Exploring Extensions Component Tags and Their Attributes

The following table serves as an easy reference for you. The description of the components and their attributes have been taken from `http://www.primefaces.org/showcase-ext/views/home.jsf`.

Component name	Attribute	Description
AjaxErrorHandler	`id`	This is a unique identifier for the component in a naming container.
	`widgetVar`	This is the name of the client-side widget. It allows us to define the widget name on the frontend with the `hide()` and `isVisible()` methods. If XHTML contains additional AjaxErrorHandler with widgetWar, every widgetWar is the same JavaScript object.
	`type`	This indicates an exception in the class name. If the type is not defined, AjaxErrorHandler is defined for all the exceptions. If the type is defined, AjaxErrorHandler is defined for this type of exception.
	`title`	This denotes the title text in the pop up error box. The default value is `{error-name}`.

Component name	Attribute	Description
	`body`	This denotes the body text/ HTML in the pop up error box. The default value is {`error-message`}.
	`button`	This indicates the button text in the pop up error box. The default value is `Reload`.
	`buttonOnclick`	This is the JavaScript function for a button in the pop up error box. The default value is the JavaScript function for the RELOAD content.
	`onerror`	This is the JavaScript function (for an error or response). It is called before the construct error box is called on the client side. If this function returns `false`, the steps are broken down and the information along with its description is constructed.
BlockUI	`id`	This is a unique identifier for the component in a naming container.
	`rendered`	This is a Boolean value that specifies how the component is rendered. When set to `false`, the component will not be rendered.
	`binding`	This is an EL expression that refers to a server-side UIComponent instance in a backing bean.
	`widgetVar`	This is the name of the client-side widget.
	`css`	This indicates the styles for the message when blocking it; they should be defined as a JSON object, for example {`backgroundColor: 'fff', width: 30%`}.
	`cssOverlay`	This indicates the styles for the overlay; they should be defined as a JSON object, for example {`opacity: 0.4, backgroundColor: 'red'`}.

Component name	Attribute	Description
	`source`	This specifies the component that sends an Ajax request and triggers blocking. If nothing is specified, the parent component is used.
	`target`	This specifies the component(s) or HTML element(s) to be blocked. If the target is null or not defined, the entire page is blocked.
	`content`	This indicates the ID of the component in terms of findComponent, which is to be used as the content shown during blocking. If nothing is specified, the content of the blockUI component is used.
	`event`	This indicates the name(s) of one or several accepted events. Several events should be defined as a space- or comma-separated list.
	`autoShow`	This is a flag that denotes whether the blocking is shown automatically without the need of a JavaScript widget. The default value is `false`.
	`timeout`	This indicates the time in milliseconds to wait before auto-unblocking. Do not define it or set it to `0` to disable auto-unblocking. The default value is `0`.
	`centerX`	Set this attribute to `true` to have the message automatically centered along the *x* axis. Note that it only affects element blocking (page blocking is controlled by CSS via the `css` attribute). The default value is `true`.
	`centerY`	Set this attribute to `true` to have the message automatically centered along the *y* axis. The default value is `true`.

Component name	Attribute	Description
CKEditor	`id`	This is a unique identifier for the component in a naming container.
	`rendered`	This Boolean value specifies how the component is rendered. When set to `false`, the component will not be rendered.
	`binding`	This is an EL expression that refers to a server-side UIComponent instance in a backing bean.
	`widgetVar`	This is the name of the client-side widget.
	`value`	This indicates the value of the component.
	`height`	This indicates the height of the editor. The default value is `200px`.
	`width`	This indicates the width of the editor. The default value is `600px`.
	`theme`	This indicates the theme of the editor.
	`skin`	This indicates the skin of the editor. The valid skins are `kama` and `moono`.
	`toolbar`	This indicates the toolbar's layout.
	`readOnly`	This defines whether the editor is read-only.
	`interfaceColor`	This indicates the color of the interface, for example `#33fc14`.
	`language`	This is the default language to be used in case no language is set using the language option and the editor is not able to use the user's language.
	`defaultLanguage`	This setting is used to set the CKEditor language. In case this option is not set, the editor will automatically try to load it with the user's language, if supported; otherwise, the default language will be used.

Component name	Attribute	Description
	`contentsCss`	This setting specifies the CSS file that is to be used to apply the style to the contents.
	`customConfig`	This indicates the path to a custom CKEditor JavaScript config file.
	`tabindex`	This is the tab index that is to be assigned to the editor. If not specified, no tab index will be assigned.
	`label`	The `label` attribute denotes a display name for this component.
	`converter`	This is 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. If it's a static text, it must refer to a converter ID.
	`required`	This marks the component as required.
	`escape`	This defines whether the content of the component should be escaped.
	`requiredMessage`	This message is to be displayed when the required field validation fails.
	`converterMessage`	This message is to be displayed when the conversion fails.
	`validatorMessage`	This message is to be displayed when the validation fails.
	`validator`	This method expression refers to a method that validates the input.
(Events)	`save`	This event is fired when the save button is pressed.
	`Initialize`	This event is fired after the editor is successfully initialized.
	`focus`	This event is fired when the editor is focused.
	`blur`	This event is fired when the editor loses focus.

Component name	Attribute	Description
	`wysiwygMode`	This event is fired after switching to the WYSIWYG mode.
	`sourceMode`	This event is fired after switching to the source mode.
	`Dirty`	This event is fired after the content has been changed without the editor losing focus.
	`change`	This event is fired after a blur is encountered and if the content has been changed.
CodeMirror	`Id`	This is a unique identifier for the component in a naming container.
	`rendered`	This Boolean value specifies how the component is rendered. When set to `false`, the component will not be rendered.
	`binding`	This is an EL expression that refers to a server-side UIComponent instance in a backing bean.
	`widgetVar`	This is the name of the client-side widget.
	`value`	This indicates the value of the component.
	`completeMethod`	This method provides suggestions.
	`theme`	This denotes the theme to style the editor with.
	`mode`	This denotes the mode to be used. When not specified, it will default to the first mode that was loaded. It may be a string that either simply names the mode or is a MIME type associated with the mode. Alternatively, it may be an object containing configuration options for the mode, with a `name` property that names the mode (for example, `{name: "javascript", json: true}`).

Component name	Attribute	Description
	`indentUnit`	This indicates how many spaces a block (whatever that means in the edited language) should be indented by. The default value is `2`.
	`smartIndent`	This indicates whether the context-sensitive indentation that the mode provides is to be used (or whether it is to be indented in the same way as the line before). It defaults to `true`.
	`tabSize`	This indicates the width of a tab character. It defaults to `4`.
	`electricChars`	This decides whether the editor should reindent the current line when a character that is typed might change its proper indentation (it only works if the mode supports indentation). The default value is `true`.
	`keyMap`	This configures the keymap to be used. The default value is `default`, which is the only keymap defined in `codemirror.js`. Extra keymaps are found in the `keymap` directory.
	`lineWrapping`	This indicates whether CodeMirror should scroll or wrap for long lines. It defaults to `false` (scroll).
	`lineNumbers`	This decides whether to show the line numbers to the left of the editor.
	`firstLineNumber`	This denotes the number to start counting lines from. The default value is `1`.
	`gutter`	This can be used to force a "gutter" (empty space on the left of the editor) to be shown even when no line numbers are active. It is useful for setting markers.
	`fixedGutter`	When enabled (`off` by default), it will make the gutter visible when the document is scrolled horizontally.

Component name	Attribute	Description
	`readOnly`	This disables editing of the editor content by the user. If the special value, `nocursor`, is specified (instead of simply `true`), focusing the editor is also disallowed.
	`matchBrackets`	This determines whether the brackets are matched whenever the cursor is moved next to a bracket.
	`workTime`	The highlight is done by a pseudo background thread that will work for workTime milliseconds and then use timeout to sleep for workDelay milliseconds. The defaults are `200` and `300`; you can change these options to make the highlights more or less aggressive.
	`workDelay`	The highlight is done by a pseudo background thread that will work for workTime milliseconds and then use timeout to sleep for workDelay milliseconds. The defaults are `200` and `300`; you can change these options to make the highlights more or less aggressive.
	`pollInterval`	This indicates how quickly CodeMirror should poll its input textarea for changes. Most inputs are captured by events, but some inputs, such as the IME input on some browsers, don't generate events that allow CodeMirror to properly detect it. Thus, it polls. The default value is `100` milliseconds.
	`undoDepth`	This denotes the maximum number of undo levels that the editor stores. It defaults to `40`.
	`tabindex`	This denotes the tab index to be assigned to the editor. If it is not specified, no tab index will be assigned.
	`Label`	The `label` attribute displays the name for this component.

Component name	Attribute	Description
	`extraKeys`	This can be used to specify extra key bindings for the editor. When specified, there should be objects with property names such as `Ctrl-A`, `Home`, and `Ctrl-Alt-Left`.
	`Converter`	This is 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. If it's a static text, it must refer to a converter ID.
	`process`	This component(s) is used to process a view partially instead of the whole view.
	`onstart`	This is the JavaScript handler to be executed before an Ajax request begins.
	`oncomplete`	This is the JavaScript handler to be executed when an Ajax request is completed.
	`onerror`	This is the JavaScript handler to be executed when an Ajax request fails.
	`onsuccess`	This is the JavaScript handler to be executed when an Ajax request succeeds.
	`global`	Global Ajax requests are listened to by the ajaxStatus component; setting `global` to `false` will not trigger ajaxStatus. The default value is `true`.
	`async`	When this is set to `true`, Ajax requests are not queued. The default value is `false`.
	`escape`	This defines whether the content of the component should be escaped from.
	`escapeSuggestions`	This defines whether the suggestions should be escaped from.
	`Required`	This marks the component as required.

Component name	Attribute	Description
(Events)	`Change`	This event is fired every time the content of the editor is changed.
	`highlightComplete`	This event is fired whenever the editor's content has been fully highlighted.
	`Focus`	This event is fired when the editor is focused on.
	`Blur`	This event is fired when the editor loses focus.
DynaForm	`Id`	This is a unique identifier for the component in a naming container.
	`Rendered`	This Boolean value specifies how to render the component. When set to `false`, the component will not be rendered.
	`Binding`	This is an EL expression that refers to a server-side UIComponent instance in a backing bean.
	`widgetVar`	This denotes the name of the client-side widget.
	`Value`	This is an instance of type DynaFormModel, which represents the model of this component.
	`Var`	This indicates the name of the request-scoped variable of the underlying object for each iteration.
	`varContainerId`	This indicates the name of the request-scoped variable that contains the prefix of the client ID within pe:dynaFormControl. This property allows us to get the entire client ID of a component within pe:dynaFormControl. The entire client ID is sometimes required for JavaScript or `RequestContext.update(...)`.

Component name	Attribute	Description
	`autoSubmit`	This flag checks whether the form has to be submitted automatically after the current page has been loaded. Note that the form can be submitted if it contains at least one Submit button. The default value is `false`.
	`openExtended`	This flag checks whether the extended grid has to be shown during page load. The default value is `false`.
	`buttonBarPosition`	This indicates the button bar's position. The possible values are `top`, `bottom`, and `both`. The default value is `bottom`.
	`Style`	This denotes the style of the main container element. The default value is `NULL`.
	`styleClass`	This denotes the style class of the main container element. The default value is `NULL`.
dynaFormControl	`Id`	This is a unique identifier for the component in a naming container.
	`Rendered`	This Boolean value specifies how the component is rendered. When set to `false`, the component will not be rendered.
	`binding`	This is an EL expression that refers to a server-side UIComponent instance in a backing bean.
	`type`	This is the type that corresponds to the form control. The default value is `default`.
	`for`	This provides a reference between `label` and the corresponding controls such as the `for` attribute in h:outputLabel or p:outputLabel. The default value is `NULL`.
	`styleClass`	This indicates the style class of the container element for this control. The default value is `NULL`.

Component name	Attribute	Description
FluidGrid	`id`	This is a unique identifier for the component in a naming container.
	`Rendered`	This Boolean value specifies how the component is rendered. When set to `false`, the component will not be rendered.
	`binding`	This is an EL expression that refers to a server-side UIComponent instance in a backing bean.
	`widgetVar`	This indicates the name of the client-side widget.
	`value`	This is a collection of items (FluidGridItem) that represent the model of this component.
	`var`	This is the name of the request-scoped variable of the underlying object for each iteration.
	`varContainerId`	This indicates the name of the request-scoped variable that contains the prefix of the client ID within pe:fluidGridItem. This property gets the whole client ID of a component within pe:dynaFormControl. The whole client ID is sometimes required for JavaScript or `RequestContext.update(...)`.
	`style`	This indicates the style of the main container element. The default value is `NULL`.
	`styleClass`	This indicates the style class of the main container element. The default value is `NULL`.
	`hGutter`	This indicates the horizontal space between item elements (interpreted as pixels). The default value is `0`.
	`vGutter`	This indicates the vertical space between item elements (interpreted as pixels). The default value is `0`.

Component name	Attribute	Description
	`fitWidth`	This sets the width of the container to fit the available number of columns, based on the size of the container's parent element. When enabled, you can center the container with CSS. Note that this option does not work with the percentage width of items. The default value is `false`.
	`originLeft`	This controls the horizontal flow of the layout. By default, item elements start positioning from the left. It is set to `false` for right-to-left layouts.
	`originTop`	This controls the vertical flow of the layout. By default, item elements start positioning from the top. It is set to `false` for bottom-up layouts (just as with Tetris!)
	`resizeBound`	This binds the layout to resize the window. You can bind and unbind the resize layout afterwards with the bindResize and unbindResize client-side widget methods. The default value is `true`.
	`stamp`	This specifies the elements that are stamped within the layout. These are special layout elements that will not be laid out. Rather, FluidGrid will lay out the item elements below the stamped elements. You can use every search expression supported by PrimeFaces Search Expression Framework here. The default value is `NULL`.
	`transitionDuration`	This indicates the duration of the transition when items change position or appearance, set in a CSS time format. To disable all transitions, set this attribute to `NULL`. The default value is `0.4s`.

Component name	Attribute	Description
	`hasImages`	This is a Boolean flag; it checks whether items contain any images. It can fix overlapping items that are caused by items that change their size after layout due to unloaded media, such as images. Setting this flag allows us to lay out all the items after they have their proper sizes (images have been loaded completely). The default value is `false`.
(events) fluidGridItem	`layoutComplete`	This event is fired after layout and all the positioning transitions have been completed.
	`id`	This is a unique identifier for the component in a naming container.
	`rendered`	This denotes the Boolean value to specify the rendering of the component; when set to `false`, the component will not be rendered.
	`binding`	This is an EL expression referring to a server-side UIComponent instance in a backing bean.
	`type`	This specifies the type corresponding to FluidGridItem. The default value is `default`.
	`styleClass`	This is the style class of the container element for this item. The default value is `NULL`.
Exporter	`target`	This indicates the server-side ID of the DataTable/DataList whose data is to be exported.
	`type`	This indicates the export type, which can be "pdf" and/or "xlsx".
	`fileName`	This indicates the filename of the generated export file; it defaults to the server-side ID of the DataTable/DataList.
	`tableTitle`	This denotes the table/List header title to use.
	`pageOnly`	This exports the current page instead of the whole dataset.

Component name	Attribute	Description
	`preProcessor`	This is the PreProcessor for the exported document.
	`postProcessor`	This is the PostProcessor for the exported document.
	`encoding`	This indicates the character encoding to be used.
	`selectionOnly`	When enabled, only the selection will be exported.
	`subTable`	When enabled, the subtable will be exported.
	`facetBackground`	This indicates the facet background to be used, for example, hex colors such as `#FFFF00` or `#FF0000`.
	`facetFontSize`	This indicates the facet font size to be used, for example, `10` or `12`.
	`facetFontColor`	This denotes the facet font color to use, for example, hex colors such as `#FFFF00` and `#FF0000`.
	`facetFontStyle`	This indicates the facet font style to be used, for example, normal, bold, and/or italics. The default value is `bold`.
	`fontName`	This indicates the font name/ family to be used, for example, Courier New and/or Verdana.
	`cellFontSize`	This indicates the cell's font size to be used, for example, `10` or `12`.
	`cellFontColor`	This denotes the cell's font color to be used, for example, hex colors `#FFFF00` or `#FF0000`.
	`cellFontStyle`	This indicates the cell's font style to be used, for example, normal, bold, or italics. The default value is `normal`.
	`datasetPadding`	This indicates the spacing between multiple DataTable/ DataList instances.
	`orientation`	This indicates the orientation of the PDF format. The possible values are Portrait or Landscape. The default value is `Portrait`.

Component name	Attribute	Description
	`skipComponents`	This denotes the list of components to be skipped in the PDF and Excel export.
ImageAreaSelect	`Id`	This is a unique identifier of the component in a naming container.
	`rendered`	This is a Boolean value that specifies the rendering of the component. When set to `false`, the component will not be rendered.
	`Binding`	This is an EL expression that refers to a server-side UIComponent instance in a backing bean.
	`widgetVar`	This indicates the name of the client-side widget.
	`For`	This indicates the target image.
	`aspectRatio`	This is a string of the form "width:height" that represents the aspect ratio to be maintained, for example, `4:3`.
	`autoHide`	If set to `true`, the selection area will disappear when the selection ends. The default value is `false`.
	`fadeSpeed`	If set to a number greater than zero, the plugin is shown/hidden with a graceful fade in / fade out animation. The default value is `0`.
	`Handles`	If set to `true`, resize handles are shown on the selection area; if set to the "corners", only the corner handles are shown. The default value is `false`.
	`Hide`	If set to `true`, the selection area is hidden.
	`imageHeight`	This indicates the true height of the image (if scaled with the CSS width and height properties).
	`imageWidth`	This indicates the true width of the image (if scaled with the CSS width and height properties).

Component name	Attribute	Description
	`movable`	This determines whether the selection area should be movable. The default value is `true`.
	`Persistent`	If set to `true`, clicking outside the selection area will not start a new selection (that is, the user will only be able to move/resize the existing selection area). The default value is `false`.
	`resizable`	This determines whether the selection area is resizable. The default value is `true`.
	`Show`	If set to `true`, the selection area is shown.
	`zIndex`	This indicates the z-index value to be assigned to plugin elements; usually, imgAreaSelect figures it out automatically, but there are a few cases when it's necessary to set it explicitly.
	`maxHeight`	This indicates the maximum selection height (in pixels).
	`maxWidth`	This indicates the maximum selection width (in pixels).
	`minHeight`	This indicates the minimum selection height (in pixels).
	`minWidth`	This indicates the minimum selection width (in pixels).
	`parentSelector`	This is a jQuery object or selector string that specifies the parent element that the plugin will be appended to. The default value is `body`.
	`keyboardSupport`	This enables/disables keyboard support. The default value is `false`.
(events)	`selectEnd`	This event is fired after the selection is finished.
	`selectStart`	This event is fired after the selection is started.
	`selectChange`	This event is fired after the selection is changed.

Component name	Attribute	Description
ImageRotateAndResize	`id`	This is a unique identifier for the component in a naming container.
	`rendered`	This indicates a Boolean value to specify how the component is rendered. When set to `false`, the component will not be rendered.
	`Binding`	This is an EL expression that refers to a server-side UIComponent instance in a backing bean.
	`widgetVar`	This indicates the name of the client-side widget.
	`for`	This indicates the target image.
(events)	`rotate`	This event is fired after image rotation.
	`resize`	This event is fired after image resizing.
InputNumber	`Id`	This is a unique identifier for the component in a naming container.
	`rendered`	This indicates a Boolean value to specify how the component is rendered. When set to `false`, the component will not be rendered.
	`readOnly`	This defines whether the input is read-only.
	`binding`	This is an EL expression that refers to a server-side UIComponent instance in a backing bean.
	`value`	This denotes the value of the component.
	`converter`	This is 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. If it's a static text, it must refer to a converter ID.
	`immediate`	When set to `true`, the process validation logic is executed at the Apply Request Values phase for this component. The default value is `false`.

Component name	Attribute	Description
	`required`	This marks the component as required.
	`validator`	This is a method expression that refers to a method validating the input.
	`valueChangeListener`	This is a method-binding expression that refers to a method for handling a value change event.
	`requiredMessage`	This message is displayed when the required field validation fails.
	`converterMessage`	This message is displayed when the conversion fails.
	`validatorMessage`	This message is displayed when the validation fails.
	`widgetVar`	This indicates the name of the client-side widget.
	`Disabled`	This disables the component.
	`Label`	This denotes the user label.
	`Onchange`	This is a callback on the `change` event.
	`Style`	This indicates the inline style of the component.
	`styleClass`	This indicates the container's style class.
	`decimalSeparator`	This indicates the decimal separator char. The default value is `.`.
	`thousandSeparator`	This indicates the thousand separator char. The default value is `,`.
	`Symbol`	This indicates the desired symbol or unit. The default value is `none`.
	`symbolPosition`	This indicates the symbol's suffix. The default value is `prefix`.
	`minValue`	This indicates the minimum values. The default value is `0.00`.
	`maxValue`	This indicates the maximum values. The default value is `999999999.99`.

Component name	Attribute	Description
	`roundMethod`	This controls the rounding method. The default value is `Round-Half-Up Symmetric`.
	`decimalPlaces`	This indicates the number of decimal places. The default values are taken from `minValue` and `maxValue`.
	`emptyValue`	This controls empty input display behavior; the options are `empty`, `zero`, and `sign`. The default value is `empty`.
	`accesskey`	This is the access key that transfers focus to the input element.
	`alt`	This is an alternate textual description of the input element.
	`autocomplete`	This controls the browser's autocomplete behavior.
	`dir`	This is a direction for text that does not inherit directionality.
	`lang`	This is a localized user-presentable name.
	`maxlength`	This indicates the maximum number of characters that may be entered in this field.
	`onblur`	This is a client-side callback that executes when the input element loses focus.
	`onclick`	This is a client-side callback that executes when the input element is clicked on.
	`ondblclick`	This is a client-side callback that executes when the input element is double-clicked on.
	`onfocus`	This is a client-side callback that executes when the input element receives focus.
	`onkeydown`	This is a client-side callback that executes when a key is pressed down over an input element.

Component name	Attribute	Description
	`onkeypress`	This is a client-side callback that executes when a key is pressed down and released over an input element.
	`onkeyup`	This is a client-side callback that executes when a key is released over an input element.
	`onmousedown`	This is a client-side callback that executes when a pointer input element is pressed down over an input element.
	`onmousemove`	This is a client-side callback that executes when a pointer input element is moved within an input element.
	`onmouseout`	This is a client-side callback that executes when a pointer input element is moved away from an input element.
	`onmouseover`	This is a client-side callback that executes when a pointer input element is moved onto an input element.
	`onmouseup`	This is a client-side callback that executes when a pointer input element is released over an input element.
	`Onselect`	This is a client-side callback that executes when the text within the input element is selected by the user.
	`tabindex`	This is advisory tooltip information.
	`Title`	This is advisory tooltip information.
KeyFilter	`id`	This is a unique identifier for the component in a naming container.
	`Rendered`	This Boolean value specifies how the component should be rendered. When set to `false`, the component will not be rendered.

Component name	Attribute	Description
	`binding`	This is an EL expression that refers to a server-side UIComponent instance in a backing bean.
	`widgetVar`	This is the name of the client-side widget.
	`for`	This is the target input.
	`regEx`	This defines the regular expression that is used for filtering the input.
	`mask`	This defines the predefined mask that is to be used (`pint`, `int`, `pnum`, `num`, `hex`, `email`, `alpha`, and `alphanum`).
	`testFunction`	This defines the JavaScript code or function that is used for filtering.
	`preventPaste`	This Boolean value specifies whether the component should also prevent the paste operation. The default value is `true`.
Layout	`id`	This is a unique identifier for the component in a naming container.
	`rendered`	This Boolean value specifies how the component should be rendered. When set to `false`, the component will not be rendered.
	`binding`	This is an EL expression that refers to a server-side UIComponent instance in a backing bean.
	`widgetVar`	This is the name of the client-side widget.
	`fullPage`	This specifies whether the layout should span the entire page. The default value is `true`.

Component name	Attribute	Description
	`options`	Layout options are specified either as an instance of the class `LayoutOptions` or as a JSON string representing the serialized `LayoutOptions`. `LayoutOptions`, when created as a Java model, takes precedence over the layout options as tag attributes. Serializing the layout options to a JSON string can increase the time taken for building the layout during an application startup in an application-scoped bean (`LayoutOptions` always gets serialized to JSON so that it can be used in the underlying widget). Use the method `toJson()` in `LayoutOptions` if you want to serialize all the options to a JSON string.
	`style`	This indicates the style of the main Layout container element. The default value is `NULL`.
	`styleClass`	This indicates the style class of the main Layout container element. The default value is `NULL`.
	`state`	This indicates server-side state management. A hash (JSON string) containing all the dimensions and the close and open states of existing layout panes can be bound to a bean.
	`stateCookie`	This indicates client-side state management. When set to `true`, the current layout state will be stored in cookies on window unload and restored during the layout buildup when the user enters the same page. The default value is `false`.
	`resizerTip`	This tip denotes when the resizer-bar can be dragged to resize a pane. The default value is `Resize`.

Component name	Attribute	Description
	`togglerTipOpen`	This tip is toggled when the pane is open. The default value is `Close`.
	`togglerTipClosed`	This tip is toggled when the pane is closed. The default value is `Open`.
	`maskPanesEarly`	This is a useful flag if you are dealing with iframes or objects such as the applets inside layout panes. This option triggers the mask action each time the mouse moves over a resizer instead of waiting until you "grab it" with your mouse. This is done just in case you are about to grab and drag the resizer. Enabling this avoids a slight delay that sometimes occurs when you quickly grab a resizer and try to drag over an iframe or object. The default value is `false`.
(events)	`open`	This event is fired after a layout pane is opened.
	`close`	This event is fired after a layout pane is closed.
	`resize`	This event is fired after a layout pane is resized.
LayoutPane	`id`	This is a unique identifier for the component in a naming container.
	`rendered`	This Boolean value specifies how the component should be rendered. When set to `false`, the component will not be rendered.
	`binding`	This is an EL expression that refers to a server-side UIComponent instance in a backing bean.
	`position`	This refers to the position of a pane. The possible values are `north`, `south`, `west`, `east`, and `center`. The default value is `center`.

Component name	Attribute	Description
	`styleHeader`	This indicates the style of the pane's header. The default value is `NULL`.
	`styleClassHeader`	This indicates the style class of the pane's header. The default value is `NULL`.
	`styleContent`	This indicates the style of the pane's content. The default value is `NULL`.
	`styleClassContent`	This indicates the style class of the pane's content. The default value is `NULL`.
	`resizable`	This makes a pane resizable. The default value is `true`.
	`closable`	This makes a pane closable. The default value is `true`.
	`size`	This specifies the initial size of a pane in px or %. The default value is `NULL`.
	`minSize`	This specifies the minimum size limit in px or % when resizing a pane. The default value is `NULL`.
	`maxSize`	This specifies the maximum size limit in px or % when resizing a pane. The default value is `NULL`.
	`minWidth`	This specifies the minimum width limit in px or % when resizing a pane. The default value is `NULL`—as small as a pane can go.
	`maxWidth`	This specifies the maximum width limit in px or % when resizing a pane. The default value is `NULL` - as large as a pane can go.
	`minHeight`	This specifies the minimum height limit in px or % when resizing a pane. The default value is `NULL`—as small as a pane can go.
	`maxHeight`	This specifies the maximum height limit in px or % when resizing a pane. The default value is `NULL`—as large as a pane can go.

Component name	Attribute	Description
	`spacingOpen`	This denotes the spacing in px between the current and adjacent panes—when the pane is open. The default value is `6`.
	`spacingClosed`	This denotes the spacing in px between the current and adjacent panes—when the pane is closed. The default value is `6`.
	`initClosed`	If this value is `true`, the pane is closed when the layout is created. The default value is `false`.
	`initHidden`	If this value is `true`, the pane is hidden when the layout is created. No resizer or spacing is visible; it is as if the pane does not exist. The default value is `false`.
	`resizeWhileDragging`	If this value is `true`, the pane is resized while it is being dragged. If `false`, only the helper element is resized. The default value is `false`.
	`maskContents`	This is a useful flag if you are dealing with iframes inside layout panes. If `true`, this option adds a DIV-mask over or inside this pane, so the user can drag (resize) the pane over the iframe. The default value is `false`.
	`maskObjects`	This is a useful flag if you are dealing with objects such as the applets inside layout panes. If this value is true, this option adds IFRAME-mask over or inside this pane to cover the objects/applets. The content-mask will overlay this mask. The default value is `false`.
MasterDetail	`id`	This is a unique identifier for the component in a naming container.
	`rendered`	This Boolean value specifies how the component should be rendered. When set to `false`, the component will not be rendered.

Component name	Attribute	Description
	`binding`	This is an EL expression that refers to a server-side UIComponent instance in a backing bean.
	`level`	This indicates the current level in the flow. It can be arbitrarily initialized and is updated with each navigation. It shows any initial level when a view with the MasterDetail component(s) is displayed. The default value is `1`.
	`contextValue`	This is the context value for the current level. It can be arbitrarily initialized at the beginning and is updated with each navigation. The default value is `NULL`.
	`selectLevelListener`	This is a method with signature int methodName(SelectLevelEvent). The server-side listener is invoked when a navigation attempt takes place (via pe:selectDetailLevel). The returned value defines the level to go. The default value is `NULL`.
	`showBreadcrumb`	This flag denotes whether a breadcrumb navigation is shown. The default value is `true`.
	`showAllBreadcrumbItems`	By default, breadcrumb items are not shown right from the current level. This flag shows all the breadcrumb items right from the current level as disabled. The default value is `false`.
	`breadcrumbAboveHeader`	This Boolean flag allows us to render the breadcrumb above or below the header facet. The default value is `true`.
	`style`	This indicates the style of the main MasterDetail container element. The default value is `NULL`.

Component name	Attribute	Description
	`styleClass`	This indicates the style class of the main MasterDetail container element. The default value is `NULL`.
MasterDetailLevel	`Id`	This is a unique identifier for the component in a naming container.
	`rendered`	This Boolean value specifies how the component is rendered. When set to `false`, the component will not be rendered.
	`binding`	This is an EL expression that refers to a server-side UIComponent instance in a backing bean.
	`level`	It denotes the level of this detail, which is unique inside the masterDetail component. This is a required attribute. The default value is `none`.
	`contextVar`	This variable is used to access the value set by pe:selectDetailLevel when the level is changed. If the value was not set by currently invoking pe:selectDetailLevel, the last set value is used. The default value is `NULL`.
	`levelLabel`	The label for the corresponding breadcrumb item is shown at the top. The default value is `none`.
	`levelDisabled`	This flag indicates whether the breadcrumb item is disabled (non-clickable). The default value is `false`.
SelectDetailLevel	`contextValue`	This context value is passed to the level to be navigated to. The value defined here is accessible by contextVar in MasterDetailLevel. The default value is `none`.

Component name	Attribute	Description
	`listener`	When this listener method is fired before the action/actionListener event of the component that pe:selectDetailLevel is attached to, it is called. The returned value of this listener is used as a new/modified contextValue. The listener passes a new/modified contextValue between levels. The default value is `none`.
	`level`	This is the level to go to. The default value is `none`.
	`step`	This specifies how many steps are left. It can be negative. If `level` and `step` are not specified, then a value of `1` is assumed for `step`. The default value is `none`.
	`preserveInputs`	This specifies a comma- or blank-separated list of client IDs of the editable components whose inputs have to be preserved while the levels get switched. The `@all` symbol is allowed; it means that all the values within the master detail have to be preserved. It makes sense, for example, when the corresponding command/component has the `immediate` attribute set to `true`. If nothing is specified, the input values get cleared.
	`resetInputs`	This specifies a comma- or blank-separated list of client IDs of the editable components whose inputs have to be cleared while the levels get switched. The `@all` symbol is allowed; it means that all the values within the master detail have to be cleared. It takes precedence over client IDs in `preserveInputs`.

Component name	Attribute	Description
	`event`	This denotes the name(s) of one or several accepted events of f:ajax/ p:ajax in case SelectDetailLevel is attached to any component with Ajax behavior. Several events should be defined as a space- or comma-separated list. If no events are specified, all the events will be accepted.
RemoteCommand	`id`	This is a unique identifier for the component in a naming container.
	`rendered`	This Boolean value specifies how the component should be rendered. When set to `false`, the component will not be rendered.
	`binding`	This is an EL expression that refers to a server-side UIComponent instance in a backing bean.
	`value`	This denotes the label of the component.
	`actionListener`	This is to be processed when a command is executed.
	`action`	This is a method expression or string outcome to process when a command is executed.
	`immediate`	This Boolean value determines the phase ID of the action event. When it is `true`, the actions are processed in the Apply Request Values phase; when `false`, they are invoked in the Invoke Application phase.
	`name`	This denotes the name of the command.
	`update`	This specifies the component(s) to be updated with Ajax.
	`process`	This specifies the component(s) to be processed partially instead of the whole view.
	`onstart`	This JavaScript handler is to be executed before the Ajax request begins.

Component name	Attribute	Description
	`oncomplete`	This JavaScript handler is to be executed when the Ajax request is completed.
	`onerror`	This JavaScript handler is to be executed when the Ajax request fails.
	`onsuccess`	This JavaScript handler is to be executed when the Ajax request succeeds.
	`global`	Global Ajax requests are listened to by the ajaxStatus component. Setting `global` to `false` will not trigger ajaxStatus. The default value is `true`.
	`async`	When set to `true`, Ajax requests are not queued. The default value is `false`.
	`partialSubmit`	When enabled, only values related to the partially processed components will be serialized for Ajax instead of the whole form.
	`autoRun`	When set to `true`, the command will be invoked on page load.
Spotlight	`id`	This is a unique identifier for the component in a naming container.
	`rendered`	This Boolean value specifies how the component is to be rendered. When set to `false`, the component will not be rendered.
	`style`	This indicates the style of the main Layout container element. The default value is `NULL`.
	`styleClass`	This indicates the style class of the main Layout container element. The default value is `NULL`.
	`widgetVar`	This is the name of the client-side widget.
	`blocked`	This Boolean value specifies the blocked UI that is to be kept out of the panel.

Component name	Attribute	Description
TimePicker	`id`	This is a unique identifier of the component in a naming container.
	`rendered`	This Boolean value specifies how the component is to be rendered. When set to `false`, the component will not be rendered.
	`binding`	This is an EL expression that refers to a server-side UIComponent instance in a backing bean.
	`value`	This denotes the value of the component.
	`widgetVar`	This denotes the name of the client-side widget.
	`required`	This marks the component as required.
	`validator`	This method expression refers to the method that validates the input.
	`valueChangeListener`	This method-binding expression refers to a method for handling a value change event.
	`requiredMessage`	This message is displayed when the required field's validation fails.
	`converterMessage`	This message is displayed when the conversion fails.
	`validatorMessage`	This message is displayed when validation fails.
	`accesskey`	This access key transfers focus on to the input element.
	`alt`	This is an alternate textual description of the input element.
	`autocomplete`	This controls the browser's autocomplete behavior.
	`dir`	This is a direction for the text that does not inherit directionality.
	`disabled`	This disables the time picker when set to `true`.
	`label`	This is a localized user-presentable name for the component.

Component name	Attribute	Description
	`style`	This indicates the style of the time picker input element.
	`styleClass`	This indicates the style class of the time picker input element.
	`timeSeparator`	This character is used to separate the hours and minutes. The default value is `:`.
	`showPeriod`	This defines whether AM/PM is to be shown along with the selected time. The default value is `false`.
	`dialogPosition`	This indicates the position of the corner of the dialog. The default value is `left top`.
	`inputPosition`	This indicates the position of the corner of the input. The default value is `left bottom`.
	`mode`	This mode specifies the appearance of the time picker. The possible values are `popup`, `spinner`, and `inline`. The default value is `spinner`.
	`startHours`	This specifies the first displayed hour. The possible range is from `0` to `23`. The default value is `0`.
	`endHours`	This specifies the last displayed hour. The possible range is from `0` to `23`. The default value is `23`.
	`startMinutes`	This specifies the first displayed minute. The possible range is from `0` to `55`. The default value is `0`.
	`endMinutes`	This specifies the last displayed minute. The possible range is from `0` to `55`. The default value is `55`.
	`intervalMinutes`	This specifies the interval of displayed minutes. The default value is `5`.
	`rows`	This specifies the number of rows for the input tables. The value should be at least 2 (it makes more sense if you use a multiple of 2). The default value is `4`.

Component name	Attribute	Description
	`showHours`	This defines whether the hours section is displayed. It is set to `false` to get a minute-only dialog. The default value is `true`.
	`showMinutes`	This defines whether the minutes section is displayed. It is set to `false` to get an hours-only dialog. The default value is `true`.
	`showCloseButton`	This shows an **OK** button to confirm the edit. The default value is `false`.
	`showDeselectButton`	This shows the deselect time button. The default value is `false`.
	`showNowButton`	This shows the Now button. The default value is `false`.
	`onHourShow`	This defines a callback to enable/disable certain hours, for example, the function `onHourShow(hour)`. The default value is `NULL`.
	`onMinuteShow`	This defines a callback to enable/disable certain minutes, for example, the function `onMinuteShow(hour, minute)`. The default value is `NULL`.
	`showOn`	This defines when the time picker is shown. The value `focus` is used when the input is focused on, `button` is used when the button trigger element is clicked on, and `both` is used when the input is focused on and when the button is clicked on. The default value is `focus`.
	`locale`	This string or java.util.Locale represents the user locale. The default value is a locale specified in the view root.

Component name	Attribute	Description
(events)	`timeSelect`	This event is fired when an hour/minute is selected.
	`close`	This event is fired when the time picker is closed.
	`beforeShow`	This event is fired before the time picker is rendered and displayed.
Timeline	`id`	This is a unique identifier for the component in a naming container.
	`binding`	This is an EL expression that refers to a server-side UIComponent instance in a backing bean.
	`rendered`	This Boolean value specifies how the component is rendered. When set to `false`, the component will not be rendered.
	`style`	This denotes the inline style of the component.
	`styleClass`	This denotes the container style class.
	`widgetVar`	This denotes the name of the client-side widget.
	`value`	This is an instance of TimelineModel that represents the backing model.
	`var`	This is the name of the request-scoped variable in an underlying object for each iteration in the TimelineEvent.
	`locale`	This denotes the user locale for i18n messages. The attribute can be either a String or a Locale object.
	`timeZone`	This is the target time zone to convert the start/end dates for display. The user would like to see dates in the UI in this time zone. The attribute can either be a String, TimeZone object, or `NULL`. If it is `NULL`, `timeZone` defaults to the time zone that the application is running in.

Component name	Attribute	Description
	`browserTimeZone`	This time zone is the one that the user's browser/PC is running in. It corrects the conversion of the start/end dates in the target's timeZone for display. The attribute can either be a String, TimeZone object, or `NULL`. Note that `browserTimeZone` should be provided if the target timeZone is provided. If `NULL`, `browserTimeZone` defaults to the server's timeZone.
	`height`	This indicates the height of the timeline in pixels, as a percentage, or `auto`. When `height` is set to `auto`, the height of the timeline is automatically adjusted to fit the contents. If not, it is possible that the events might get stacked so high that they are not visible in the timeline. When `height` is set to `auto`, a minimum height can be specified with the option `minHeight`. The default value is `auto`.
	`minHeight`	This specifies the minimum height for the timeline in pixels. It is useful when `height` is set to `auto`. The default value is `0`.
	`width`	This denotes the width of the timeline in pixels or as a percentage. The default value is `100%`.
	`responsive`	This checks whether the timeline container is resized; if so, the timeline is resized. It is useful when the web page is resized. The default value is `true`.
	`axisOnTop`	If this is `false`, the horizontal axis is drawn at the bottom. If `true`, the axis is drawn on the top. The default value is `false`.

Component name	Attribute	Description
	`dragAreaWidth`	This denotes the width of the drag area in pixels. When an event with a date range is selected, it has a drag area on the left and right sides due to which the start or end dates of the event can be manipulated. The default value is `10`.
	`editable`	If this is `true`, the events can be edited, changed, created, and deleted. The events can only be editable when the option `selectable` is set to `true` (default). When `editable` is set to `true`, the timeline can fire Ajax events such as `change`, `edit`, `add`, `delete`, and `drop`. The global setting `editable` can be overwritten for individual events by setting a value in the field `editable`. The default value is `false`.
	`selectable`	If this is `true`, the events on the timeline are selectable. Selectable events can fire Ajax `select` events. The default value is `true`.
	`unselectable`	If this is `true`, you can unselect an item by clicking on the empty space of the timeline. If `false`, you cannot unselect an item as there will always be one item selected. The default value is `true`.
	`zoomable`	If this is `true`, the timeline can be zoomed. When the timeline is zoomed in on, Ajax `rangechange` events are fired. The default value is `true`.
	`moveable`	If this is `true`, the timeline is movable. When the timeline is moved, Ajax `rangechange` events are fired. The default value is `true`.

Component name	Attribute	Description
	`start`	This is the initial start date for the axis of the timeline. If it is not provided, the earliest date present in the events is taken as the start date. The default value is `NULL`.
	`stop`	This is the initial end date for the axis of the timeline. If it is not provided, the latest date present in the events is taken as the end date. The default value is `NULL`.
	`min`	This sets a minimum date for the visible range. It won't be possible to move beyond this minimum value. The default value is `NULL`.
	`max`	This sets a maximum date for the visible range. It won't be possible to move beyond this maximum value. The default value is `NULL`.
	`zoomMin`	This sets a minimum zoom interval for the visible range in milliseconds. It won't be possible to zoom in further than this minimum value. The default value is `10`.
	`zoomMax`	This sets a maximum zoom interval for the visible range in milliseconds. It won't be possible to zoom out further than this maximum value. The default value equals 315,360,000,000,000 ms (about 10,000 years).

Component name	Attribute	Description
	`preloadFactor`	The preload factor is a positive float value or `0`, which can be used for the lazy loading of events. When the lazy loading feature is active, the calculated time range for preloading will be multiplied by the preload factor. The result of this multiplication specifies the additional time range that will be considered for the preload while moving/zooming too. For example, if the calculated time range for preloading is 5 days and the preload factor is `0.2`, the result is *5 x 0.2 = 1 day*. This means that one day backwards and/or one day onwards will be added to the originally calculated time range. The event area to be preloaded is wider in this case. This helps to avoid the frequent time-consuming fetching of events. The default value is `0`.
	`eventMargin`	This is the minimal margin in pixels between events. The default value is `10`.
	`eventMarginAxis`	This is the minimal margin in pixels between events and the horizontal axis. The default value is `10`.
	`eventStyle`	This specifies the style for timeline events. We can choose from `dot` or `box`. The default value is `box`.
	`groupsChangeable`	If this is `true`, items can be moved from one group to another. It is only applicable when groups are used. The default value is `true`.
	`groupsOnRight`	If this is `false`, the group's legend is drawn on the left-hand side of the timeline. If `true`, the group's legend is drawn on the right-hand side. The default value is `false`.

Component name	Attribute	Description
	`groupsWidth`	By default, the width of the group's legend is adjusted to the group names. A fixed width can be set for the group's legend by specifying `groupsWidth` as a string, for example, `200px`. The default value is `NULL`.
	`snapEvents`	If this is `true`, the start and end of an event will be snapped to integer values while moving or resizing the event. The default value is `true`.
	`stackEvents`	If this is `true`, the events are stacked on top of each other to prevent them from overlapping. This option cannot be used in combination with grouped events. The default value is `true`.
	`showCurrentTime`	If this is `true`, the timeline shows a red vertical line that displays the current time. The default value is `true`.
	`showMajorLabels`	By default, the timeline shows both minor and major date labels on the horizontal axis. For example, the minor labels show minutes and the major labels show hours. When `showMajorLabels` is `false`, no major labels are shown. The default value is `true`.
	`showMinorLabels`	By default, the timeline shows both minor and major date labels on the horizontal axis. For example, the minor labels show minutes and the major labels show hours. When `showMinorLabels` is false, no minor labels are shown. When both `showMajorLabels` and `showMinorLabels` are `false`, the horizontal axis will not be visible. The default value is `true`.

Component name	Attribute	Description
	`showButtonNew`	This shows the button `Create new event` in the navigation menu. The default value is `false`.
	`showNavigation`	This shows a navigation menu with buttons to move and zoom the timeline. The default value is `false`.
	`dropHoverStyleClass`	This indicates the style class to be applied when an acceptable draggable is dragged over. The default value is `NULL`.
	`dropActiveStyleClass`	This indicates the style class to be applied when an acceptable draggable is dragged over. The default value is `NULL`.
	`dropAccept`	This is the selector that defines the accepted draggables. The default value is `NULL`.
	`dropScope`	This is the scope key that matches the draggables and the droppables. The default value is `NULL`.
(events)	`add`	This event is fired when an event is added.
	`change`	This event is fired after the user modifies the start or end dates of an event by moving/dragging the event in the timeline.
	`edit`	This event is fired when an event is edited (that is, when the user double-clicks on an event).
	`delete`	This event is fired when an event is deleted (that is, when the user clicks on the "delete event" button on the right-hand side of an event).
	`select`	This event is fired when the user clicks on an event.
	`rangechange`	This event is fired when the visible range changes. It is fired repeatedly as the user modifies the visible time by moving/dragging the timeline or by zooming/scrolling it.

Component name	Attribute	Description
	`rangechanged`	This event is fired when the visible range has been changed. It is fired once after the user has modified the visible time by moving/dragging the timeline or by zooming/scrolling it.
	`lazyload`	This event is fired when the events should be lazy-loaded. It is fired once after the user has modified the visible time by moving/dragging the timeline or by zooming/scrolling it.
	`drop`	This event is fired when an item is dragged and dropped on to the timeline from outside.
Tooltip	`id`	This is a unique identifier for the component in a naming container.
	`rendered`	This Boolean value specifies how the component is rendered. When set to `false`, the component will not be rendered.
	`binding`	This is an EL expression that refers to a server-side UIComponent instance in a backing bean.
	`Value`	This denotes the value of the component that can either be an EL expression or a literal text.
	`converter`	This is an EL expression or a literal text that defines a converter for the component.
	`widgetVar`	This denotes the name of the client-side widget.
	`Global`	This is a global tooltip that converts each title attribute to a tooltip. The default value is `false`.
	`shared`	This denotes a shared tooltip—one tooltip with multiple targets. The default value is `false`.

Component name	Attribute	Description
	`autoShow`	This flag enables the showing of tooltips automatically after page load. The auto shown tooltips cannot be global or shared. The default value is `false`.
	`mouseTracking`	This flag enables the tooltip's position in relation to the mouse. The default value is `false`.
	`fixed`	When set to `true`, the tooltip will not hide if moused over, thus allowing the contents to be clicked on and interacted with. The default value is `false`.
	`adjustX`	This denotes a positive or negative pixel value by which the tooltip in the horizontal plane (*x* axis) is offset. Negative values cause a reduction in the value (moves the tooltip to the left). The default value is `0`.
	`adjustY`	This denotes a positive or negative pixel value by which the tooltip in the vertical plane (*y* axis) is offset. Negative values cause a reduction in the value (moves the tooltip upwards). The default value is `0`.
	`atPosition`	This denotes the corner at which the target element positions the tooltip. The default value is `bottom right`.
	`myPosition`	This denotes the corner at which the tooltip is positioned in relation to the target element. The default value is `top left`.
	`showEvent`	This denotes the event that displays the tooltip. The default value is `mouseenter`.
	`showDelay`	This denotes the delay time for displaying the tooltip. The default value is `0`.
	`showEffect`	This denotes the effect to be used for display. The default value is `fadeIn`.

Component name	Attribute	Description
	`showEffectLength`	This denotes the time in milliseconds that is required to display the effect. The default value is `500`.
	`hideEvent`	This denotes the event that hides the tooltip. The default value is `mouseleave`.
	`hideDelay`	This denotes the delay time taken for hiding the tooltip. The default value is `0`.
	`hideEffect`	This denotes the effect to be used for hiding. The default value is `fadeOut`.
	`hideEffectLength`	This denotes the time in milliseconds that is required to process the hide effect. The default value is `500`.
	`for`	This denotes the ID of the component that the tooltip has to attach to.
TriStateCheckbox	`id`	This denotes the unique identifier for the component in a naming container.
	`rendered`	This Boolean value specifies how the component is to be rendered. When set to `false`, the component will not be rendered.
	`binding`	This is an EL expression that refers to a server-side UIComponent instance in a backing bean.
	`value`	This denotes the value of the component.
	`converter`	This is 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. If it's a static text, it must refer to a converter ID.

Component name	Attribute	Description
	`Immediate`	When this is set to `true`, the process validation logic is executed at the Apply Request Values phase for this component. The default value is `false`.
	`required`	This marks the component as required.
	`validator`	This method expression refers to the method validating the input.
	`valueChangeListener`	This method-binding expression refers to a method for handling a value change event.
	`requiredMessage`	This message is displayed when the required field validation fails.
	`converterMessage`	This message is displayed when the conversion fails.
	`validatorMessage`	This message is displayed when the validation fails.
	`widgetVar`	This is the name of the client-side widget.
	`disabled`	This property disables the component.
	`label`	This denotes the user label.
	`onchange`	This is a callback on the `change` event.
	`style`	This denotes the inline style of the component.
	`styleClass`	This denotes the container's style class.
	`stateOneIcon`	This defines an icon for StateOne as a CSS class.
	`stateTwoIcon`	This defines an icon for StateTwo as a CSS class.
	`stateThreeIcon`	This defines an icon for StateThree as a CSS class.
	`itemLabel`	This defines a label to display next to the checkbox.
	`stateOneTitle`	This defines a title for StateOne.
	`stateTwoTitle`	This defines a title for StateTwo.

Component name	Attribute	Description
	`stateThreeTitle`	This defines a title for StateThree.
	`tabindex`	This denotes the tab index to be assigned to the component. If not specified, a value of `0` will be assigned.
TriStateManyCheckbox	`id`	This is a unique identifier for the component in a naming container.
	`rendered`	This Boolean value specifies how the component is to be rendered. When set to `false`, the component will not be rendered.
	`binding`	This is an EL expression that refers to a server-side UIComponent instance in a backing bean.
	`value`	This denotes the value of the component.
	`converter`	This is 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. If it's a static text, it must refer to a converter ID.
	`immediate`	When this is set to `true`, the process validation logic is executed at the Apply Request Values phase for this component. The default value is `false`.
	`required`	This marks the component as required.
	`validator`	This method expression refers to the method validating the input.
	`valueChangeListener`	This method-binding expression refers to a method for handling a value change event.
	`requiredMessage`	This message is displayed when the required field validation fails.
	`converterMessage`	This message is displayed when the conversion fails.
	`validatorMessage`	This message is displayed when the validation fails.

Component name	Attribute	Description
	`widgetVar`	This is the name of the client-side widget.
	`disabled`	This option disables the component.
	`label`	This denotes a user-presentable name.
	`layout`	This defines the layout of the checkboxes. The valid values are `lineDirection(horizontal)` and `pageDirection(vertical)`.
	`onchange`	This is a callback on the `change` event.
	`style`	This indicates the inline style of the component.
	`styleClass`	This denotes the style class of the main Layout container element. The default value is `NULL`.
	`stateOneIcon`	This defines an icon for StateOne as a CSS class.
	`stateTwoIcon`	This defines an icon for StateTwo as a CSS class.
	`stateThreeIcon`	This defines an icon for StateThree as a CSS class.
	`stateOneTitle`	This defines a title for StateOne.
	`stateTwoTitle`	This defines a title for StateTwo.
	`stateThreeTitle`	This defines a title for StateThree.
	`tabindex`	This defines the tab index to be assigned to the component. If not specified, a value of `0` will be assigned.
QR Code	`id`	This is a unique identifier for the component in a naming container.
	`binding`	This is an EL expression that refers to a server-side UIComponent instance in a backing bean.
	`renderMethod`	The render method can take values `canvas`, `img`, or `div`. The default value is `canvas`.

Component name	Attribute	Description
	`renderMode`	The render method can take values `0` (normal), `1` (label strip), `2` (label box), `3` (image strip), and `4` (image box). The default value is `0`.
	`minVersion`	This defines the version's minimum range. The default value is `1`.
	`maxVersion`	This defines the version's maximum range. The default value is `40`.
	`ecLevel`	This defines the error correction levels `L`, `M`, `Q`, or `H`. The default value is `L`.
	`leftOffset`	This defines the offset in pixels if drawn onto the existing canvas. The default is `0`.
	`topOffset`	This defines the offset in pixels if drawn onto the existing canvas. The default value is `0`.
	`size`	This defines the size in pixels. The default value is `200px`.
	`text`	This defines the QRCode content's text. The default value is `empty`.
	`fontName`	This defines the QRCode content's label font. The default value is `sans`.
	`fontColor`	This defines the QRCode content's label color. The default value is `#000`.
	`label`	This defines the QRCode content's label. The default value is `empty`.
	`fillColor`	This defines the code color or image element. The default value is `#000`.
	`background`	This defines the background color or image element. It is `empty` for a transparent background. The default value is `transparent`.
	`radius`	This defines the corner radius relative to the module width. It ranges from `0.0` to `0.5`. The default value is `0.0`.

Component name	Attribute	Description
	`quiet`	This defines the quiet zone in modules. The default value is `0.0`.
	`mSize`	This defines the label's size percentage. The default value is `0.1`.
	`mPosX`	This defines the label's relative X position. The default value is `0.5`.
	`mPosY`	This defines the label's relative Y position. The default value is `0.5`.
Waypoint	`id`	This is a unique identifier for the component in a naming container.
	`rendered`	This Boolean value specifies how the component is to be rendered. When set to `false`, the component will not be rendered.
	`binding`	This is an EL expression that refers to a server-side UIComponent instance in a backing bean.
	`widgetVar`	This denotes the name of the client-side widget.
	`for`	The target component is registered as a waypoint. If `NULL`, the parent component is taken as the target. The default value is `NULL`.
	`forContext`	The context defines the scrollable element that the waypoint belongs to and acts within. It can be a component or a plain HTML element such as `div`. The default value is `window` if nothing is specified.
	`enabled`	If this is `false`, the waypoint is created but it will be disabled from triggering. You can call the widget's `enable` method to turn it back on. The default value is `true`.

Component name	Attribute	Description
	`horizontal`	The default waypoints live on the vertical axis. Their offset is calculated in relation to the top of the viewport and they listen for vertical scroll changes. If `horizontal` is set to `true`, the offset is calculated in relation to the left of the viewport and it listens for horizontal scroll changes. For instance, the value `50` means "when the left side of this element reaches 50px from the left side of the viewport". The default value is `false`.
	`offset`	This option determines how far the top of the element must be from the top of the viewport to trigger a waypoint (callback function). It can be a number that is taken as a number of pixels (it can also be negative, for example, `-10`), a string representing a percentage of the viewport height (for example, `50%`), or a function that will return a number of pixels. The default value is `0`.
	`continuous`	If true, and multiple waypoints are triggered in one scroll, this waypoint will trigger even if it is not the last waypoint reached. If false, it will only trigger if it is the last waypoint. The default value is true.
	`triggerOnce`	If this is true, the waypoint will be destroyed when triggered. The default value is `false`.

Component name	Attribute	Description
(events) Utitlity functions and components	`reached`	This event is fired when the user scrolls past the element.
ClientBehaviors (JavaScript)	`execute`	This denotes the JavaScript that should be executed.
	`disabled`	This Boolean value specifies how the behavior is to be rendered. When set to `false`, the behavior will not be rendered.
	`event`	This denotes the name of the event.
Converters		
JSONConverter (ConvertJSON)	`type`	This denotes the datatype of the value object (optional). Any primitive type, array, non generic, or generic type is supported. A datatype is sometimes required to convert a value to a JSON representation. All datatypes should be fully qualified. The default value is `NULL`.
LocaleConverter (ConvertLocale)	`separator`	This denotes the character that will be used to separate the country and language. The default value is _.
ImportConstants	`className`	This denotes the constants class.
	`var`	This EL variable can be used to obtain constants. The default value is the name of the class without the package.
ImportEnum	`type`	This denotes the enum class.
	`var`	This EL variable can be used to obtain the enum values. The default value is the name of the class without the package.
	`allSuffix`	This indicates the suffix mapping for an array with all the enum values. The default value is `ALL_VALUES`.

Component name	Attribute	Description
Switch	`id`	This is a unique identifier for the component in a naming container.
	`rendered`	This Boolean value specifies how the component is to be rendered. When set to `false`, the component will not be rendered.
	`binding`	This is an EL expression that refers to a server-side UIComponent instance in a backing bean.
	`value`	This denotes the value of the component.