

PrimeFaces

Trabajo SDI

Pablo Bravo Mediavilla UO223531

Álvaro Panizo Romano UO227748

Íñigo Llaneza Aller UO206367

Javier Iglesias García UO232562

Contenido

Introducción	6
Ajax Core	6
Listener	6
Process.....	6
Dropdown	6
Selector.....	6
Poll	7
Status.....	7
Atributos.....	8
Ejemplo de código.....	8
Events.....	9
Validation.....	9
Partial Submit.....	9
Fragment.....	9
Input.....	9
Autocompletar.....	10
Atributos.....	10
Ejemplo de código.....	10
BooleanButton.....	10
Calendar	11
Atributos.....	11
Ejemplo de código.....	11
SelectOneMenu.....	12
InputText.....	12
SelectOneListbox	12
Atributos.....	12
Ejemplo de código.....	12
SelectRadio	13

Password.....	13
Atributos.....	13
Ejemplo de código.....	13
Rating.....	14
InputTextArea.....	14
Atributos.....	14
Ejemplo de código.....	14
Editor.....	15
InputMask.....	15
Keyboard.....	15
Atributos.....	15
Ejemplo de código.....	15
Button.....	16
Atributos.....	16
Ejemplo de código.....	16
Data	17
Carousel	17
Atributos.....	17
DataList.....	18
Atributos.....	19
Ejemplo de código.....	19
DataTable.....	19
Atributos.....	20
Ejemplo de código.....	20
PickList	20
GMap.....	21
Atributos.....	21
Paneles	22
Wizard.....	22
TabView.....	23

Fieldset	25
OutputPanel	26
Overlay	28
LightBox	28
Confirm Dialog	29
Tooltip.....	31
Overlay Panel	33
Menu.....	35
BreadCrumb.....	35
Context Menu	36
Dock.....	37
MegaMenu.....	38
Steps.....	39
Chart.....	41
Line.....	41
Live	42
Export.....	43
Interactive.....	44
Messages.....	45
Growl	45
Atributos.....	46
Ejemplo de código.....	46
Messages	47
Atributos.....	48
Ejemplo de código.....	48
Multimedia.....	49
Barcode	49
Atributos.....	50
Ejemplo de código.....	51
ContentFlow	51

Atributos.....	51
Ejemplo de código.....	51
Galleria.....	52
Atributos.....	52
Media	53
Atributos.....	54
Ejemplo de código.....	54
ImageSwitch	54
Atributos.....	55
Ejemplo de código.....	55
ImageCompare.....	55
Atributos.....	56
Ejemplo de código.....	56
Cropper	57
Atributos.....	57
GraphicImage.....	58
Atributos.....	59
Ejemplo de código.....	60
PhotoCam.....	60
Atributos.....	60
Ejemplo de Código.....	61
File.....	62
FileUpload	62
Atributos.....	63
Ejemplo de Código.....	65
Download	65
Atributos.....	65
Ejemplo de código.....	65
DragDrop.....	66
Draggable.....	66

Atributos.....	66
Ejemplo de código.....	67
DataTable.....	67
Atributos.....	69
Ejemplo de código.....	71
DataGrid.....	72
Ejemplo de código.....	72
Client side validation.....	73
Basico	73
Custom	73
Bean.....	73
Event.....	73
Ejemplo de código.....	74
Visualización	74
Dialog framework.....	74
Ejemplo de código.....	74
Visualización	75
Misc	75
FontAwesome	75
Ejemplo de código.....	75
Visualización	76
IdleMonitor	76
Ejemplo de código.....	76
Visualización	77
ThemeSwitcher	78
Ejemplo de código.....	78
Visualización	79

Introducción

PrimeFaces es una librería de código abierto que se utiliza con JavaServer Faces, contiene una serie de componentes que ayudan a la creación de páginas Web. Algunas de sus características principales es que se puede instalar fácilmente y ocupa muy poco espacio. Además nos permite adaptar nuestras páginas a dispositivos móviles con gran facilidad.

En este documento haremos una descripción de todos los componentes de los que dispone PrimeFaces. Veremos un resumen del funcionamiento de cada uno, una lista de sus atributos y un ejemplo de código.

Ajax Core

Mediante Ajax podemos realizar cambios sobre nuestra página Web sin necesidad de recargarla. Algunos de los componentes sobre los que podemos utilizar Ajax con PrimeFaces son los siguientes:

Listener

Permite que un método de Java sea invocado con Ajax usando la opción "listener".

Process

Permite un procesamiento parcial de alguna de los componentes de una Web ignorando otros.

Dropdown

Con este componente podemos crear una dependencia entre varios desplegados.

Selector

Este componente permite usar la API jQuery Selector para hacer referencias a componentes.

PFS Form



Text 1: Validation Error: Value is required.
Text 2: Validation Error: Value is required.

Text 1: *



Text 1: Validation Error: Value is required.

Text 2: *



Text 2: Validation Error: Value is required.

Text 3: *

All Forms

Last Form

All Panels

Inputs of Panel

Inputs Except Select

Poll

Este componente realiza llamadas a métodos periódicamente.

Status

Mediante AjaxStatus se proporciona información acerca de la solicitud Ajax que se está realizando en ese momento.

Atributos

Name	Default	Type	Description
id	null	String	Unique identifier of the component.
rendered	true	Boolean	Boolean value to specify the rendering of the component.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
onstart	null	String	Client side callback to execute after ajax requests start.
oncomplete	null	String	Client side callback to execute after ajax requests complete.
onsuccess	null	String	Client side callback to execute after ajax requests completed succesfully.
onerror	null	String	Client side callback to execute when an ajax request fails.
style	null	String	Inline style of the component.
styleClass	null	String	Style class of the component.
widgetVar	null	String	Name of the client side widget.

Ejemplo de código

```
<p:ajaxStatus style="display:block;margin-bottom:2em;height:24px;">
  <f:facet name="default">
    <h:outputText value="Status: StandBy" />
  </f:facet>

  <f:facet name="start">
    <p:graphicImage name="/demo/images/ajaxloadingbar.gif" />
  </f:facet>

  <f:facet name="complete">
    <h:outputText value="Status: Completed" />
  </f:facet>
</p:ajaxStatus>

<p:ajaxStatus onstart="PF('statusDialog').show()"
onsuccess="PF('statusDialog').hide()" />

<p:dialog widgetVar="statusDialog" modal="true" draggable="false"
closable="false" resizable="false" showHeader="false">
  <p:graphicImage name="/demo/images/ajaxloadingbar.gif" />
```

```
</p:dialog>
```

```
<h:form>  
    <p:commandButton value="Send" icon="ui-icon-refresh" />  
</h:form>
```

Events

Permite realizar eventos compatibles en los componentes.

Validation

Este componente permite realizar una comprobación en el servidor y actualizarse con el resultado.



The screenshot shows a web form with a 'New User' button at the top. Below it is a light blue message box with an information icon and the text 'Welcome erterte erterte'. The form contains two text input fields: 'Firstname: *' and 'Lastname: *', both containing the text 'erterte'. At the bottom is a 'Save' button with a checkmark icon.

Partial Submit

Permite reducir el tráfico de la red al cargar una página. En páginas grandes con muchos componentes de entrada, carga en primer lugar los más rápidos.

Fragment

Permite fragmentar componentes y cargar la parte que se desee.

Input

Mediante los inputs podemos introducir datos en una página Web, con PrimeFaces podemos personalizar los inputs de múltiples formas para mejorar la interacción:

Autocompletar

Este componente permite mostrar sugerencias para auto completar una palabra que se está escribiendo en una página Web.

Atributos

Name	Default	Type	Description
id	null	String	Unique identifier of the component.
rendered	true	Boolean	Boolean value to specify the rendering of the component.
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	Object	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.

Ejemplo de código

```
<p:outputLabel value="Simple:" for="acSimple" />
    <p:autoComplete id="acSimple" value="#{autoCompleteView.txt1}"
completeMethod="#{autoCompleteView.completeText}" />

    <p:outputLabel value="Min Length (3):" for="acMinLength" />
    <p:autoComplete id="acMinLength" minQueryLength="3"
value="#{autoCompleteView.txt2}"
completeMethod="#{autoCompleteView.completeText}" effect="fade" />

    <p:outputLabel value="Delay(1000):" for="acDelay" />
    <p:autoComplete id="acDelay" queryDelay="1000"
value="#{autoCompleteView.txt3}"
completeMethod="#{autoCompleteView.completeText}" effect="blind" />

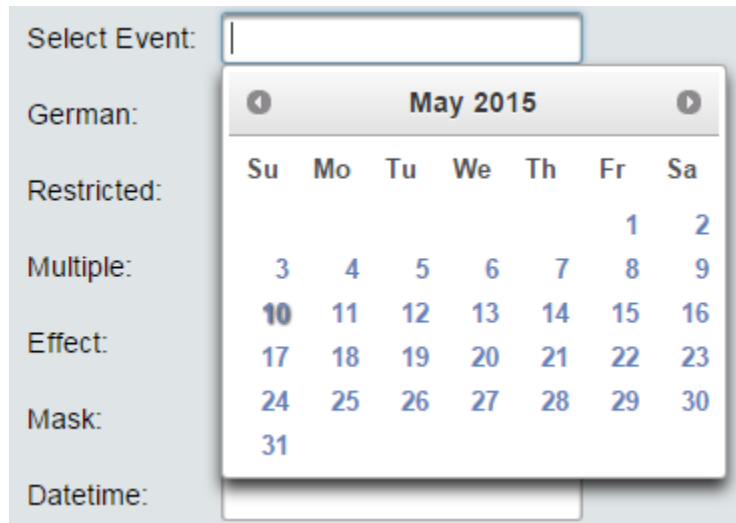
    <p:outputLabel value="Max Results(5):" for="acMaxResults" />
    <p:autoComplete id="acMaxResults" maxResults="5"
value="#{autoCompleteView.txt4}"
completeMethod="#{autoCompleteView.completeText}" />
```

BooleanButton

Permite crear un botón con un valor boolean que cambie al pulsarlo.

Calendar

Este componente es un input que, al seleccionarlo, nos muestra un calendario y nos permite seleccionar una fecha en él para no tener que escribirla a mano.



Atributos

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	true	Boolean	Boolean value to specify the rendering of the component.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	java.util.Date	Value of the component
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	When set true, process validations logic is executed at apply request values phase for this component.
required	false	Boolean	Marks component as required

Ejemplo de código

```
<p:outputLabel for="inline" value="Inline:" />
    <p:calendar id="inline" value="#{calendarView.date1}" mode="inline"
/>
```

SelectOneMenu

Muestra un menú con diferentes opciones para seleccionar, permite elegir el tipo de menú que deseemos: Basic, Grouping, Editable, Advanced...

InputText

Campo de texto que se puede personalizar el estilo.

SelectOneListbox

Componente que muestra una lista de opciones que se pueden seleccionar.

Atributos

Name	Default	Type	Description
id	null	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 referring to a List.
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	When set true, process validations logic is executed at apply request values phase for this component.
required	false	Boolean	Marks component as required
validator	null	MethodExpr	A method expression that refers to a method validationg the input

Ejemplo de código

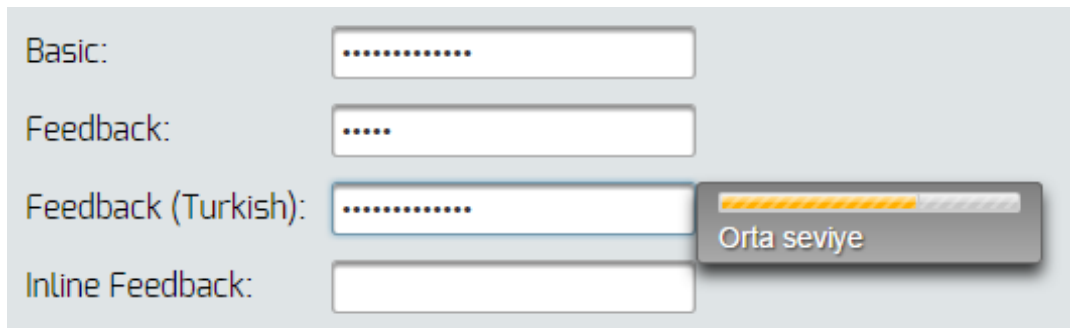
```
<p:selectOneListbox id="basic" value="#{selectOneView.option}">
    <f:selectItem itemLabel="Option 1" itemValue="1" />
    <f:selectItem itemLabel="Option 2" itemValue="2" />
    <f:selectItem itemLabel="Option 3" itemValue="3" />
</p:selectOneListbox>
```

SelectRadio

Permite elegir un elemento de una colección.

Password

Es un campo para introducir contraseñas al que se le puede añadir un indicador de seguridad, para saber cómo de segura es la contraseña que hemos introducido.



The image shows a user interface for a password field. It includes four labels: 'Basic:', 'Feedback:', 'Feedback (Turkish):', and 'Inline Feedback:'. Each label is followed by a text input field. The 'Basic:' field contains eight dots. The 'Feedback:' field contains four dots. The 'Feedback (Turkish):' field contains eight dots. To the right of the 'Feedback (Turkish):' field is a security level indicator consisting of a horizontal bar with a yellow-to-orange gradient and the text 'Orta seviye' below it. The 'Inline Feedback:' field is empty.

Atributos

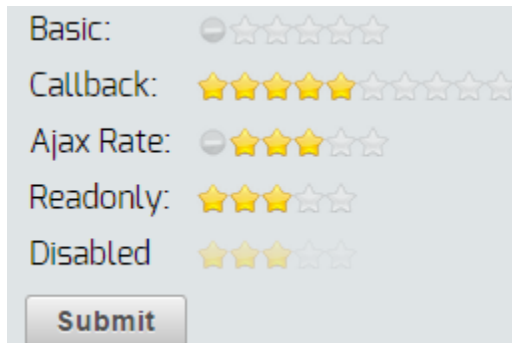
Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	true	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Value of the component than can be either an EL expression 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	When set true, process validations logic is executed at apply request values phase for this component.
required	false	boolean	Marks component as required
validator	null	Method Expr	A method expression that refers to a method validating the input.

Ejemplo de código

```
<p:password id="inlineFeedback" value="#{passwordView.password4}"  
feedback="true" inline="true"/>
```

Rating

Es un componente para introducir valoraciones mediante estrellas.



InputTextArea

Permite crear componentes textarea en los que se pueden añadir opciones como auto completado, cambiar el tamaño, contadores, temas...

Atributos

Name	Default	Type	Description
id	null	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 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	When set true, process validations logic is executed at apply request values phase for this component.
required	false	Boolean	Marks component as required

Ejemplo de código

```
<p:inputTextarea rows="10" cols="50"
completeMethod="#{inputTextAreaView.completeArea}"
queryDelay="750" minQueryLength="4"/>
```

Editor

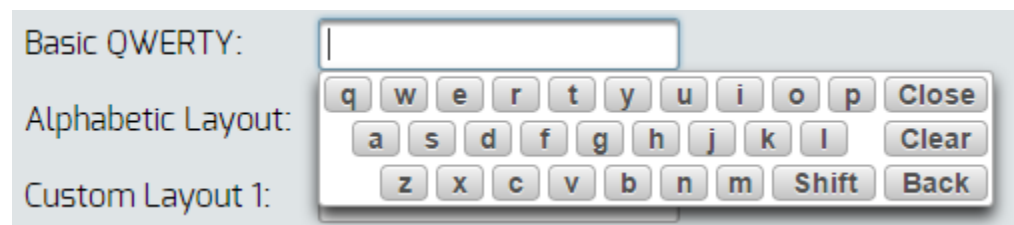
Podemos introducir en nuestra Web un editor de texto con múltiples opciones para editar el texto, con barra de herramientas y posibilidad de configurar los controles.

InputMask

Permite mostrar el formato en el que debemos introducir el texto.

Keyboard

Permite introducir un teclado en la pantalla al seleccionar un elemento de introducción de texto.



Atributos

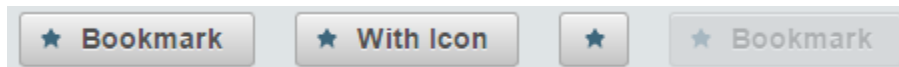
Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	true	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side <code>UICo</code> mponent instance 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/Str ing	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	When set true, process validations logic is executed at apply request values phase for this component.
required	false	Boolean	Marks component as required

Ejemplo de código

```
<p:keyboard id="default" value="#{keyboardView.value1}"/>
```

Button

Este componente es una extensión del componente de JSF "h:button". Con esta opción podemos modificar múltiples características del botón que de la otra manera no podíamos, ponerle un icono, cambiar el skin...



Atributos

Name	Default	Type	Description
id	null	String	Unique identifier of the component.
rendered	true	Boolean	Boolean value to specify the rendering of the component.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean.
widgetVar	null	String	Name of the client side widget.
value	null	Object	Value of the component than can be either an EL expression of a literal text.
outcome	null	String	Used to resolve a navigation case.
includeViewParams	false	Boolean	Whether to include page parameters in target URI
fragment	null	String	Identifier of the target page which should be scrolled to.
disabled	false	Boolean	Disables button.
accesskey	null	String	Access key that when pressed transfers focus to button.
alt	null	String	Alternate textual description.
dir	null	String	Direction indication for text that does not inherit directionality. Valid values are LTR and RTL.
image	null	String	Style class for the button icon. (deprecated: use icon)
lang	null	String	Code describing the language used in the generated markup

Ejemplo de código

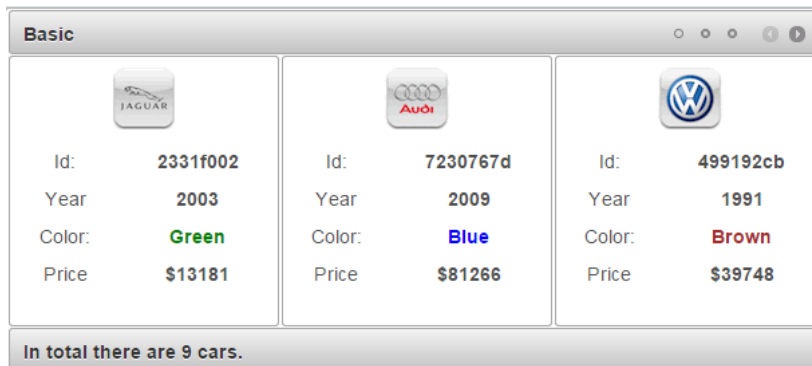
```
<p:button outcome="productDetail" value="Bookmark" icon="ui-icon-star"
disabled="true">
    <f:param name="productId" value="40" />
</p:button>
```

Data

Con primefaces tenemos múltiples opciones de mostrar datos en una página Web:

Carousel

Con este componente podemos mostrar datos usando un slide al que le podemos dar la forma que queramos (básico, tabla, avanzado).



Atributos

Name	Default	Type	Description
id	null	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	A value expression that refers to a collection
var	null	String	Name of the request scoped iterator
numVisible	3	Integer	Number of visible items per page
firstVisible	0	Integer	Index of the first element to be displayed
widgetVar	null	String	Name of the client side widget.
circular	false	Boolean	Sets continuous scrolling
vertical	false	Boolean	Sets vertical scrolling
autoPlayInterval	0	Integer	Sets the time in milliseconds to have Carousel start

Ejemplo de código

```
<p:carousel value="#{carouselView.cars}" headerText="Basic" var="car"
itemStyleClass="carItem">
    <h:panelGrid columns="2" style="width:100%" cellpadding="5"
columnClasses="label,value">
        <f:facet name="header">
            <p:graphicImage name="demo/images/car/#{car.brand}.gif"/>
        </f:facet>

        <h:outputText value="Id:" />
        <h:outputText value="#{car.id}" />

        <h:outputText value="Year:" />
        <h:outputText value="#{car.year}" />

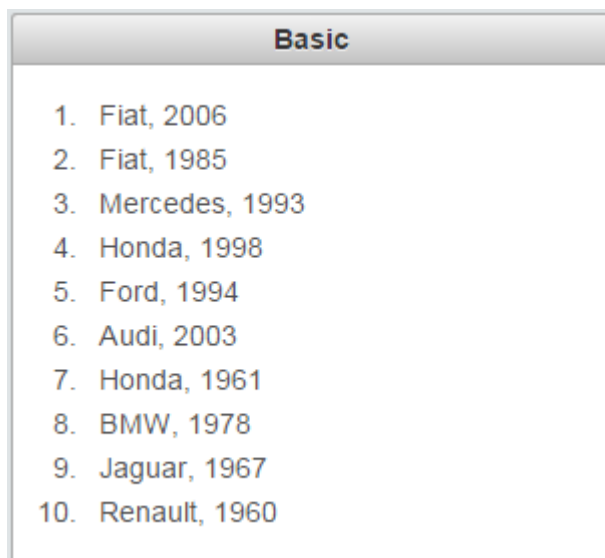
        <h:outputText value="Color:" />
        <h:outputText value="#{car.color}" style="color:#{car.color}"/>

        <h:outputText value="Price:" />
        <h:outputText value="$#{car.price}" />
    </h:panelGrid>

    <f:facet name="footer">
        In total there are #{fn:length(carouselView.cars)} cars.
    </f:facet>
</p:carousel>
```

DataList

Con este componente podemos mostrar los datos en un layout con forma de lista.



Atributos

Name	Default	Type	Description
id	null	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	Data to display.
var	null	String	Name of the request-scoped variable used to refer each data.
rows	null	Integer	Number of rows to display per page.
first	0	Integer	Index of the first row to be displayed

Ejemplo de código

```
<p:dataList value="#{dataListView.cars1}" var="car" type="ordered">
    <f:facet name="header">
        Basic
    </f:facet>
    #{car.brand}, #{car.year}
</p:dataList>
```

DataTable

Este componente permite mostrar los datos en una tabla tabulada.

Id	Year	Brand	Color
2c287077	1979	Fiat	Blue
51c4428f	2009	Ford	Green
441f6d52	1983	Volkswagen	Maroon
2778129a	1965	Ford	Silver
352fb8a0	1964	BMW	Orange
04de6144	1982	Mercedes	Black
ec7cd822	1989	Volkswagen	Brown
1335a630	1966	Renault	Black
059c4db0	2007	Volvo	Yellow
95fe384d	1965	BMW	Blue

Atributos

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	false	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	Data to display.
var	null	String	Name of the request-scoped variable used to refer each data.
rows	null	Integer	Number of rows to display per page.
first	0	Integer	Index of the first row to be displayed

Ejemplo de código

```
<p:dataTable var="car" value="#{dtBasicView.cars}">
  <p:column headerText="Id">
    <h:outputText value="#{car.id}" />
  </p:column>

  <p:column headerText="Year">
    <h:outputText value="#{car.year}" />
  </p:column>

  <p:column headerText="Brand">
    <h:outputText value="#{car.brand}" />
  </p:column>

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

PickList

Este componente permite seleccionar datos de una lista inicial y añadirlos a otra.

San Francisco

London

Istanbul

Berlin

Barcelona

Rome

→

→|

←

|←

Paris

GMap

Con este componente podemos mostrar en nuestra Web un mapa con una posición.

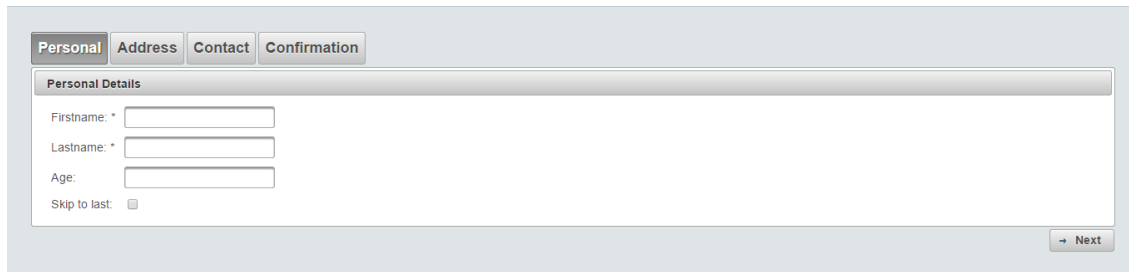
Atributos

Name	Default	Type	Description
style	null	String	Inline style of the map container.
styleClass	null	String	Style class of the map container.
type	null	String	Type of the map.
center	null	String	Center point of the map.
zoom	8	Integer	Defines the initial zoom level.
streetView	false	Boolean	Controls street view support.
disableDefaultUI	false	Boolean	Disables default UI controls
navigationControl	true	Boolean	Defines visibility of navigation control.
mapTypeControl	true	Boolean	Defines visibility of map type control.
draggable	true	Boolean	Defines draggability of map.
disabledDoubleClickZoom	false	Boolean	Disables zooming on mouse double click.
onPointClick	null	String	Javascript callback to execute when a point on map is clicked.
fitBounds	true	Boolean	Defines if center and zoom should be calculated automatically to contain all markers on the

Paneles

Los componentes de paneles son útiles para agrupar la información que mostremos en nuestra web, así como los formularios, logrando no saturar con demasiado contenido al visitante.

Wizard



The image shows a wizard interface with four steps: Personal, Address, Contact, and Confirmation. The 'Personal' step is currently selected and active. It contains a form with the following fields: 'Firstname: *' with a text input, 'Lastname: *' with a text input, 'Age: ' with a text input, and 'Skip to last: ' with a checkbox. A 'Next' button is located at the bottom right of the form.

Con este componente podemos agrupar el contenido de largos formularios por categorías. De esta forma nuestro usuario no se agobiará si tiene que rellenar muchos campos. Una de sus ventajas es la validación por partes. Es decir, en la imagen mostrada por ejemplo, no te dejará acceder a rellenar “Address” sin antes haber validado correctamente los campos de “Personal”.

Name	Default	Type	Description
id	null	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	String	Id of the current step in flow
style	null	String	Style of the main wizard container element.
styleClass	null	String	Style class of the main wizard container element.
flowListener	null	MethodExpr	Server side listener to invoke when wizard attempts to go forward or back.
showNavBar	true	Boolean	Specifies visibility of default navigator arrows.
showStepStatus	true	Boolean	Specifies visibility of default step title bar.

El uso de esta etiqueta es bastante sencillo. Siguiendo el ejemplo mostrado en la imagen, su correspondiente .xhtml sería el siguiente:

```

<h:form>

    <p:growl id="growl" sticky="true" showDetail="true"/>

    <p:wizard flowListener="#{userWizard.onFlowProcess}">
        <p:tab id="personal" title="Personal">
            <p:panel header="Personal Details">
                <p:messages />
                <h:panelGrid columns="2" columnClasses="label, value">
                    <h:outputText value="Firstname: *" />
                    <p:inputText value="#{userWizard.user.firstname}" required="true"
label="Firstname"/>

                    <h:outputText value="Lastname: *" />
                    <p:inputText value="#{userWizard.user.lastname}" required="true"
label="Lastname"/>

                    <h:outputText value="Age: " />
                    <p:inputText value="#{userWizard.user.age}" />

                    <h:outputText value="Skip to last: " />
                    <h:selectBooleanCheckbox value="#{userWizard.skip}" />
                </h:panelGrid>
            </p:panel>
        </p:tab>

        <p:tab id="address" title="Address">
            <p:panel header="Address Details">
                <p:messages />
                <h:panelGrid columns="2" columnClasses="label, value">
                    <h:outputText value="Street: " />
                    <p:inputText value="#{userWizard.user.street}" />

                    <h:outputText value="Postal Code: " />
                    <p:inputText value="#{userWizard.user.postalCode}" />

                    <h:outputText value="City: " />
                    <p:inputText value="#{userWizard.user.city}" />

                    <h:outputText value="Skip to last: " />
                    <h:selectBooleanCheckbox value="#{userWizard.skip}" />
                </h:panelGrid>
            </p:panel>
        </p:tab>

        <p:tab id="contact" title="Contact">
            <p:panel header="Contact Information">
                <p:messages />
                <h:panelGrid columns="2" columnClasses="label, value">
                    <h:outputText value="Email: *" />
                    <p:inputText value="#{userWizard.user.email}" required="true"
label="Email"/>

                    <h:outputText value="Phone: " />
                    <p:inputText value="#{userWizard.user.phone}" />

                    <h:outputText value="Additional Info: " />
                    <p:inputText value="#{userWizard.user.info}" />
                </h:panelGrid>
            </p:panel>
        </p:tab>

        <p:tab id="confirm" title="Confirmation">
            <p:panel header="Confirmation">
                <h:panelGrid id="confirmation" columns="3"
columnClasses="grid,grid,grid">
                    <h:panelGrid columns="2" columnClasses="label, value">
                        <h:outputText value="Firstname: *" />
                        <h:outputText value="#{userWizard.user.firstname}"
styleClass="outputLabel"/>

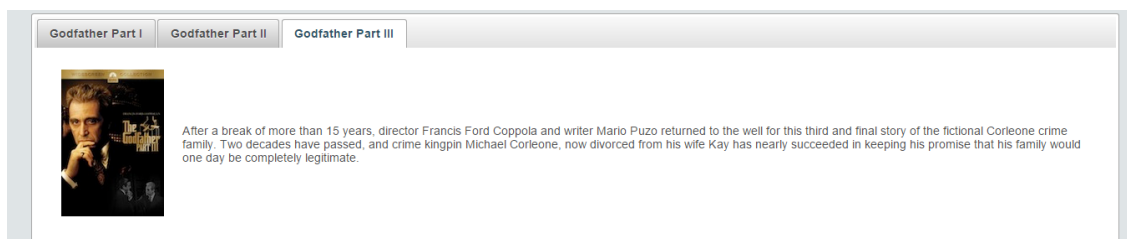
                        <h:outputText value="Lastname: " />
                        <h:outputText value="#{userWizard.user.lastname}"
styleClass="outputLabel"/>

                        <h:outputText value="Age: " />

```

Como podemos observar, tan solo sería necesario especificar las agrupaciones de los campos del formulario, así como sus condiciones de validación, encargándose PrimeFaces de realizar el resto del trabajo.

TabView



Esta etiqueta agrupa la información que antes mostraríamos como una lista en pestañas. Gracias a ella logramos reducir el contenido mostrado de cada vez al visitante, y dependiendo de los

parámetros escogidos, incluso el ancho de banda consumido por la página al cargar las vistas sólo cuando el usuario pinche en ellas.

Name	Default	Type	Description
id	null	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	Variable name of the client side widget.
activeIndex	0	Integer	Index of the active tab.
effect	null	String	Name of the transition effect.
effectDuration	null	String	Duration of the transition effect.
dynamic	false	Boolean	Enables lazy loading of inactive tabs.

Su funcionamiento al igual que el resto de componentes de este framework es muy sencillo:

```
<h:form id="form">

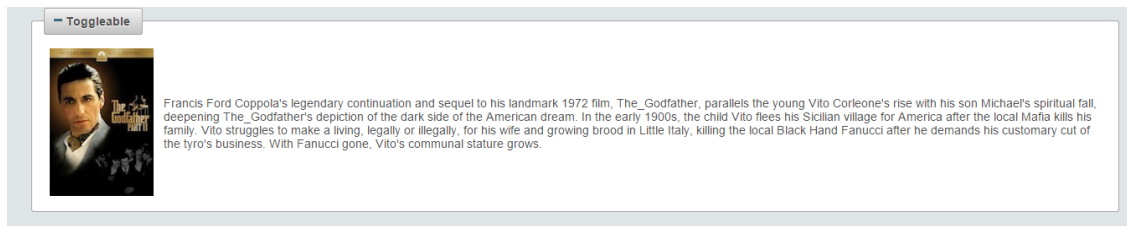
    <p:growl id="msgs" showDetail="true" />

    <h3 style="margin-top:0">Basic</h3>
    <p:tabView>
        <p:tab title="Godfather Part I">
            <h:panelGrid columns="2" cellpadding="10">
                <p:graphicImage name="demo/images/godfather/godfather1.jpg" />
                <h:outputText
                    value="The story begins as Don Vito Corleone..." />
            </h:panelGrid>
        </p:tab>
        <p:tab title="Godfather Part II">
            <h:panelGrid columns="2" cellpadding="10">
                <p:graphicImage name="demo/images/godfather/godfather2.jpg" />
                <h:outputText value="Francis Ford Coppola's legendary..." />
            </h:panelGrid>
        </p:tab>
        <p:tab title="Godfather Part III">
            <h:panelGrid columns="2" cellpadding="10">
                <p:graphicImage name="demo/images/godfather/godfather3.jpg" />
                <h:outputText value="After a break of more than 15 years..." />
            </h:panelGrid>
        </p:tab>
    </p:tabView>

</h:form>
```

Simplemente, instanciamos el conjunto de pestañas con `<p:tabView>`, y definimos cada una con `<p:tab>`, pudiendo añadir el contenido que queramos dentro de ellas.

Fieldset



Con `<p:fieldset>` logramos resaltar cierto contenido, del mismo modo que podemos darla opción al usuario de ocultarlo si este lo considerase necesario. Un buen ejemplo aparte del aquí mostrado, podría ser mostrar código al usuario convenientemente formateado dentro de un tutorial.

Name	Default	Type	Description
id	null	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	Name of the client side widget.
legend	null	String	Title text.
style	null	String	Inline style of the fieldset.
styleClass	null	String	Style class of the fieldset.
toggleable	false	Boolean	Makes content toggleable with animation.
toggleSpeed	500	Integer	Toggle duration in milliseconds.
collapsed	false	Boolean	Defines initial visibility state of content.

El funcionamiento de esta etiqueta no puede ser más simple:

```
<p:fieldset legend="Toggleable" toggleable="true" toggleSpeed="500">
  <p:ajax event="toggle" listener="#{fieldsetView.handleToggle}" update="msgs" />
  <h:panelGrid columns="2" cellpadding="5">
    <p:graphicImage name="demo/images/godfather/godfather2.jpg" />
    <h:outputText value="Francis Ford Coppola's legendary continuation and
sequel to his landmark 1972 film, The Godfather, parallels the young Vito Corleone's
rise with his son Michael's spiritual fall, deepening The Godfather's depiction of the
dark side of the American dream.
In the early 1900s, the child Vito flees his Sicilian village
for America after the local Mafia kills his family. Vito struggles to make a living,
legally or illegally, for his wife and growing brood in Little Italy,
killing the local Black Hand Fanucci after he demands his
customary cut of the tyro's business. With Fanucci gone, Vito's communal stature
grows."></h:outputText>
  </h:panelGrid>
</p:fieldset>
```

El componente se instancia con `<p:fieldset>`, y con los atributos `toggleable`, y `toggleSpeed` definimos respectivamente si el usuario puede ocultar el contenido de este, y la velocidad de la animación respectivamente.

OutputPanel

Loaded after the panel becomes visible on page scroll.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Este componente permite agregar contenido después la descarga de la página. Gracias a él podemos reducir el ancho de banda consumido por el servidor, además de incrementar la velocidad de carga de la web. Es muy interesante, ya que los visitantes no siempre cuentan con una buena conexión, y pueden evitar la carga de partes de la web que en algunas circunstancias no les son relevantes.

Name	Default	Type	Description
id	null	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
layout	block	String	Shortcut for the css display property, valid values are block (default) and inline.
autoUpdate	false	Boolean	Enables auto update mode if set true.
deferred	false	Boolean	Deferred mode loads the contents after page load to speed up page load.
deferredMode	load	String	Defines deferred loading mode, valid values are "load" (after page load) and "visible" (once the panel is visible on scroll).
global	false	Boolean	Global ajax requests are listened by ajaxStatus component, setting global to false will not trigger ajaxStatus on deferred loading.
delay	none	String	Delay in milliseconds to wait before initiating a deferred loading, default value is "none".

Al igual que las anteriores su uso no tiene más complicación:

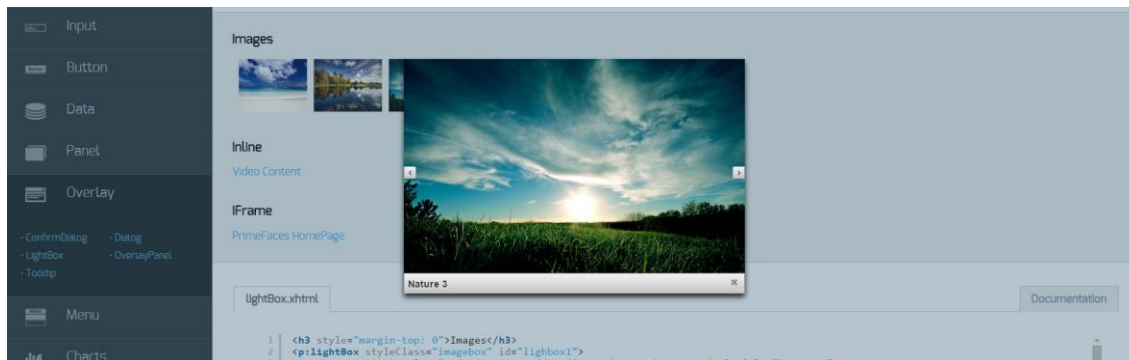
```
<h:form>
  <p:outputPanel deferred="true" deferredMode="visible">
    <h3>Loaded after the panel becomes visible on page scroll.</h3>
    <p>Lorem ipsum dolor sit amet...</p>
  </p:outputPanel>
</h:form>
```

Se instancia con `<p:outputPanel>`, y con los atributos `deferred`, y `deferredMode`, establecemos que se cargue más tarde que el resto de la página, y cuando exactamente ocurrirá esto repectivamente.

Overlay

En esta categoría nos encontramos componentes con los que generar contenido extra sin romper la estructura de la página, y que pueden ser útiles para hacer la web más amigable de cara al usuario, así como para mejorar su rendimiento y aspecto.

LightBox



Este elemento nos permite integrar contenido multimedia en un modal. Es una buena forma de mostrar contenido que ocupará un gran espacio en la pantalla, y que concentrará nuestra atención totalmente mientras es visualizado, como por ejemplo videos o imágenes en alta resolución.

Name	Default	Type	Description
id	null	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	Name of the client side widget.
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.
width	null	String	Width of the overlay in iframe mode.

Name	Default	Type	Description
height	null	String	Height of the overlay in iframe mode.
iframe	false	Boolean	Specifies an iframe to display an external url in overlay.
iframeTitle	null	String	Title of the iframe element.
visible	false	Boolean	Displays lightbox without requiring any user interaction by default.
onHide	null	String	Client side callback to execute when lightbox is displayed.
onShow	null	String	Client side callback to execute when lightbox is hidden.

En el siguiente ejemplo vemos su funcionamiento con una galería de imágenes:

```
<p:lightBox styleClass="imagebox" id="lighbox1">
  <h:outputLink value="../../resources/demo/images/nature/nature1.jpg" title="Nature
1">
    <h:graphicImage name="/demo/images/nature/nature1.jpg" id="nature1"
style="height: 77px; width: 100px" />
  </h:outputLink>

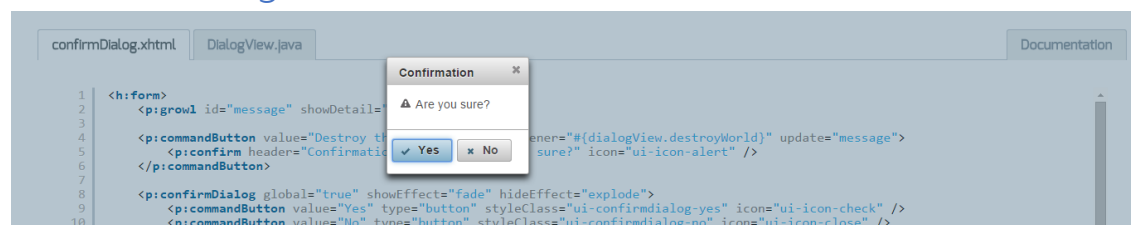
  <h:outputLink value="../../resources/demo/images/nature/nature2.jpg" title="Nature
2">
    <h:graphicImage name="/demo/images/nature/nature2.jpg" id="nature2"
style="height: 77px; width: 100px" />
  </h:outputLink>

  <h:outputLink value="../../resources/demo/images/nature/nature3.jpg" title="Nature
3">
    <h:graphicImage name="/demo/images/nature/nature3.jpg" id="nature3"
style="height: 77px; width: 100px" />
  </h:outputLink>

  <h:outputLink value="../../resources/demo/images/nature/nature4.jpg" title="Nature
4">
    <h:graphicImage name="/demo/images/nature/nature4.jpg" id="nature4"
style="height: 77px; width: 100px" />
  </h:outputLink>
</p:lightBox>
```

Para usarlo, como vemos sólo debemos instanciar el elemento con `<p:lightBox>`, y a continuación añadir los `<h:outputLink>` con el contenido a mostrar dentro de ellos.

Confirm Dialog



Con este componente, mostraremos al usuario un panel de confirmación al realizar alguna acción "peligrosa", como podría ser borrar su usuario de la web. Con él, evitaremos clicks

accidentales en botones, así como dar una oportunidad a que el usuario recapacite al realizar acciones que no tengan vuelta atrás.

Name	Default	Type	Description
id	null	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	Name of the client side widget.
message	null	String	Text to be displayed in body.
header	null	String	Text for the header.
severity	null	String	Message severity for the displayed icon.
width	auto	Integer	Width of the dialog in pixels
height	auto	Integer	Width of the dialog in pixels
style	null	String	Inline style of the dialog container.
styleClass	null	String	Style class of the dialog container

Name	Default	Type	Description
closable	true	Boolean	Defines if close icon should be displayed or not
appendTo	false	Boolean	Appends the dialog to the element defined by the given search expression.
visible	false	Boolean	Whether to display confirm dialog on load.
showEffect	null	String	Effect to use on showing dialog.
hideEffect	null	String	Effect to use on hiding dialog.
closeOnEscape	false	Boolean	Defines if dialog should hide on escape key.
dir	ltr	String	Defines text direction, valid values are <i>ltr</i> and <i>rtl</i> .
global	false	Boolean	When enabled, <code>confirmDialog</code> becomes a shared for other components that require confirmation.

En este ejemplo se muestra un ejemplo de botón que activaría este diálogo:

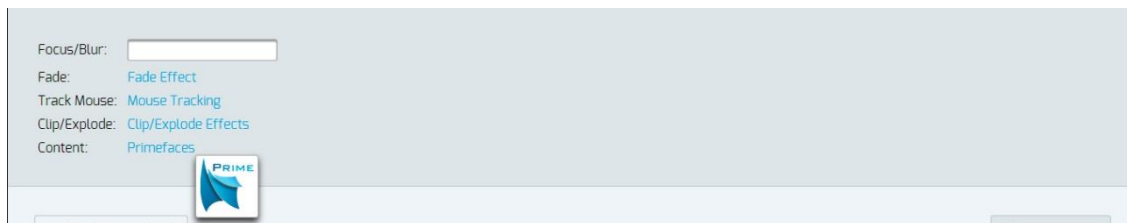
```
<h:form>
  <p:growl id="message" showDetail="true" />

  <p:commandButton value="Destroy the World"
    actionListener="#{dialogView.destroyWorld}" update="message">
    <p:confirm header="Confirmation" message="Are you sure?" icon="ui-icon-alert" />
  </p:commandButton>

  <p:confirmDialog global="true" showEffect="fade" hideEffect="explode">
    <p:commandButton value="Yes" type="button" styleClass="ui-confirmdialog-yes"
      icon="ui-icon-check" />
    <p:commandButton value="No" type="button" styleClass="ui-confirmdialog-no"
      icon="ui-icon-close" />
  </p:confirmDialog>
</h:form>
```

Si nos fijamos en el código necesario para agregarlo a nuestra web, necesitaremos un `commandButton` que al pulsarlo activará el diálogo de confirmación. Dentro este agregaremos el mensaje a mostrar al usuario con `<p:confirm>`, y finalmente un `<p:confirmDialog>` junto a este botón en el que agregaremos las distintas opciones a ofrecer al usuario, así como efectos o animaciones que queramos que este tenga. Este `<p:confirmDialog>` puede ser global y ser usado en toda la página, activado por distintos botones.

Tooltip



Este componente agregará funcionalidad a los ya conocidos tooltips, permitiéndonos agregar imágenes, efectos, o que se muevan junto con el cursor del ratón.

Name	Default	Type	Description
id	null	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 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
widgetVar	null	String	Name of the client side widget.
showEvent	mouseover	String	Event displaying the tooltip.
showEffect	fade	String	Effect to be used for displaying.
hideEvent	mouseout	String	Event hiding the tooltip.
hideEffect	fade	String	Effect to be used for hiding.
showDelay	150	Integer	Delay time to show tooltip in milliseconds.

Name	Default	Type	Description
hideDelay	0	Integer	Delay time to hide tooltip in milliseconds.
for	null	String	Component to attach the tooltip.
style	null	String	Inline style of the tooltip.
styleClass	null	String	Style class of the tooltip.
globalSelector	null	String	jquery selector for global tooltip, defaults to "a,input,button".
escape	true	Boolean	Defines whether html would be escaped or not.
trackMouse	false	Boolean	Tooltip position follows pointer on mousemove.
beforeShow	null	String	Client side callback to execute before tooltip is shown. Returning false will prevent display.
onHide	null	String	Client side callback to execute after tooltip is shown.
onShow	null	String	Client side callback to execute after tooltip is shown.

Una muestra del código es la siguiente:

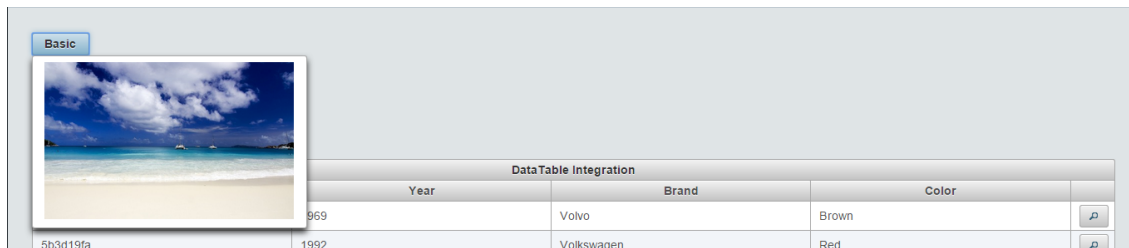
```

<h:panelGrid columns="3" cellpadding="5">
  <h:outputText value="Content: " />
  <h:outputLink id="content" value="#">
    <h:outputText value="Primefaces" />
  </h:outputLink>
  <p:tooltip id="toolTipContent" for="content">
    <p:graphicImage name="/demo/images/prime_logo.png" />
  </p:tooltip>

```

Agregarlo es muy fácil, y ayudará a hacer nuestra web más fácil de usar por el usuario. Con `<p:tooltip>` instanciamos el elemento, y con el atributo "for" le indicamos a que otro objeto pertenece.

Overlay Panel



Este componente nos permite mostrar contenido como imágenes, tablas, o simplemente texto al activar un "trigger" como un botón. Este contenido puede cargarse sólo al activar el correspondiente "trigger", reduciendo el tráfico consumido por la web, y acelerando la carga de la página en conexiones lentas.

Name	Default	Type	Description
id	null	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	Name of the client side widget.
style	null	String	Inline style of the panel.
styleClass	null	String	Style class of the panel.
for	null	String	Target component to display panel next to.
showEvent	click	String	Event on target to show the panel.

Name	Default	Type	Description
hideEvent	click	String	Event on target to hide the panel.
showEffect	null	String	Animation to display when showing the panel.
hideEffect	null	String	Animation to display when hiding the panel.
appendToBody	0	Boolean	When true, panel is appended to document body.
onShow	null	String	Client side callback to execute when panel is shown.
onHide	null	String	Client side callback to execute when panel is hidden.
my	left top	String	Position of the panel relative to the target.
at	left bottom	String	Position of the target relative to the panel.
dynamic	false	Boolean	Defines content loading mode.
dismissable	true	Boolean	When set true, clicking outside of the panel hides the overlay.
showCloseIcon	false	Boolean	Displays a close icon to hide the overlay, default is false.

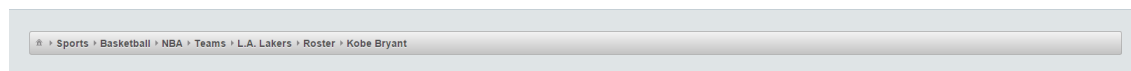
```
<p:commandButton id="imageBtn" value="Basic" type="button" />
  <p:overlayPanel id="imagePanel" for="imageBtn" hideEffect="fade">
    <p:graphicImage name="/demo/images/nature/nature1.jpg" width="300" />
  </p:overlayPanel>
```

Como podemos observar, el panel se instancia con una etiqueta <p:overlayPanel>, que contendrá el contenido a cargar, y en el atributo for el valor del correspondiente "trigger".

Menu

En esta categoría de elementos nos encontramos básicamente con distintos estilos de agrupaciones de links y/o botones, que nos ayudarán a indicar de manera más clara al usuario donde se encuentra y sus distintas opciones de navegación.

BreadCrumb



Este componente muestra al usuario en que sección de la web se encuentra, así como sus secciones padre en forma de links. Con esto logramos dar una mejor navegabilidad a la web.

Name	Default	Type	Description
id	null	String	Unique identifier of the component.
rendered	true	Boolean	Boolean value to specify the rendering of the component.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
widgetVar	null	String	Name of the client side widget.
model	null	MenuModel	MenuModel instance to create menus programmatically
style	null	String	Style of main container element.
styleClass	null	String	Style class of main container
homeDisplay	icon	String	Defines display mode of root link, valid values are "icon" default and "text".

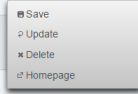
```
<p:breadcrumb>
  <p:menuitem value="Categories" url="#" />
  <p:menuitem value="Sports" url="#" />
  <p:menuitem value="Basketball" url="#" />
  <p:menuitem value="NBA" url="#" />
  <p:menuitem value="Teams" url="#" />
  <p:menuitem value="L.A. Lakers" url="#" />
  <p:menuitem value="Roster" url="#" />
  <p:menuitem value="Kobe Bryant" url="#" />
</p:breadcrumb>
```

Como vemos el código no tiene más complicación, simplemente es una etiqueta `<p:breadcrumb>` que contendrá dentro un listado de enlaces ordenados de mayor a menor jerarquía. Sería interesante generar estos enlaces siempre de manera dinámica y no a mano.

Context Menu

ContextMenu is an overlay menu display mainly displayed using right-click.

Right click to view the menu options.



Gracias a este elemento podremos sustituir el menú abierto por defecto al clicar con el botón derecho en cualquier parte de la web, mostrando un listado de acciones personalizadas.

Name	Default	Type	Description
id	null	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	Name of the client side widget.
for	null	String	Id of the component to attach to
style	null	String	Style of the main container element
styleClass	null	String	Style class of the main container element
model	null	MenuModel	Menu model instance to create menu programmatically.
nodeType	null	String	Specific type of tree nodes to attach to.
event	null	String	Event to bind contextMenu display, default is contextmenu aka right click.
beforeShow	null	String	Client side callback to execute before showing.
selectionMode	multiple	String	Defines the selection behavior, e.g "single" or "multiple".

Name	Default	Type	Description
targetFilter	null	String	Selector to filter the elements to attach the menu.

```
<p:contextMenu>
  <p:menuitem value="Save" actionListener="#{menuView.save}" update="messages"
icon="ui-icon-disk"/>
  <p:menuitem value="Update" actionListener="#{menuView.update}" update="messages"
icon="ui-icon-arrowrefresh-1-w"/>
  <p:menuitem value="Delete" actionListener="#{menuView.delete}" ajax="false"
icon="ui-icon-close"/>
  <p:menuitem value="Homepage" url="http://www.primefaces.org" icon="ui-icon-
extlink"/>
</p>
```

En el código podemos observar como tan sólo tenemos que instanciar un `<p:contextMenu>` con un listado de `<p:menuitem>` que serán los enlaces a nuestras acciones.

Dock



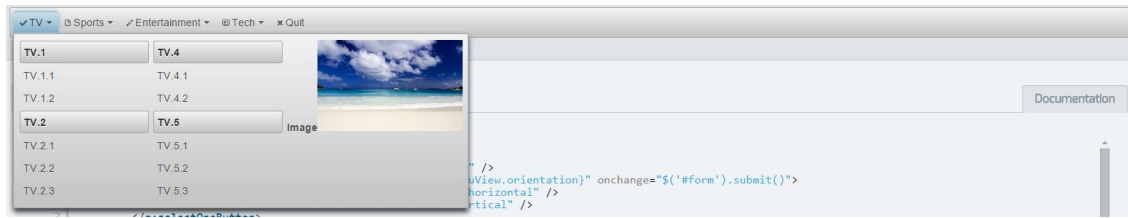
Este componente inspirado claramente en los mismos docks vistos en algunos sistemas Unix (Mac OSX por ejemplo), nos permite mostrar de una manera llamativa enlaces en nuestra web.

Name	Default	Type	Description
id	null	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
model	null	MenuModel	MenuModel instance to create menus programmatically
position	bottom	String	Position of the dock, <i>bottom</i> or <i>top</i> .
itemWidth	40	Integer	Initial width of items.
maxWidth	50	Integer	Maximum width of items.
proximity	90	Integer	Distance to enlarge.
halign	center	String	Horizontal alignment,
widgetVar	null	String	Name of the client side widget.

```
<p:dock position="bottom">
  <p:menuitem value="Home" icon="/resources/demo/images/dock/home.png" url="#"/>
  <p:menuitem value="Music" icon="/resources/demo/images/dock/music.png" url="#"/>
</p:dock>
```

Como ya estamos acostumbrados con este framework el código es muy intuitivo, tan solo teniendo que crear una etiqueta `<p:dock>` que contenga los `<p:menuItem>` con sus iconos y enlaces correspondientes.

MegaMenu



Con él podremos mostrar un completo menú con opciones de todo tipo en nuestra web, ya sea horizontal o vertical.

Name	Default	Type	Description
id	null	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	Name of the client side widget
model	null	MenuModel	MenuModel instance to create menus programmatically
style	null	String	Inline style of the component.
styleClass	null	String	Style class of the component.

Name	Default	Type	Description
autoDisplay	true	Boolean	Defines whether submenus will be displayed on mouseover or not. When set to false, click event is required to display.
activeIndex	null	Integer	Index of the active root menu to display as highlighted. By default no root is highlighted.
orientation	horizontal	String	Defines the orientation of the root menuitems, valid values are "horizontal" and "vertical".

```

<p:megaMenu orientation="#{megaMenuView.orientation}" style="margin-top:20px">
  <p:submenu label="TV" icon="ui-icon-check">
    <p:column>
      <p:submenu label="TV.1">
        <p:menuitem value="TV.1.1" url="#" />
        <p:menuitem value="TV.1.2" url="#" />
      </p:submenu>
      <p:submenu label="TV.2">
        <p:menuitem value="TV.2.1" url="#" />
        <p:menuitem value="TV.2.2" url="#" />
        <p:menuitem value="TV.2.3" url="#" />
      </p:submenu>
    </p:column>
  </p:submenu>

  <p:submenu label="Sports" icon="ui-icon-document">

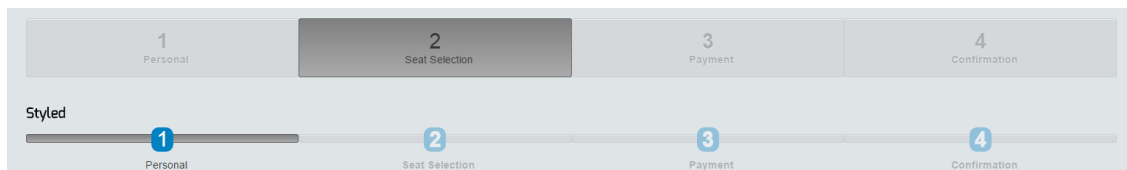
    <p:column>
      <p:submenu label="Sports.1">
        <p:menuitem value="Sports.1.1" url="#" />
        <p:menuitem value="Sports.1.2" url="#" />
      </p:submenu>
      <p:submenu label="Sports.2">
        <p:menuitem value="Sports.2.1" url="#" />
        <p:menuitem value="Sports.2.2" url="#" />
        <p:menuitem value="Sports.2.3" url="#" />
      </p:submenu>
    </p:column>

    <p:column>
      <p:submenu label="Sports.4">
        <p:menuitem value="Sports.4.1" url="#" />
        <p:menuitem value="Sports.4.2" url="#" />
      </p:submenu>
      <p:submenu label="Sports.5">
        <p:menuitem value="Sports.5.1" url="#" />
        <p:menuitem value="Sports.5.2" url="#" />
        <p:menuitem value="Sports.5.3" url="#" />
      </p:submenu>
    </p:column>
  </p:submenu>
</p:megaMenu>

```

El código esta vez es algo más complejo, dada la dimensión del elemento que estamos creando, pero sigue siendo muy legible y fácil de manejar. El procedimiento básico para crear este menú será agregar una etiqueta `<p:megaMenu>` con una serie de `<p:submenu>` dentro en los que podremos agregar más contenido y menús o enlaces.

Steps



Al igual que el componente wizard, los steps nos permiten subdividir un gran formulario en unos más pequeños. Muy útil para webs con carritos de la compra para indicar al usuario los pasos que le quedan hasta completar su pedido, e ir comprobando poco a poco que los datos introducidos son correctos.

Name	Default	Type	Description
id	null	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
model	null	MenuModel	MenuModel instance to build menu dynamically.
style	null	String	Inline style of the component.
styleClass	null	String	Style class of the component.
activeIndex	0	Integer	Index of the active tab.
widgetVar	null	String	Name of the client side widget.

```
<p:steps activeIndex="1">
  <p:menuitem value="Personal" />
  <p:menuitem value="Seat Selection" />
  <p:menuitem value="Payment" />
  <p:menuitem value="Confirmation" />
</p:steps>
```

Para agregarlo en nuestra web tendremos que crear un `<p:steps>` con una serie de `<p:menuitem>` dentro, y un atributo `activeIndex` que dirán en que parte nos encontramos.

Chart

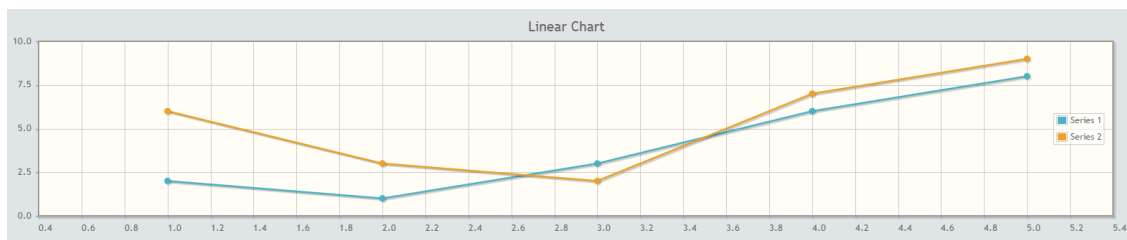
Aquí nos encontraremos básicamente gráficos de todo tipo, con los que mostrar información de manera vistosa, y más fácil de interpretar y comparar para nuestros visitantes.

En el caso de los charts los atributos son comunes para todos.

Name	Default	Type	Description
id	null	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
type	null	String	Type of the chart.
model	null	ChartModel	Model object of data and settings.
style	null	String	Inline style of the component.
styleClass	null	String	Style class of the component.
widgetVar	null	String	Name of the client side widget.
responsive	false	Boolean	In responsive mode, chart is redrawn when window is resized.

Veremos ahora algunos ejemplos de los gráficos que nos ofrece este framework con varias de sus opciones más interesantes.

Line



Clásico gráfico lineal, que será generado dentro de un elemento de tipo `<canvas>` con los datos suministrados por la aplicación. De necesitar otro tipo de gráfico, tan solo tendríamos que cambiar el atributo `type` por otro modelo que se ajustase más a nuestras necesidades.

```
<p:chart type="line" model="#{chartView.lineModel1}" style="height:300px;"/>
```

Esta vez ya no será suficiente con añadir contenido tan sólo en el .xhtml que necesitemos. Los gráficos necesitan tomar los datos de algún ManagedBean como veremos a continuación

```
@ManagedBean
public class ChartView implements Serializable {

    private LineChartModel lineModel1;

    @PostConstruct
    public void init() {
        createLineModels();
    }

    public LineChartModel getLineModel1() {
        return lineModel1;
    }

    private void createLineModels() {
        lineModel1 = initLinearModel();
        lineModel1.setTitle("Linear Chart");
        lineModel1.setLegendPosition("e");
        Axis yAxis = lineModel1.getAxis(AxisType.Y);
        yAxis.setMin(0);
        yAxis.setMax(10);
    }

    private CartesianChartModel initLinearModel() {
        CartesianChartModel model = new CartesianChartModel();

        LineChartSeries series1 = new LineChartSeries();
        series1.setLabel("Series 1");

        series1.set(1, 2);
        series1.set(2, 1);
        series1.set(3, 3);
        series1.set(4, 6);
        series1.set(5, 8);

        LineChartSeries series2 = new LineChartSeries();
        series2.setLabel("Series 2");

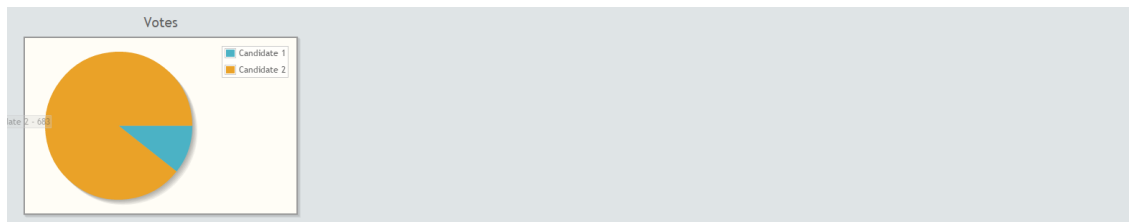
        series2.set(1, 6);
        series2.set(2, 3);
        series2.set(3, 2);
        series2.set(4, 7);
        series2.set(5, 9);

        model.addSeries(series1);
        model.addSeries(series2);

        return model;
    }
}
```

En nuestro ManagedBean correspondiente debemos añadir un objeto de tipo LineChartModel, e inicializarlo con los datos correspondientes, ya sea manual o automáticamente, así como configurar la escala y leyenda si fuese necesario.

Live



La peculiaridad de este tipo de gráfico es que se actualiza automáticamente cada cierto intervalo de tiempo, sin necesidad de recargar la página. Si los datos cambian rápidamente y es importante tenerlos siempre actualizados, este puede ser un gran componente en nuestra web.

```
<p:poll interval="3" update="votes" />
<p:chart id="votes" type="pie" model="#{chartView.livePieModel}"
style="width:400px;height:300px"/>
```

Para que el gráfico se actualice automáticamente deberemos añadir junto a la etiqueta `<p:chart>`, una `<p:poll>`, que referencie en el atributo `update` al gráfico, y en `interval` el tiempo en segundos para cada petición de datos nueva.

```
@ManagedBean
public class ChartView implements Serializable {

    private PieChartModel livePieModel;

    public PieChartModel getLivePieModel() {
        int random1 = (int) (Math.random() * 1000);
        int random2 = (int) (Math.random() * 1000);

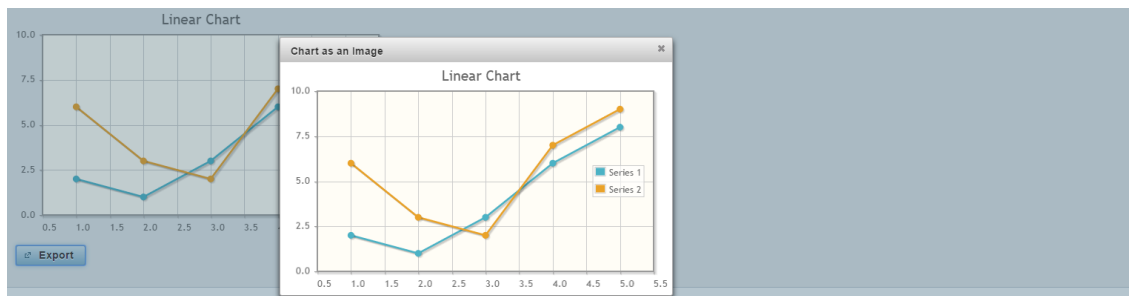
        livePieModel.getData().put("Candidate 1", random1);
        livePieModel.getData().put("Candidate 2", random2);

        livePieModel.setTitle("Votes");
        livePieModel.setLegendPosition("ne");

        return livePieModel;
    }
}
```

En el ejemplo, los datos simplemente son dos enteros aleatorios, que se generan en el ManagedBean correspondiente como podemos observar.

Export



Utilizando un poco de javascript podemos permitir al usuario descargar el gráfico que está visualizando en forma de imagen.

```

<p:chart type="line" value="#{chartView.lineModel1}" style="width:500px;height:300px"
widgetVar="chart"/>

<p:commandButton type="button" value="Export" icon="ui-icon-extlink"
onclick="exportChart()" />

<p:dialog widgetVar="dlg" showEffect="fade" modal="true" header="Chart as an Image"
resizable="false">
    <p:outputPanel id="output" layout="block" style="width:500px;height:300px"/>
</p:dialog>

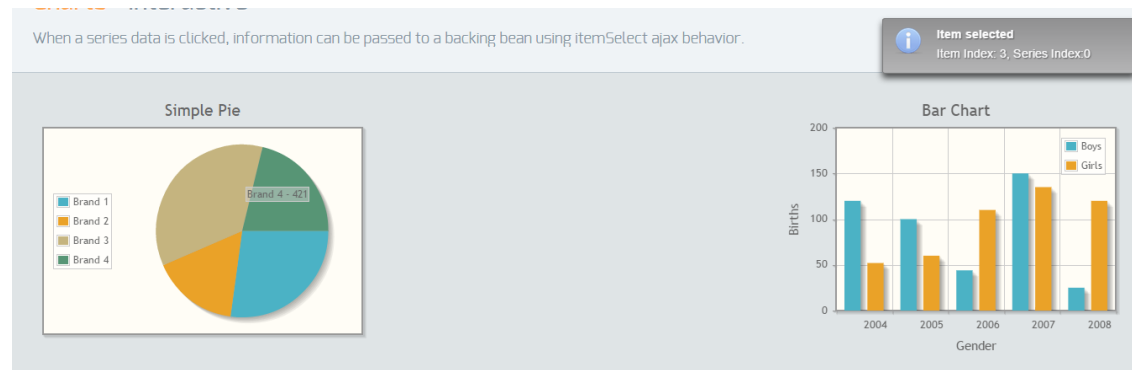
<script type="text/javascript">
function exportChart() {
    //export image
    $('#output').empty().append(PF('chart').exportAsImage());

    //show the dialog
    PF('dlg').show();
}
</script>

```

Como se puede ver en el código, aparte del básico `<p:chart>`, tendremos también que añadir un script, que exporte el gráfico a una imagen.

Interactive



Podemos permitir a nuestros visitantes interactuar con los gráficos clicando sobre ellos gracias a AJAX, actualizando otros componentes de la página o mostrando una información más detallada por ejemplo.

```

<p:growl id="growl" showDetail="true" />
<p:chart type="pie" model="#{chartView.pieModel1}" style="width:400px;height:300px">
    <p:ajax event="itemSelect" listener="#{chartView.itemSelect}" update="growl" />
</p:chart>

```

En nuestro .xhtml tendremos que añadir dentro del chart una etiqueta de tipo `<p:Ajax>`, a la cual le diremos el evento al que tiene que estar esperando, el método que se encargará de procesar los datos del evento, y finalmente si tiene que actualizar algún otro componente de la web. En el ejemplo mostrado se actualiza un componente de tipo `<p:growl>`.

```

@ManagedBean
public class ChartView implements Serializable {

    private PieChartModel pieModel1;

    @PostConstruct
    public void init() {
        createPieModels();
    }

    public PieChartModel getPieModel1() {
        return pieModel1;
    }

    private void createPieModels() {
        createPieModel1();
    }

    private void createPieModel1() {
        pieModel1 = new PieChartModel();

        pieModel1.set("Brand 1", 540);
        pieModel1.set("Brand 2", 325);
        pieModel1.set("Brand 3", 702);
        pieModel1.set("Brand 4", 421);

        pieModel1.setTitle("Simple Pie");
        pieModel1.setLegendPosition("w");
    }

    public void itemSelect(ItemSelectEvent event) {
        FacesMessage msg = new FacesMessage(FacesMessage.SEVERITY_INFO, "Item selected",
            "Item Index: " + event.getItemIndex() + ", Series Index:" +
            event.getSeriesIndex());

        FacesContext.getCurrentInstance().addMessage(null, msg);
    }
}

```

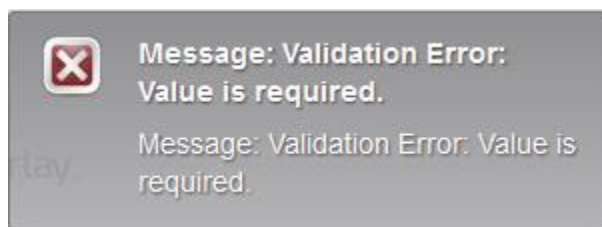
En el ManagedBean que se encargue de gestionar la petición AJAX, tendremos que crear un método que procese los datos recibidos y realice alguna acción con ellos. En este caso enviar un mensaje al usuario de que datos ha seleccionado.

Messages

Este framework nos permite la utilización de distintos tipos de mensajes que nos facilitarán mensajes que nos solucionarán la mayor parte de los problemas de una manera simple.

Growl

Este tipo de mensajes está basado en el widget de notificaciones de Mac y se utiliza para mostrar 'FacesMessages'.



Atributos

Name	Default	Type	Description
id	null	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.
globalOnly	false	Boolean	When true, only facesmessages without clientids are displayed.
life	6000	Integer	Duration in milliseconds to display non-sticky messages.
autoUpdate	false	Boolean	Specifies auto update mode.
redisplay	true	Boolean	Defines if already rendered messages should be displayed.
for	null	String	Name of associated key, takes precedence when used with globalOnly.
escape	true	Boolean	Defines whether html would be escaped or not.
severity	null	String	Comma separated list of severities to display only.

Ejemplo de código

```
<h:form>

    <p:growl id="growl" showDetail="true" sticky="true" />

    <p:panel header="Growl">

        <h:panelGrid columns="2" cellpadding="5">

            <p:outputLabel for="msg" value="Message:" />

            <p:inputText id="msg" value="#{growlView.message}"
required="true" />

        </h:panelGrid>

        <p:commandButton value="Save"
actionListener="#{growlView.saveMessage}" update="growl" />

    </p:panel>
```

</h:form>

Messages

Estos mensajes son extensiones de los componentes básicos de JSF para mensajes.

Nos sirven para comunicarnos de forma directa con el usuario.



Atributos

Name	Default	Type	Description
id	null	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	true	Boolean	Specifies if the summary of the FacesMessages should be displayed.
showDetail	false	Boolean	Specifies if the detail of the FacesMessages should be displayed.
globalOnly	false	String	When true, only facesmessages with no clientIds are displayed.
redisplay	true	Boolean	Defines if already rendered messages should be displayed
autoUpdate	false	Boolean	Enables auto update mode if set true.
for	null	String	Name of associated key, takes precedence when used with globalOnly.
escape	true	Boolean	Defines whether html would be escaped or not.
severity	null	String	Comma separated list of severities to display only.
closable	false	Boolean	Adds a close icon to hide the messages.
style	null	String	Inline style of the component.
styleClass	null	String	Style class of the component.
showIcon	true	Boolean	Defines if severity icons would be displayed.

Ejemplo de código

```
<p:tooltip />
```

```
<h:form>
```

```
    <p:messages id="messages" showDetail="true" autoUpdate="true"
closable="true" />
```

```
    <p:commandButton value="Info" actionListener="#{messagesView.info}" />
```

```
    <p:commandButton value="Warn" actionListener="#{messagesView.warn}" />
```

```
    <p:commandButton value="Error" actionListener="#{messagesView.error}" />
```

```
<p:commandButton value="Fatal" actionListener="#{messagesView.fatal}" />
</h:form>
```

Multimedia

Estos componentes permiten una mayor interacción gracias a la representación de contenido multimedia en el navegador. Este se presenta de una manera sencilla al usuario y también al programador mediante los siguientes elementos.

Barcode

Este componente nos permite mostrar desde códigos de barras hasta códigos QR de una manera sencilla.

Atributos

Name	Default	Type	Description
id	null	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.
type	null	String	Type of the barcode.
cache	true	Boolean	Controls browser caching mode of the resources.
format	svg	String	Format of the generated barcode, valid values are "svg" (default) and "png".
orientation	0	Integer	Orientation in terms of angle. (0, 90, 180, 270)
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
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

Ejemplo de código

```
<p:barcode value="0123456789" type="qr"/>
```

ContentFlow



Este component consta de una galería horizontal donde se pueden mostrar los componentes con una animación para el paso entre ellas.

Atributos

Name	Default	Type	Description
id	null	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	Name of the client side widget.
value	null	String	Collection of items to display.
var	null	String	Name of the iterator to display an item.
style	null	String	Inline style of the component.
styleClass	null	String	Style class of the component.

Ejemplo de código

```
<p:contentFlow value="#{imageView.images}" var="image">

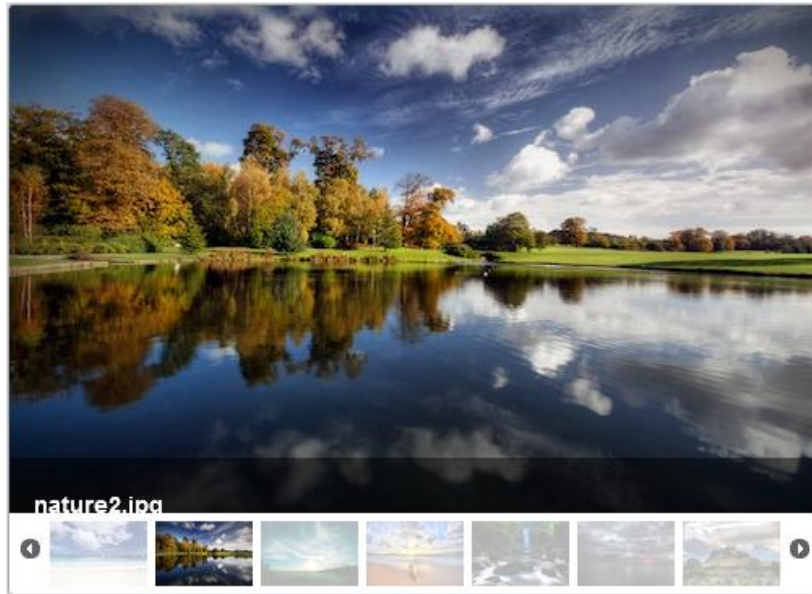
    <p:graphicImage name="demo/images/nature/#{image}" styleClass="content"
/>

    <div class="caption">#{image}</div>

</p:contentFlow>
```

Galleria

Este componente es parecido al anterior, pero nos permite mostrar la información como una galería de imágenes.



Atributos

Name	Default	Type	Description
id	null	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	Name of the client side widget.
value	null	Collection	Collection of data to display.
var	null	String	Name of variable to access an item in collection.
style	null	String	Inline style of the container element.
styleClass	null	String	Style class of the container element.
effect	fade	String	Name of animation to use.

effectSpeed	700	Integer	Duration of animation in milliseconds.
panelWidth	600	Integer	Width of the viewport.
panelHeight	400	Integer	Height of the viewport.
frameWidth	60	Integer	Width of the frames.
frameHeight	40	Integer	Height of the frames.
showFilmstrip	true	Boolean	Defines visibility of filmstrip.
showCaption	false	Boolean	Defines visibility of captions.
transitionInterval	4000	Integer	Defines interval of slideshow.
autoPlay	true	Boolean	Images are displayed in a slideshow in autoPlay.

Media

Este componente es el reproductor genérico de contenido multimedia embotrable en una página JSF. Soporta varios formatos como Flash, quicktime, Windows media, realplayer y pdf.



Atributos

Name	Default	Type	Description
id	null	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	Media source to play.
player	null	String	Type of the player, possible values are "quicktime", "windows", "flash", "real" and "pdf".
width	null	String	Width of the player.
height	null	String	Height of the player.
style	null	String	Style of the player.
styleClass	null	String	StyleClass of the player.
cache	true	Boolean	Controls browser caching mode of the resource.

Ejemplo de código

(FlashPlayer)

```
<p:media value="http://www.youtube.com/v/KZnUr8lcqjo" width="420"
height="315" player="flash"/>
```

ImageSwitch

Es otro estilo de galería de imágenes que permite múltiples efectos para el cambio entre imagen.

Atributos

Name	Default	Type	Description
id	null	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	Name of the client side widget.
effect	null	String	Name of the effect for transition.
speed	500	Integer	Speed of the effect in milliseconds.
slideshowSpeed	3000	Integer	Slideshow speed in milliseconds.
slideshowAuto	true	Boolean	Starts slideshow automatically on page load.
style	null	String	Style of the main container.
styleClass	null	String	Style class of the main container.

Ejemplo de código

```
<p:imageSwitch effect="fade" id="fadeEffect">

    <ui:repeat value="#{imageView.images}" var="image"
id="fadeEffectImages">

        <p:graphicImage name="/demo/images/nature/#{image}" id="image" />

    </ui:repeat>

</p:imageSwitch>
```

ImageCompare

El comparador de imágenes proporciona una interfaz gráfica que permite mostrar similares imágenes para poder ver más claramente similitudes o diferencias.



Atributos

Name	Default	Type	Description
id	null	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	Name of the client side widget.

leftImage	null	String	Source of the image placed on the left side
rightImage	null	String	Source of the image placed on the right side
width	null	String	Width of the images
height	null	String	Height of the images
style	null	String	Inline style of the container element
styleClass	null	String	Style class of the container element

Ejemplo de código

```
<p:imageCompare leftImage="/resources/demo/images/compare/lara-ps3.png"
                rightImage="/resources/demo/images/compare/lara-
ps4.png"
                width="450" height="435"/>
```

Cropper

Este es un component que nos permite extraer una parte concreta de una imagen para crear una nueva.



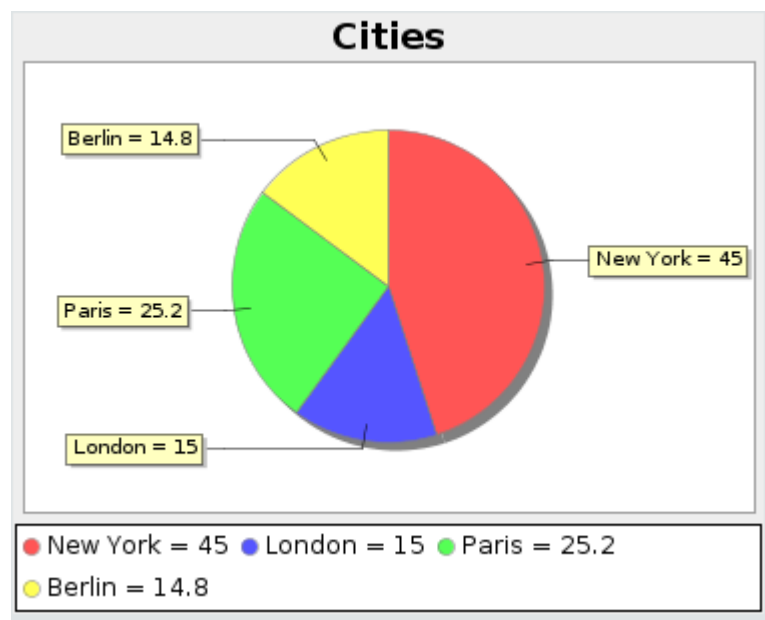
Atributos

Name	Default	Type	Description
id	null	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 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	When set true, process validations logic is executed at apply request values phase for this component.
required	false	Boolean	Marks component as required

validator	null	Method Expr	A method binding expression that refers to a method validationg the input
valueChangeListener	null	Method Expr	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	Name of the client side widget.
image	null	String	Context relative path to the image.
alt	null	String	Alternate text of the image.
aspectRatio	null	Double	Aspect ratio of the cropper area.
minSize	null	String	Minimum size of the cropper area.
maxSize	null	String	Maximum size of the cropper area.
backgroundColor	null	String	Background color of the container.
backgroundOpacity	0,6	Double	Background opacity of the container
initialCoords	null	String	Initial coordinates of the cropper area.

GraphicImage

Este es un componente que nos permite presentar imágenes que se crean en tiempo de ejecución o imágenes almacenadas en bases de datos.



Atributos

Name	Default	Type	Description
id	null	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
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

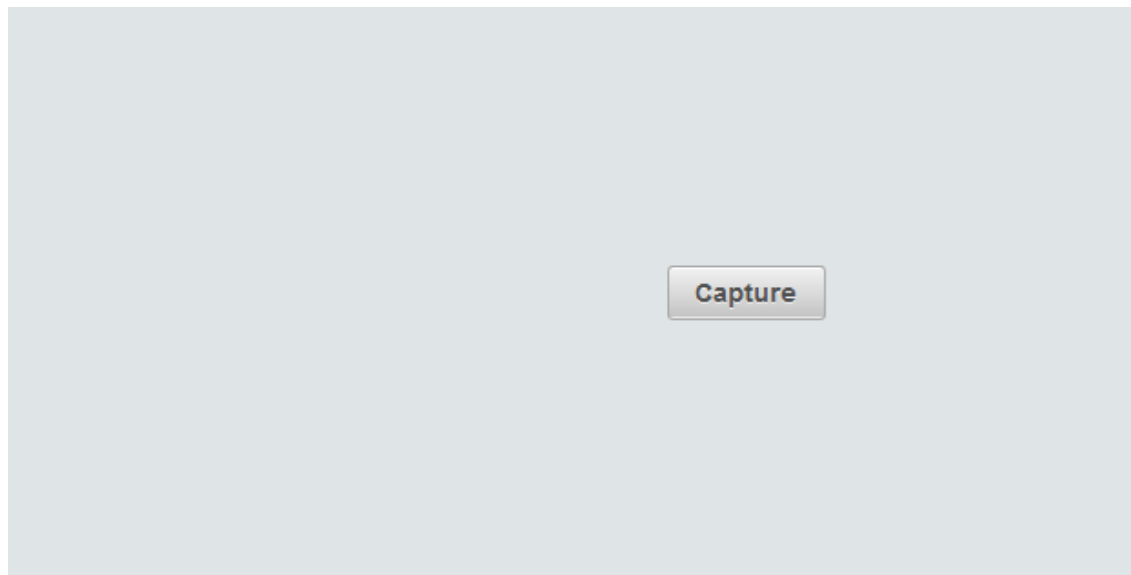
onclick	null	String	onclick dom event handler
ondblclick	null	String	ondblclick dom event handler
onkeydown	null	String	onkeydown dom event handler
onkeypress	null	String	onkeypress dom event handler
onkeyup	null	String	onkeyup dom event handler
onmousedown	null	String	onmousedown dom event handler
onmousemove	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
cache	true	String	Enables/Disables browser from caching the image
name	null	String	Name of the image.
library	null	String	Library name of the image.

Ejemplo de código

```
<p:graphicImage value="#{graphicImageView.chart}" />
```

PhotoCam

Es un componente de entrada que permite tomar fotos desde la webcam y enviarlos.



Atributos

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	false	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	0	Boolean	When set true, process validations logic is executed at apply request values phase for this component.
required	0	boolean	Marks component as required
validator	null	Method Expr	A method binding expression that refers to a method validationg the input
valueChangeListener	null	Method Expr	A method binding expression that refers to a method for handling a valuechangeevent
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	Name of the client side widget.
style	null	String	Inline style of the component.
styleClass	null	String	Style class of the component.
process	null	String	Identifiers of components to process during capture.
update	null	String	Identifiers of components to update during capture.
listener	null	Method Expr	Method expression to listen to capture events.

Ejemplo de Código

```
<h:form>

    <h:panelGrid columns="3" cellpadding="5">

        <p:photoCam widgetVar="pc" listener="#{photoCamView.oncapture}"
update="photo" />

        <p:commandButton type="button" value="Capture"
onclick="PF('pc').capture()" />

        <p:outputPanel id="photo">

            <p:graphicImage
name="demo/images/photocam/#{photoCamView.filename}.jpeg" rendered="#{not
empty photoCamView.filename}" />

        </p:outputPanel>

    </h:panelGrid>
```

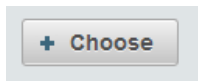
</h:form>

File

Estos componentes permiten la interacción mediante datos entre el usuario y la aplicación permitiendo tanto la subida como la descarga de datos.

FileUpload

Este es un simple componente que utiliza el método nativo del navegador para subir los archivos.



Name	Default		Description
id	null	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 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	When set true, process validations logic is executed at apply request values phase for this component.
required	false	Boolean	Marks component as required.
validator	null	MethodExpr	A method expression that refers to a method validationg the input.
valueChangeListener	null	MethodExpr	A method 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 fails.
widgetVar	null	String	Name of the client side widget.
update	null	String	Component(s) to update after fileupload completes.
process	null	String	Component(s) to process in fileupload request.
fileUploadListener	null	MethodExpr	Method to invoke when a file is uploaded.
multiple	false	Boolean	Allows choosing of multi file uploads from native file browse dialog
auto	false	Boolean	When set to true, selecting a file starts the upload process implicitly.
label	Choose	String	Label of the browse button.
allowTypes	null	String	Regular expression for accepted file types, e.g. /(\\.*)(gif jpe?g png)\$/
sizeLimit	null	Integer	Individual file size limit in bytes.
fileLimit	null	Integer	Maximum number of files allowed to upload.
style	null	String	Inline style of the component.
styleClass	null	String	Style class of the component.
mode	advanced	String	Mode of the fileupload, can be <i>simple</i> or <i>advanced</i> .
uploadLabel	Upload	String	Label of the upload button.
cancelLabel	Cancel	String	Label of the cancel button.

invalidSizeMessage	null	String	Message to display when size limit exceeds.
invalidFileMessage	null	String	Message to display when file is not accepted.
fileLimitMessage	null	String	Message to display when file limit exceeds.
dragDropSupport	true	Boolean	Specifies dragdrop based file selection from filesystem, default is true and works only on supported browsers.
onstart	null	String	Client side callback to execute when upload begins.
onerror	null	String	Callback to execute if fileupload request fails.
oncomplete	null	String	Client side callback to execute when upload ends.
disabled	false	Boolean	Disables component when set true.
messageTemplate	{name} {size}	String	Message template to use when displaying file validation errors.
previewWidth	80	Integer	Width for image previews in pixels.

Ejemplo de Código

```
<h:form enctype="multipart/form-data">

    <p:growl id="messages" showDetail="true" />

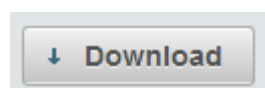
    <p:fileUpload value="#{fileUploadView.file}" mode="simple"
skinSimple="true"/>

    <p:commandButton value="Submit" ajax="false"
actionListener="#{fileUploadView.upload}" disabled="true" />

</h:form>
```

Download

Este componente se utiliza para que los contenidos subidos puedan ser descargados por el usuario.



Atributos

Name	Default	Type	Description
value	null	StreamedContent	A streamed content instance
contextDisposition	attachment	String	Specifies display mode.

Ejemplo de código

```
<p:dialog modal="true" widgetVar="statusDialog" header="Status"
draggable="false" closable="false" resizable="false">

    <p:graphicImage name="/demo/images/ajaxloadingbar.gif" />
```

```
</p:dialog>

<h:form>

    <p:commandButton value="Download" ajax="false"
onclick="PrimeFaces.monitorDownload(start, stop);" icon="ui-icon-arrowthick-
1-s">

        <p:fileDownload value="#{fileDownloadView.file}" />

    </p:commandButton>

</h:form>

<script type="text/javascript">

function start() {

    PF('statusDialog').show();

}

function stop() {

    PF('statusDialog').hide();

}

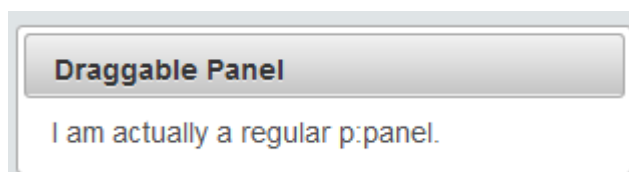
</script>
```

DragDrop

Estos elementos se basan en la característica de ser arrastrables por el usuario, permitiendo una mayor interacción entre la aplicación y el usuario. También permiten la creación de unos elementos propios.

Draggable

Estos son componentes con la capacidad de ser arrastrados por el usuario.



Atributos

Name	Default	Type	Description
id	null	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	Name of the client side widget
proxy	false	Boolean	Displays a proxy element instead of actual element.
dragOnly	false	Boolean	Specifies a draggable that can't be dropped.
for	null	String	Id of the component to add draggable behavior
disabled	false	Boolean	Disables draggable behavior when true.
axis	null	String	Specifies drag axis, valid values are 'x' and 'y'.
containment	null	String	Constraints dragging within the boundaries of containment element
helper	null	String	Helper element to display when dragging
revert	false	Boolean	Reverts draggable to it's original position when not dropped onto a valid droppable
snap	false	Boolean	Draggable will snap to edge of near elements
snapMode	null	String	Specifies the snap mode. Valid values are 'both', 'inner' and 'outer'.
snapTolerance	20	Integer	Distance from the snap element in pixels to trigger snap.
zindex	null	Integer	ZIndex to apply during dragging.
handle	null	String	Specifies a handle for dragging.
opacity	1	Double	Defines the opacity of the helper during dragging.
stack	null	String	In stack mode, draggable overlap is controlled automatically using the provided selector, dragged item always overlays other draggables.
grid	null	String	Dragging happens in every x and y pixels.
scope	null	String	Scope key to match draggables and droppables.
cursor	crosshair	String	CSS cursor to display in dragging.
dashboard	null	String	Id of the dashboard to connect with.
appendTo	null	String	A search expression to define which element to append the draggable helper to.

Ejemplo de código

```
<p:panel id="pn1" header="Draggable Panel">

    <h:outputText value="I am actually a regular p:panel." />

</p:panel>

<p:draggable for="pn1" />
```


DataTable

Los elementos arrastrables amplían la utilidad de las tablas, por ejemplo cambiando de sitio ciertos elementos de una table.

Available Cars

	Id	Year	Brand	Color
+	e86457a6	2003	Volvo	Orange
+	43f277a6	2008	Honda	Brown
+	b79b158e	1998	Volvo	Red
+	c957ef47	1986	Mercedes	Brown
+	035d1163	2006	Ford	Red
+	e814e499	1981	Ford	Silver
+	a0bdb1d9	1990	Honda	Green
+	2a0ac221	1970	Ford	Yellow

Selected Cars

Id	Year	Brand	Color	
a10cd21a	1966	BMW	Brown	

Atributos

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	false	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	Data to display.
var	null	String	Name of the request-scoped variable used to refer each data.
rows	null	Integer	Number of rows to display per page.
first	0	Integer	Index of the first row to be displayed
widgetVar	null	String	Name of the client side widget.
paginator	false	Boolean	Enables pagination.
paginatorTemplate	null	String	Template of the paginator.
rowsPerPageTemplate	null	String	Template of the rowsPerPage dropdown.
rowsPerPageLabel	null	String	Label for the rowsPerPage dropdown.
currentPageReportTemplate	null	String	Template of the currentPageReport UI.
pageLinks	10	Integer	Maximum number of page links to display.
paginatorPosition	both	String	Position of the paginator.
paginatorAlwaysVisible	true	Boolean	Defines if paginator should be hidden if total data count is less than number of rows per page.
scrollable	false	Boolean	Makes data scrollable with fixed header.
scrollHeight	null	Integer	Scroll viewport height.
scrollWidth	null	Integer	Scroll viewport width.
selectionMode	null	String	Enables row selection, valid values are "single" and "multiple".
selection	null	Object	Reference to the selection data.
rowIndexVar	null	String	Name of iterator to refer each row index.
emptyMessage	No records found.	String	Text to display when there is no data to display. Alternative is emptyMessage facet.
style	null	String	Inline style of the component.
styleClass	null	String	Style class of the component.
dblClickSelect	false	Boolean	Enables row selection on double click.
liveScroll	false	Boolean	Enables live scrolling.
rowStyleClass	null	String	Style class for each row.
onExpandStart	null	String	Client side callback to execute before expansion.
resizableColumns	false	Boolean	Enables column resizing.
sortBy	null	Object	Property to be used for default sorting.
sortOrder	ascending	String	"ascending" or "descending".
scrollRows	0	Integer	Number of rows to load on live scroll.
rowKey	null	String	Unique identifier of a row.
tableStyle	null	String	Inline style of the table element.

Name	Default	Type	Description
tableStyleClass	null	String	Style class of the table element.
filterEvent	keyup	String	Event to invoke filtering for input filters.
filterDelay	300	Integer	Delay in milliseconds before sending an ajax filter query.
draggableColumns	false	Boolean	Columns can be reordered with dragdrop when enabled.
editable	false	Boolean	Controls incell editing.
lazy	false	Boolean	Controls lazy loading.
filteredValue	null	List	List to keep filtered data.
sortMode	single	String	Defines sorting mode, valid values are <i>single</i> and <i>multiple</i> .
editMode	row	String	Defines edit mode, valid values are <i>row</i> and <i>cell</i> .
editingRow	false	Boolean	Defines if cell editors of row should be displayed as editable or not.
cellSeparator	null	String	Separator text to use in output mode of editable cells with multiple components.
summary	null	String	Summary attribute for WCAG.
frozenRows	null	Object	Collection to display as fixed in scrollable mode.
dir	ltr	String	Defines text direction, valid values are <i>ltr</i> and <i>rtl</i> .
liveResize	false	Boolean	Columns are resized live in this mode without using a resize helper.
stickyHeader	false	Boolean	Sticky header stays in window viewport during scrolling.
expandedRow	false	Boolean	Defines if row should be rendered as expanded by default.
disabledSelection	false	Boolean	Disables row selection when true.
rowSelectMode	new	String	Defines row selection mode for multiple selection. Valid values are "new", "add" and "checkbox".
rowExpandMode	new	String	Defines row expand mode, valid values are "single" and "multiple" (default).
dataLocale	null	Object	Locale to be used in features such as filtering and sorting, defaults to view locale.
nativeElements	false	Boolean	Uses native radio-checkbox elements for row selection.
frozenColumns	null	Integer	Number of columns to freeze from start index 0.

draggableRows	false	Boolean	When enabled, rows can be reordered using dragdrop.
caseSensitiveSort	false	Boolean	Case sensitivity for sorting, insensitive by default.
skipChildren	false	Boolean	Ignores processing of children during lifecycle, improves performance if table only has output components.
disabledTextSelection	true	Boolean	Disables text selection on row click.
sortField	null	String	Name of the field to pass lazy load method for sorting. If not specified, sortBy expression is used to extract the name.

Ejemplo de código

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

    <p:column style="width:20px">

        <h:outputText id="dragIcon" styleClass="ui-icon ui-icon-
arrow-4" />

        <p:draggable for="dragIcon" revert="true" helper="clone"/>
    </p:column>

    <p:column headerText="Id">

        <h:outputText value="#{car.id}" />
    </p:column>

    <p:column headerText="Year">

        <h:outputText value="#{car.year}" />
    </p:column>

    <p:column headerText="Brand">

        <h:outputText value="#{car.brand}" />
    </p:column>

    <p:column headerText="Color">

        <h:outputText value="#{car.color}" />
    </p:column>
</p:dataTable>
```

```
</p:column>
```


```
</p:dataTable>
```

DataGrid


Este mismo concepto puede aplicarse a otros componentes y de ello surgen los **DataGrid**, que son rejillas donde pueden arrastrarse otro tipo de objetos para, por ejemplo, ampliar su información.

AvailableCars


1b5926a6




75204795




353c332b




c43701fd




0b48a382




38e89a48




24407589



6e13d4d8



Selected Cars

Id	Year	Brand	Color	
392579d3	1972	Jaguar	Red	

Ejemplo de código

En este caso la sintaxis es igual que la anterior pero incluyendo el nuevo elemento

```
<p:dataGrid id="availableCars" var="car" value="#{dndCarsView.cars}"
columns="3">

    <p:panel id="pnl" header="#{car.id}" style="text-align:center">

        <h:panelGrid columns="1" style="width:100%">

            <p:graphicImage name="/demo/images/car/#{car.brand}.gif"
/>

        </h:panelGrid>

    </p:panel>

    <p:draggable for="pnl" revert="true" handle=".ui-panel-titlebar"
stack=".ui-panel"/>

</p:dataGrid>
```

Client side validation

PrimeFaces permite validación del lado cliente, mediante patrones establecidos en el propio xhtml como definidos en las clases java anexos. Estos elementos actúan sobre otros elementos, en la mayoría de los casos input o output de texto simples y pueden ser de varios tipos:

Basico

El patrón se define dentro del xhtml en etiquetas de validación, o de conversión.

Custom

El patrón se define dentro del xhtml pero separado de la estructura y anidado en una etiqueta javascript.

Bean

Los patrones se definen sobre los atributos java dentro del Bean asociado, gracias a anotaciones con las que acotamos el rango o valores asociados.

Event

Son comprobados dinámicamente sobre el input, y se definen en el xhtml con atributos event dentro de etiquetas de input o output.

Ejemplo de código

Este ejemplo se corresponde con una validación de tipo básico, en este caso formada por una comprobación por patrón y una conversión.

```
<h:form>
    <p:panel header="Validate">
        <p:messages autoUpdate="true"/>
    <h:outputLabel for="money" value="Currency ($):" style="font-
weight:bold"/>
        <p:inputText id="money" value="#{validationView.money}"
label="Currency">
            <f:convertNumber type="currency" currencySymbol="$"/>
        </p:inputText>
        <p:message for="money" />
        <h:outputText value="#{validationView.money}">
            <f:convertNumber type="currency" currencySymbol="$" />
        </h:outputText>

        <h:outputLabel for="regex" value="Regex (^[a-zA-Z]+$):"
style="font-weight:bold"/>
        <p:inputText id="regex" value="#{validationView.regexText}"
validatorMessage="Value does not match pattern.">
            <f:validateRegex pattern="^[a-zA-Z]+$" />
        </p:inputText>
        <p:message for="regex" />
        <h:outputText value="#{validationView.regexText}" />
    </p:panel>
</h:form>
```

Visualización

Currency (\$):	<input type="text" value="100"/>	<div>✖ Currency: '100' could not be understood as a currency value. Example: \$100</div>
Regex (^[a-zA-Z]+\$):	<input type="text"/>	<div>✖ Value does not match pattern.</div>

Dialog framework

Permite generar cuadros de dialogo generados dinámicamente en tiempo de ejecución. Pueden ser tanto tablas de datos a visualizar, como mensajes emergentes, incluso elementos con los que se permita interactuar.

Ejemplo de código

Menú de opciones:

```
<h:form>
```

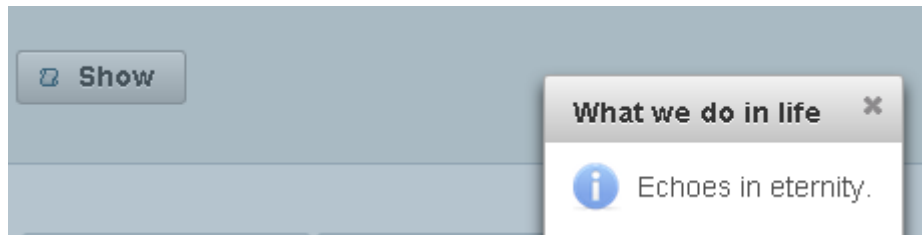
```
<p:commandButton value="Show" icon="ui-icon-script"
actionListener="#{dfView.showMessage}" />
</h:form>
```

Dialog message (Definido en el bean):

```
public void showMessage() {
    FacesMessage message = new FacesMessage(FacesMessage.SEVERITY_INFO,
    "What we do in life", "Echoes in eternity.");

    RequestContext.getCurrentInstance().showMessageInDialog(message);
}
```

Visualización



Misc

Este apartado se comentan elementos añadidos de primeFaces que no se pueden encuadrar en ninguna de las otras agrupaciones.

FontAwesome

Se trata de una librería con más de 479 iconos vectoriales escalables que podemos utilizar para dar una visualización más interesante a nuestro proyecto.

Ejemplo de código

```
< h3 > Menú </ h3 >

< p: Menú >

    < p: submenú etiqueta = "Documento" >

        < p: menuitem valor = "Nuevo" url = "#" icon = "fa fa-plus" />

        < p: menuitem valor = "Refresh" url = "#" icon = "fa fa-refresh" />

    </ p: submenú >
```

```
< p: submenú etiqueta = "Navegaciones" >

    < p: menuitem valor = "Inicio" url = " http://www.primefaces.org "
    icon = "fa fa-casa" />

    < p: menuitem valor = "Usuario" url = "#" icon = "fa fa-usuario" />

</ p: submenú >

</ p: Menú >
```

Visualización



IdleMonitor

Consiste en un detector de inactividad que nos da mensajes según nuestra interacción con la página.

Ejemplo de código

Los mensajes concretos se definen en el Bean asociado.

```
<h:form>
    <p:growl id="messages" showDetail="true" sticky="true" />

    <p:idleMonitor timeout="5000">
        <p:ajax event="idle" listener="#{idleMonitorView.onIdle}"
        update="messages" />
        <p:ajax event="active" listener="#{idleMonitorView.onActive}"
        update="messages" />
    </p:idleMonitor>
</h:form>
```

Visualización



ThemeSwitcher

Es un elemento de lista desplegable que nos permite cambiar el tema de la interfaz de forma dinámica y fácil de implementar. Los temas se definen como una clase java y se importan directamente de primeFaces.

Ejemplo de código

```
<p:themeSwitcher id="basic" style="width:165px">
    <f:selectItem itemLabel="Choose Theme" itemValue="" />
    <f:selectItems value="[afterdark, afternoon, afterwork, aristo,
black-tie, blitzer, bluesky, bootstrap, casablanca, cupertino, cruze, dark-
hive, delta, dot-luv, eggplant, excite-bike, flick, glass-x, home, hot-
sneaks, humanity, le-frog, midnight, mint-choc, overcast, pepper-grinder,
redmond, rocket, sam, smoothness, south-street, start, sunny, swanky-purse,
trontastic, ui-darkness, ui-lightness, vader]" var="theme" itemLabel=""
itemValue=""/>
</p:themeSwitcher>

<p:outputLabel for="advanced" value="Advanced:" />
<p:themeSwitcher id="advanced" style="width:165px" effect="fade"
var="t">
    <f:selectItem itemLabel="Choose Theme" itemValue="" />
    <f:selectItems value="[afterdark, afternoon, afterwork, aristo,
black-tie, blitzer, bluesky, bootstrap, casablanca, cupertino, cruze, dark-
hive, delta, dot-luv, eggplant, excite-bike, flick, glass-x, home, hot-
sneaks, humanity, le-frog, midnight, mint-choc, overcast, pepper-grinder,
redmond, rocket, sam, smoothness, south-street, start, sunny, swanky-purse,
trontastic, ui-darkness, ui-lightness, vader]" var="theme" itemLabel=""
itemValue=""/>

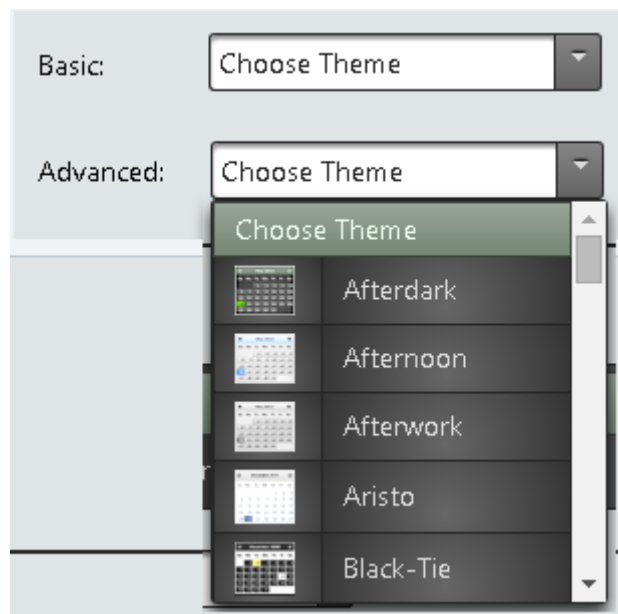
    <p:column>
        <h:outputText styleClass="ui-theme ui-theme-" style="display-
block" />
    </p:column>

    <p:column>

    </p:column>
</p:themeSwitcher>
```

Visualización

Este sería el selector.



Y así como actuaría sobre algunos elementos con diferentes temas.

