
            </h:panelGrid>
        </p:panel>
    </p:tab>

    <p:tab>
        <p:panel header="Contact Information">

            <h:messages errorClass="error"/>

            <h:panelGrid columns="2">
                <h:outputText value="Email: *" />
                <h:inputText value="#{userWizard.user.email}" required="true"/>

                <h:outputText value="Phone: " />
                <h:inputText value="#{userWizard.user.phone}"/>

                <h:outputText value="Additional Info: " />
                <h:inputText value="#{userWizard.user.info}"/>
            </h:panelGrid>
        </p:panel>
    </p:tab>

```

```

<p:tab>
    <p:panel header="Confirmation">

        <h:panelGrid id="confirmation" columns="6">
            <h:outputText value="Firstname: " />
            <h:outputText value="#{userWizard.user.firstname}" />

            <h:outputText value="Lastname: " />
            <h:outputText value="#{userWizard.user.lastname}" />

            <h:outputText value="Age: " />
            <h:outputText value="#{userWizard.user.age}" />

            <h:outputText value="Street: " />
            <h:outputText value="#{userWizard.user.street}" />

            <h:outputText value="Postal Code: " />
            <h:outputText value="#{userWizard.user.postalCode}" />

            <h:outputText value="City: " />
            <h:outputText value="#{userWizard.user.city}" />

            <h:outputText value="Email: " />
            <h:outputText value="#{userWizard.user.email}" />

            <h:outputText value="Phone " />
            <h:outputText value="#{userWizard.user.phone}" />

            <h:outputText value="Info: " />
            <h:outputText value="#{userWizard.user.info}" />

            <h:outputText />
            <h:outputText />
        </h:panelGrid>

        <p:commandButton value="Submit" actionListener="#{userWizard.save}" />

    </p:panel>
</p:tab>

</p:wizard>

```

AJAX and Partial Validations

Switching between steps is powered by ajax meaning each step is loaded dynamically with ajax. Partial validation is also built-in, by this way when you click next, only the current step is validated, if the current step is valid, next tab's contents are loaded with ajax.

Navigations

Wizard provides two images to interact with; next and prev. Please see the skinning wizard section to know more about how to change the look and feel of a wizard.

Custom UI

By default wizard displays right and left arrows to navigate between steps, if you need to come up with your own UI, use the customUI feature with the client side api.

```
<p:wizard customUI="true" widgetVar="wiz">
    ...
</p:wizard>

<h:outputLink value="#" onclick="wiz.next();">Next</h:outputLink>
<h:outputLink value="#" onclick="wiz.previous();">Back</h:outputLink>
```

Skinning Wizard

Wizard can be easily customized in terms of styling with the use of style/styleClass attributes and the well defined CSS selectors. All wizard controls reside in a div container element, style and styleClass attributes apply to this element.

Additionally a couple of css selectors exist for controlling the look and feel important parts of the wizard like the navigators. Following is the list.

Selector	Applies
.pf-wizard-nav	Container element of navigators
.pf-wizard-prev	Previous step navigator
.pf-wizard-next	Next step navigator
.pf-wizard-viewport	Element containing the tabs

4. TouchFaces

TouchFaces is the UI kit for developing mobile web applications with JSF. It mainly targets iPhone platform however mobile platforms with webkit browsers such as Android, Palm, Nokia S60,G1 etc are support as well. TouchFaces is included in PrimeFaces and no additional configuration is required other than the touchfaces taglib. TouchFaces is built on top of the jqTouch jquery plugin.

4.1 Getting Started with TouchFaces

There're a couple of special components belonging to the touchfaces namespace. Lets first create an example JSF page called touch.xhtml with the touchfaces namespace.

```
<f:view xmlns="http://www.w3.org/1999/xhtml"
        xmlns:f="http://java.sun.com/jsf/core"
        xmlns:i="http://primefaces.prime.com.tr/touch">

</f:view>
```

Next step is defining the *<i:application />* component.

```
<f:view xmlns="http://www.w3.org/1999/xhtml"
        xmlns:f="http://java.sun.com/jsf/core"
        xmlns:i="http://primefaces.prime.com.tr/touch">

    <i:application>

    </i:application>

</f:view>
```

Themes

TouchFaces ships with two built-in themes, default and dark. Themes can be customized using the theme attribute of the application. “Notes” sample app using the dark theme whereas other apps have the default iphone theme.

```
<i:application theme="dark">
    //content
</i:application>
```

Application Icon

iPhone has a nice feature allowing users to add web apps to their home screen so that later they can launch these apps just like a native iphone app. To assign an icon to your TouchFaces app use the icon attribute of the application component. It's important to use an icon of size 57x57 to get the best results.

```
<i:application icon="translate.png">
    //content
</i:application>
```

Here's an example demonstrating how it looks when you add your touchfaces app to your home screen.



That's it, you now have the base for your mobile web application. Next thing is building the UI with views.

4.2 Views

TouchFaces models each screen in a application as “views” and a view is created with the `<i:view />` component. Each view must have an id and an optional title.

```
<i:view id="home" title="Home Page">
    //content
</i:view>
```

You can have as many views as you want inside an application. To set a view as the home view use a convention and set the id of the view as “home”.

```
<f:view xmlns="http://www.w3.org/1999/xhtml"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:i="http://primefaces.prime.com.tr/touch">

    <i:application>
        <i:view id="home" title="Home Page">
            //Home view content
        </i:view>
    </i:application>
</f:view>
```

When you run this page, only the home view would be displayed, a view can be built with core JSF and components and TouchFaces specific components like `tableView`, `rowGroups`, `rowItems` and more.

TableViews

`TableView` is a useful control in iPhone sdk and touchfaces includes a `tableview` as well to provide a similar feature. `TableView` consists of `rowGroups` and `rowItems`. Here's a sample `tableView`.

```

<f:view xmlns="http://www.w3.org/1999/xhtml"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:i="http://primefaces.prime.com.tr/touch">

    <i:application>
        <i:view id="home" title="Home Page">
            <i:tableView>

                <i:rowGroup title="Group Title">
                    <i:rowItem value="Row 1"/>
                    <i:rowItem value="Row 2"/>
                </i:rowGroup>

            </i:tableView>
        </i:view>
    </i:application>
</f:view>

```

Output of this page would be;



Group Display Modes

A rowgroup can be displayed in a couple of different ways default way is 'rounded' which is used in previous example. Full list of possible values are;

- rounded
- edgeToEdge
- plastic
- metal

Following list uses edgetoedge display mode;

```
<i:tableView>
    <i:rowGroup title="Group Title" display="edgetoedge">
        <i:rowItem value="Row 1"/>
        <i:rowItem value="Row 2"/>
    </i:rowGroup>
</i:tableView>
```



4.3 Navigations

TouchFaces navigations are based on conventions and some components has the ability to trigger a navigation. An example is rowItem, using the view attribute you can specify which view to display when the rowItem is clicked. Also TouchFaces client side api provides useful navigation utilities.

```
<i:view>
    <i:tableView display="regular">
        <i:rowGroup title="Group Title">
            <i:rowItem value="Other View" view="otherview"/>
        </i:rowGroup>
    </i:tableView>
</i:view>

<i:view id="otherview" title="Other view">
    //Other view content
</i:view>
```

NavBarControl

You can also place navBarControls at the navigation bar for use cases such as navigation back and displaying another view. NavBarControl's are used as facets, following control is placed at the left top corner and used to go back to a previous view.

```
<i:view id="otherview" title="Other view">
    <f:facet name="leftNavBar">
        <i:navBarControl label="Home" view="home" />
    </f:facet>

    //view content

</i:view>
```



Similarly a navBarControl to place the right side of the navigation bar use *rightNavBar* facet.

Navigation Effects

Default animation used when navigation to a view is “slide”. Additional effects are;

- slide
- slideup
- flip
- dissolve
- fade
- flip
- pop
- swap
- cube

```
<i:view id="otherview" title="Other view">
    <f:facet name="leftNavBar">
        <i:navBarControl label="Settings" view="settings"
            effect="flip"/>
    </f:facet>

    //view content

</i:view>
```

TouchFaces Navigation API

TouchFaces client side object provides two useful navigation methods;

- goTo(viewName, animation)
- goBack()

Example below demonstrates how to execute a java method with p:commandLink and go to another view after ajax request is completed.

```
<p:commandLink actionListener="#{bean.value}" update="comp"
    oncomplete="TouchFaces.goTo('otherview', 'flip')"/>
```

4.4 Ajax Integration

TouchFaces is powered by PrimeFaces PPR infrastructure, this allows loading views with ajax, do ajax form submissions and other ajax use cases. Also rowItem component has built-in support for ajax and can easily load other views dynamically with ajax before displaying them. An example would be;

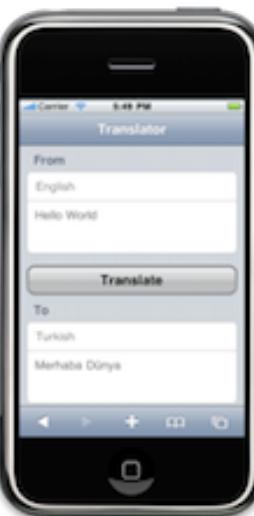
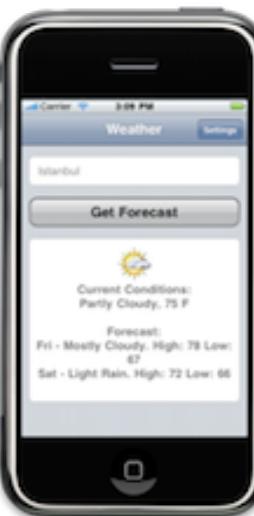
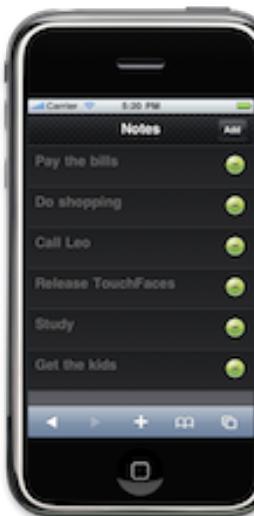
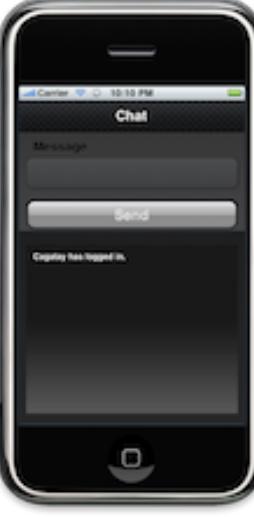
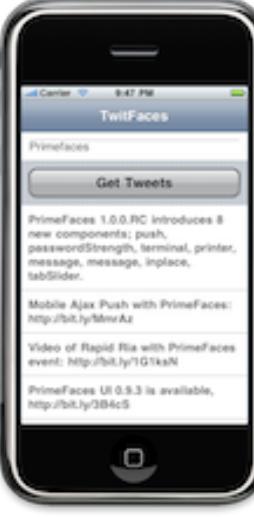
```
<i:view>
    <i:tableView display="regular">
        <i:rowGroup title="Group Title">
            <i:rowItem value="Other View" view="otherview"
                actionListener="#{bean.action}" update="table"/>
        </i:rowGroup>
    </i:tableView>
</i:view>

<i:view id="otherview" title="Other view">
    <i:tableView id="table" display="regular">
        <i:tableView>
    </i:view>
```

4.5 Sample Applications

There're various sample applications developed with TouchFaces, these apps are also deployed online so you can check them with your mobile device (preferably iphone, ipod touch or an android phone). Source codes are also available in PrimeFaces svn repository.

We strongly recommend using these apps as references since each of them use a different feature of TouchFaces.

Translate	Weather	News	Notes
			
Mobile Chat	TwitFaces	PathFinder - GPS	Empty Slot
			:)

4.6 TouchFaces Components

This section includes detailed tag information of TouchFaces Components.

4.6.1 Application

Info

Tag	application
Tag Class	<code>org.primefaces.touch.component.application.ApplicationTag</code>
Component Class	<code>org.primefaces.touch.component.applicaiton.Application</code>
Component Type	<code>org.primefaces.touch.Application</code>
Component Family	<code>org.primefaces.touch</code>
Renderer Type	<code>org.primefaces.touch.component.ApplicationRenderer</code>
Renderer Class	<code>org.primefaces.touch.component.application.ApplicationRenderer</code>

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component.
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
theme	null	String	Theme of the app, “default” or “dark”.
icon	null	String	Icon of the app.

4.6.2 NavBarControl

Info

Tag	<code>navBarControl</code>
Tag Class	<code>org.primefaces.touch.component.navbarcontrol.NavBarControlTag</code>
Component Class	<code>org.primefaces.touch.component.navbarcontrol.NavBarControl</code>
Component Type	<code>org.primefaces.touch.NavBarControl</code>
Component Family	<code>org.primefaces.touch</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component.
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
<code>label</code>	null	String	Label of the item.
<code>view</code>	null	String	Id of the view to be displayed.
<code>type</code>	back	String	Type of the display, "back" or "button".
<code>effect</code>	null	String	Effect to be used when displaying the view navigated to.

4.6.3 RowGroup

Info

Tag	<code>rowGroup</code>
Tag Class	<code>org.primefaces.touch.component.rowgroup.RowGroupTag</code>
Component Class	<code>org.primefaces.touch.component.rowgroup.RowGroup</code>
Component Type	<code>org.primefaces.touch.RowGroup</code>
Component Family	<code>org.primefaces.touch</code>
Renderer Type	<code>org.primefaces.touch.component.RowGroupRenderer</code>
Renderer Class	<code>org.primefaces.touch.component.rowgroup.RowGroupRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component.
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
<code>title</code>	null	String	Optional title of the row group.

4.6.4 RowItem

Info

Tag	rowItem
Tag Class	org.primefaces.touch.component.rowitem.RowItemTag
Component Class	org.primefaces.touch.component.rowitem.RowItem
Component Type	org.primefaces.touch.RowItem
Component Family	org.primefaces.touch
Renderer Type	org.primefaces.touch.component.RowItemRenderer
Renderer Class	org.primefaces.touch.component.rowitem.RowItemRenderer

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component.
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
view	null	String	Id of the view to be displayed.
url	null	String	Optional external url link.
update	null	String	Client side of the component(s) to be updated after the partial request.
value	null	String	Label of the item.
action	null	javax.el.MethodExpression	A method expression that'd be processed in the partial request caused by uiajax.
actionListener	null	javax.faces.event.ActionListener	An actionlistener that'd be processed in the partial request caused by uiajax.
immediate	FALSE	boolean	Boolean value that determines the phaseId, when true actions are processed at apply_request_values, when false at invoke_application phase.

4.6.5 Switch

Info

Tag	switch
Tag Class	org.primefaces.touch.component.switch.SwitchTag
Component Class	org.primefaces.touch.component.switch.Switch
Component Type	org.primefaces.touch.Switch
Component Family	org.primefaces.touch
Renderer Type	org.primefaces.touch.component.SwitchRenderer
Renderer Class	org.primefaces.touch.component.switch.SwitchRenderer

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component.
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Value of the component than can be either an EL expression or a literal text
converter	null	Converter/String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id
immediate	FALSE	boolean	Boolean value that specifies the lifecycle phase the valueChangeEvents should be processed, when true the events will be fired at "apply request values", if immediate is set to false, valueChange Events are fired in "process validations" phase
required	FALSE	boolean	Marks component as required
validator	null	MethodBinding	A method binding expression that refers to a method validationg the input

Name	Default	Type	Description
valueChange Listener	null	ValueChangeListener	A method binding expression that refers to a method for handling a valuchangeevent
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.

4.6.6 TableView

Info

Tag	<code>tableView</code>
Tag Class	<code>org.primefaces.touch.component.tableview.TableView</code>
Component Class	<code>org.primefaces.touch.component.tableview.TableView</code>
Component Type	<code>org.primefaces.touch.TableView</code>
Component Family	<code>org.primefaces.touch</code>
Renderer Type	<code>org.primefaces.touch.component.TableViewRenderer</code>
Renderer Class	<code>org.primefaces.touch.component.tableview.TableViewRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component.
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean

4.6.7 View

Info

Tag	<code>view</code>
Tag Class	<code>org.primefaces.touch.component.view.ViewTag</code>
Component Class	<code>org.primefaces.touch.component.view.View</code>
Component Type	<code>org.primefaces.touch.View</code>
Component Family	<code>org.primefaces.touch</code>
Renderer Type	<code>org.primefaces.touch.component.ViewRenderer</code>
Renderer Class	<code>org.primefaces.touch.component.viewrenderer.ViewRenderer</code>

Attributes

Name	Default	Type	Description
<code>id</code>	Assigned by JSF	String	Unique identifier of the component.
<code>rendered</code>	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
<code>binding</code>	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
<code>title</code>	regular	String	Optional title of the view.

5. Partial Page Rendering

In addition to components like autoComplete with built-in ajax capabilities, PrimeFaces also provides a generic PPR(Partial Page Rendering) mechanism to update any JSF component with an ajax request. Several components are equipped with the update attribute which could take any other JSF component's id on the page.

PPR Tips

Infrastructure

PrimeFaces Ajax Framework follows a lightweight approach compared to other AJAX and JSF solutions. PrimeFaces uses only one artifact: a PhaseListener to bring in AJAX. We don't approach AJAX requests different than the regular requests. As a result we don't need JSF extensions like AjaxViewRoot, AjaxStateManager, AjaxViewHandler, AjaxFilter, HtmlParser and so on. PrimeFaces aims to keep it clean, fast and lightweight.

Notifying Users

ajaxStatus is the component to notify the users about the status of **global** ajax requests. See the ajaxStatus section to get more information about the component.

Components such as commandButton and commandLink are equipped with the onstart and oncomplete callbacks, use these javascript callbacks to be notified about the ajax requests triggered by that particular component only.

Client Ids

PrimeFaces uses clientIds to find the component to retrieve the partial output, behind the scenes we use invokeOnComponent api which needs a clientId instead of the server id. As a result of this fact when defining your page with PPR, you need to provide the client ids. For example following will **not** work;

```
<h:form>
    <h:inputText value="#{bean.text}" />

    <p:commandButton value="Ajax" update="text"/>

    <h:outputText id="text" value="#{bean.firstname}" />
</h:form>
```

The reason of failure is that the actual client id of the outputtext is not id, it is prepended by the formId. A simple change is needed to make the example above work.

```
<h:form id="myform">
    <h:inputText value="#{bean.text}" />

    <p:commandButton value="Ajax" update="text"/>

    <h:outputText id="myform:text" value="#{bean.firstname}" />
</h:form>
```

Another way is to disable form's id being prepended. This will work as well since form doesn't add its id as a suffix to its children.

```
<h:form prependId="false">
    <h:inputText value="#{bean.text}" />

    <p:commandButton value="Ajax" update="text"/>

    <h:outputText id="text" value="#{bean.firstname}" />
</h:form>
```

Yet another way is not to use update attribute at all, if update is not present parent form is updated implicitly. So following is another working way;

```
<h:form>
    <h:inputText value="#{bean.text}" />

    <p:commandButton value="Ajax"/>

    <h:outputText id="text" value="#{bean.firstname}" />
</h:form>
```

Update Attribute

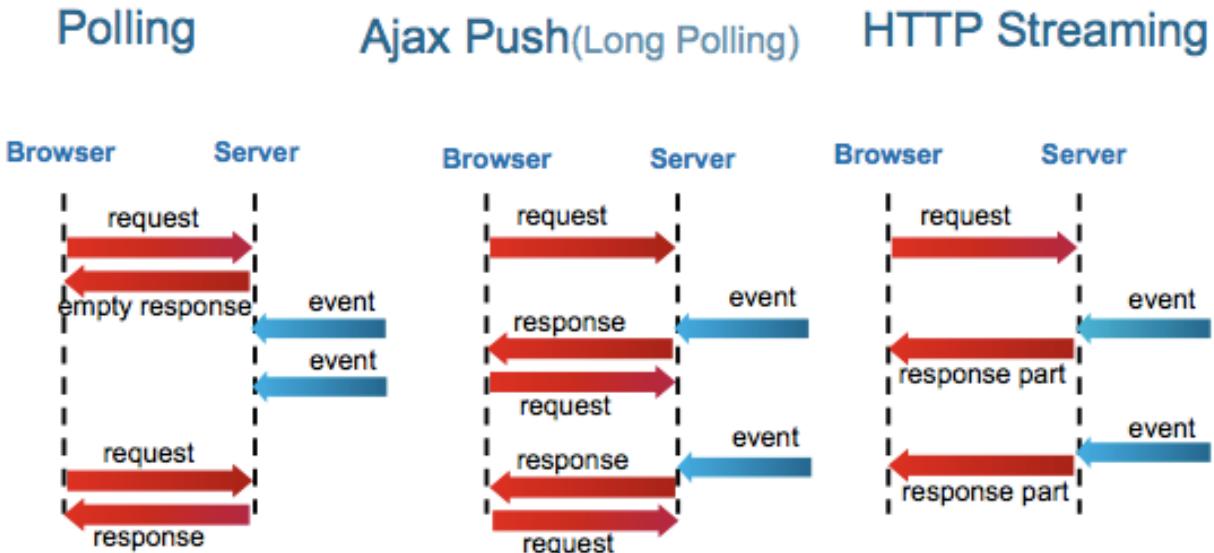
If you do not provide the component clientIds to be updated after ajax response, by default parent form of the component that triggers the ajax request is updated. This applies to p:commandButton, p:commandLink and p:ajax.

Plain HTML and JSP

When using JSP for JSF pages, PrimeFaces PPR has a limitation to update a component that contains plain html which means the html part will be ignored in partial response. This is only a case for JSP and doesn't happen with Facelets.

6. Ajax Push/Comet

Comet is a model allowing a web server to push data to the browsers. Auctions and chat are well known example use cases of comet technique. Comet can be implemented with either long-polling or http-streaming. Following is a schema describing these techniques.



Polling: Regular polling is not real comet, basically browser sends request to server based on a specific interval. This approach has nothing to do with comet and just provided for comparison.

Long-Polling: Browsers requests are suspended and only resumed when server decides to push data, after the response is retrieved browsers connects and begins to waiting for data again.

Http Streaming: With this approach, response is never committed and client always stays connected, push data is streamed to the client to process.

Current version of PrimeFaces is based on http-streaming, long-polling support will be added very soon in upcoming releases. PrimeFaces Push is built-on top of Atmosphere Framework. Next section describes atmosphere briefly.

6.1 Atmosphere

Atmosphere is a comet framework that can run on any application server supporting servlet 2.3+. Each container provides their own proprietary solution (Tomcat's CometProcessor, JBoss's HttpEvent, Glassfish Grizzly etc), Servlet 3.0 aims to unify these apis with a standard javax.servlet.AsyncListener.

Atmosphere does all the hard work, deal with container differences, browser compatibility, broadcasting of events and many more. See atmosphere home page for more information.

<http://atmosphere.dev.java.net>

6.2 PrimeFaces Push

PrimeFaces Lead Cagatay Civici is also a committer of Atmosphere Framework and as a result PrimeFaces Push is powered by Atmosphere Runtime. PrimeFaces simplifies developing comet applications with JSF, an example for this would be the PrimeFaces chat sample app that can easily be created with a couple of lines.

6.2.1 Setup

Comet Servlet

First thing to do is to configure the PrimeFaces Comet Servlet. This servlet handles the JSF integration and Atmosphere.

```
<servlet>
    <servlet-name>Comet Servlet</servlet-name>
    <servlet-class>org.primefaces.comet.PrimeFacesCometServlet</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>Comet Servlet</servlet-name>
    <url-pattern>/primefaces_comet/*</url-pattern>
</servlet-mapping>
```

Atmosphere Libraries

PrimeFaces needs at least version of 0.5, you can download atmosphere from atmosphere homepage, you'll also need the atmosphere-compat-* libraries. You can find these libraries at;

<http://download.java.net/maven/2/org/atmosphere/>

context.xml

If you're running tomcat, you'll also need a context.xml under META-INF.

```
<?xml version="1.0" encoding="UTF-8"?>
<Context>
    <Loader delegate="true"/>
</Context>
```

6.2.2. CometContext

Main element of PrimeFaces Push on server side is the `org.primefaces.comet.CometContext` that has a simple api to push data to browsers.

```
/**
 * @param channel Unique name of communication channel
 * @param data Information to be pushed to the subscribed listeners(browsers)
 */
CometContext.publish(String channel, Object data);
```

6.2.3 Push Component

`<p:push />` is a PrimeFaces component that handles the connection between the server and the browser, it has two attributes you need to define.

```
<p:push channel="chat" onpublish="handlePublish"/>
```

channel: Name of the channel to connect and listen.

onpublish: Javascript event handler to be called when server sends data.

6.2.4 Putting it together: A Chat application

In this section, we'll develop a simple chat application with PrimeFaces, let's begin with the backing bean.

```
public class ChatController implements Serializable {

    private String message;
    private String username;
    private boolean loggedIn;

    public void send(ActionEvent event) {
        CometContext.publish("chat", username + ": " + message);
        message = null;
    }

    public void login(ActionEvent event) {
        FacesContext.getCurrentInstance().addMessage(null, new
FacesMessage("You're logged in!"));
        loggedIn = true;
        CometContext.publish("chat", username + " has logged in.");
    }

    //getters&setters
}
```

And the chat.xhtml;

```

...
<head>
    <script type="text/javascript">
        function handlePublish(data) {
            $('#display').append(data + '<br />');
        }
    </script>
</head>

<body>

<p:outputPanel id="display" />

<h:form prependId="false">

    <p:growl id="growl" />

    <p:panel header="Sign in" rendered="#{!chatController.loggedIn}">
        <h:panelGrid columns="3" >
            <h:outputText value="Username:" />
            <h:inputText value="#{chatController.username}" />
            <p:commandButton value="Login"
                actionListener="#{chatController.login}"
                oncomplete="$( '#display' ).slideDown()"/>
        </h:panelGrid>
    </p:panel>

    <p:panel header="Signed in as : #{chatController.username}"
        rendered="#{chatController.loggedIn}" toggleable="true">
        <h:panelGrid columns="3">
            <h:outputText value="Message:" />
            <h:inputText id="txt" value="#{chatController.message}" />
            <p:commandButton value="Send"
                actionListener="#{chatController.send}"
                oncomplete="$( '#txt' ).val('');"/>
        </h:panelGrid>
    </p:panel>
</h:form>

<p:push channel="chat" onpublish="handlePublish" />

</body>
...

```

7. Compatibility

7.1 JSF Compatibility

Compatibility is a major goal of PrimeFaces, modules are tested against both implementations; Apache MyFaces 1.2 and Sun Mojarra 1.2. PrimeFaces is known to be working with both of the implementations and we take care of the differences between these implementations like specific parameters.

As of JSF 1.2, combining several JSF component libraries in a project might cause problems since each component library use their own JSF extensions and these customizations are subject to break each other's behavior. JSF 2.0 aims to help component library compatibility issues but it still depends on the component library developer's choices. Major cause of issues is that each component library handles Ajax differently and install several JSF extensions. PrimeFaces avoids extensions like viewhandlers, servlet filters, html parsing and etc to be a compatible library.

7.2 Browser Compatibility

Most of the PrimeFaces JSF components are built on top of YUI widgets. These components benefit from the excellent job on cross browser compatibility YUI does. YUI is tested against popular browsers like firefox 2-3, internet explorer 6-7-8, opera and safari. There's a nice browser matrix on yui page.

<http://developer.yahoo.com/yui/articles/gbs/>

8. Javascript

PrimeFaces mostly follows unobtrusive javascript which cleanly separates events from the html leading to a better design. There're two libraries we use, YUI for widget controls and jQuery for ajax, dom manipulation plus some UI plugins.

Namespaces

To be compatible with other javascript entities on a page, PrimeFaces defines three namespaces;

*PrimeFaces.widget.**

Contains custom PrimeFaces widgets like PrimeFaces.widget.Tree, PrimeFaces.widget.Poll and etc.

*PrimeFaces.ajax.**

Contains PrimeFaces ajax javascript api includung PrimeFaces.ajax.AjaxUtils, PrimeFaces.ajax.AjaxRequest and PrimeFaces.ajax.AjaxResponse.

PrimeFaces Object

PrimeFaces is the main javascript object providing utilities like onContentReady and more.

9. Resource Management

PrimeFaces UI streams bundled resources like css, js and images from the jar distribution. To be compatible with other component libraries we've chosen to use a Resource Servlet with the /primefaces_resources/* url pattern so that resource loading is separated from the JSF lifecycle.

You also need to add the <p:resources /> component to the head of the page, this involves a manual effort compared to other approaches but it's necessary to avoid parsing the html response and add <link />, <script /> tags. The best practice is to add <p:resources /> to your Facelets template so that you don't need to think about it anymore.

In addition PrimeFaces is designed with JSF 2.0 in mind so that you won't need the <p:resources /> and ResourceServlet when using PrimeFaces's JSF 2.0 optimized version.

Excluding Resources

As we've mentioned several times, compatibility is a major goal of PrimeFaces and we want to make sure PrimeFaces does not cause any problems regarding component interoperability. <p:resources /> adds all the resources to a page but suppose the following scenario, you have another component library that uses jQuery which is also used by PrimeFaces. The problem is jQuery would be added twice to your page, once by PrimeFaces and second time by the x library. In this case use the exclude property of <p:resources />. If you need to exclude more than one resources, provide a comma separated list.

```
<p:resources exclude="/jquery/jquery.js" />
```

In summary whole idea behind PrimeFaces is based on compatibility and performance.

10. Integration with Java EE

PrimeFaces UI is all about front-end and can be backed by your favorite enterprise application framework. We've created sample applications to demonstrate several technology stacks involving PrimeFaces at the front layer.

Source codes of these applications are available at the PrimeFaces subversion repository and they're deployed online time to time.

Application	Technologies
MovieCollector	PrimeFaces-Spring-JPA
PhoneBook	PrimeFaces-Seam-JPA
BookStore	PrimeFaces-Optimus-Guice-JPA

All applications are built with maven and use in memory databases so it's as easy as running;

mvn clean jetty:run or mvn clean jetty:run-exploded

command to deploy in your local environment.

Note: Spring WebFlow is not tested with and officially supported. MovieCollector application is based on the jsf centric spring integration wheres as spring webflow uses spring centric approach.

11. IDE Support

PrimeFaces IDE integration is at plan stage at the moment and a project called PrimeIDE will begin soon for eclipse support. NetBeans users already have good code completion and tag support with 6.8.

NetBeans

The screenshot shows the NetBeans code editor with the following XML code:

```

1 <?xml version='1.0' encoding='UTF-8' ?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/1999/xhtml"
3 <html xmlns="http://www.w3.org/1999/xhtml"
4   xmlns:h="http://java.sun.com/jsf/html"
5   xmlns:p="http://primefaces.prime.com.tr/ui">
6
7
8 <h:body>
9
10 <p:>
11   <p:accordionPanel> primefaces-p.tld
12   <p:ajax> primefaces-p.tld
13   <p:ajaxStatus> primefaces-p.tld
  
```

A code completion dropdown is open over the tag `<p:>`, listing various PrimeFaces components:

- <p:accordionPanel> primefaces-p.tld
- <p:ajax> primefaces-p.tld
- <p:ajaxStatus> primefaces-p.tld
- <p:autoComplete> primefaces-p.tld
- <p:barChart> primefaces-p.tld
- <p:calendar> primefaces-p.tld
- <p:captcha> primefaces-p.tld
- <p:carousel> primefaces-p.tld
- <p:chartSeries> primefaces-p.tld
- <p:collector> primefaces-p.tld
- <p:colorPicker> primefaces-p.tld
- <p:column> primefaces-p.tld
- <p:columnChart> primefaces-p.tld
- <p:commandButton> primefaces-p.tld
- <p:commandLink> primefaces-p.tld
- <p:confirmDialog> primefaces-p.tld
- <p:dataExporter> primefaces-p.tld

The screenshot shows the NetBeans code editor with the following XML code:

```

1 <?xml version='1.0' encoding='UTF-8' ?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/1999/xhtml"
3 <html xmlns="http://www.w3.org/1999/xhtml"
4   xmlns:h="http://java.sun.com/jsf/html"
5   xmlns:p="http://primefaces.prime.com.tr/ui">
6
7
8 <h:body>
9
10 <p:accordionPanel> |
11   </h:body>
12 </html>
  
```

A code completion dropdown is open over the attribute `<p:accordionPanel>`, listing its available attributes:

- activeIndex
- binding
- id
- multipleSelection
- rendered
- speed
- style
- styleClass

Eclipse

PrimeIDE will be an eclipse based plugin with Palette, Code Editor and Wizard integration. Beta release of PrimeIDE is for early 2010.

12. Portlets

PrimeFaces works well in a portlet environment, both portlet 1.0 and portlet 2.0 JSRs are tested and supported. PrimeFaces Ajax infrastructure is tuned for portal environments.

PrimeFaces portlets should work with any compliant portlet-bridge implementation, we highly encourage using MyFaces portlet bridge which is the reference implementation.

portlet.xml

Here's a sample portlet.xml from the prime-portlet application that is available in subversion repository of PrimeFaces.

```
<?xml version="1.0"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
    version="2.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-
app_2_0.xsd http://java.sun.com/xml/ns/portlet/app_2_0.xsd">
    <portlet>
        <portlet-name>primeportlet</portlet-name>
        <display-name>Prime Portlet</display-name>
        <portlet-class>javax.portlet.faces.GenericFacesPortlet</portlet-class>

        <init-param>
            <name>javax.portlet.faces.defaultViewId.view</name>
            <value>/view.jsp</value>
        </init-param>

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

```

```
<init-param>
    <name>javax.portlet.faces.preserveActionParams</name>
    <value>true</value>
</init-param>

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

<portlet-info>
    <title>Prime Portlet</title>
    <short-title>Prime</short-title>
</portlet-info>

</portlet>
</portlet-app>
```

Important note here is to preserve the action parameters to make PrimeFaces PPR work with portlets.

```
<init-param>
    <name>javax.portlet.faces.preserveActionParams</name>
    <value>true</value>
</init-param>
```

Other than this there's no specific change in the application configuration.

13. Project Resources

Documentation

Reference documentation is the major resource for documentation, for additional documentation like apidocs, taglib docs, wiki and more please visit;

<http://primefaces.prime.com.tr/en/documentation>

Support Forum

PrimeFaces discussions take place at the support forum. Forum is public to everyone and registration is required to do a post.

<http://primefaces.prime.com.tr/forum>

Source Code

PrimeFaces source is at google code subversion repository.

<http://primefaces.googlecode.com/svn>

Issue Tracker

PrimeFaces issue tracker uses google code's issue management system. Please use the forum before creatin an issue instead.

<http://code.google.com/p/primefaces/issues/list>

Online Demo

PrimeFaces ShowCase demo is deployed online at;

<http://97.107.138.40:8080/prime-showcase>

Twitter

You can follow PrimeFaces on twitter using **@primefaces**.

14. FAQ

1. Who develops PrimeFaces?

PrimeFaces is developed and maintained by Prime Technology, a Turkish software development company specialized in Agile Software Development, JSF and Java EE. PrimeFaces lead Cagatay Civici is also the PMC member of Apache MyFaces and committer of Atmosphere Framework.

2. How can I get support?

Support forum is the main area to ask for help, it's publicly available and free registration is required before posting. Please do not email the developers of PrimeFaces directly and use support forum instead.

3. Is enterprise support available?

Yes, enterprise support is also available. Please visit support page on PrimeFaces homepage for more information.

<http://primefaces.prime.com.tr/en/support.html>

4. I'm using x component library in my project, can primefaces be compatible?

Compatibility is a major goal of PrimeFaces and we aim to be compatible with major component suites like Trinidad, Tomahawk and RichFaces.

5. Where is the source for the example demo applications?

Source code of demo applications are in the svn repository of PrimeFaces at /examples/trunk folder.

6. With facelets some components like charts do not work in Safari or Chrome but there's no problem with Firefox.

The common reason is the response mimeType when using with PrimeFaces with *facelets*. You need to make sure responseType is "text/html". With facelets you can use the <f:view contentType="text/html"> to enforce this setting.

7. Where can I get an unreleased snapshot?

Periodically snapshot of a future release is deployed to <http://repository.prime.com.tr>.

8. What is the license PrimeFaces have?

PrimeFaces is free to use and licensed under Apache License V2

9. Can I use PrimeFaces in a commercial software?

Yes, Apache V2 License is a commercial friendly library.