Inline Edit Configuration Options

Name	Description	Values	Default
`selectors`	Javascript object containing selectors for various fragments of the Inline Edit component.	The object can contain any subset of the following keys: `text` `editContainer` `edit` `textEditButton` (New in v1.3) Any values not provided will revert to the default.	`selectors: { text: ".flc-inlineEdit-text", editContainer: ".flc-inlineEdit-editContainer", edit: ".flc-inlineEdit-edit", textEditButton: ".flc-inlineEdit-textEditButton" // New in v1.3 }` NOTE: The default `editModeRenderer` uses the default selector "`.flc-inlineEdit-edit`". If you are using the default `editModeRenderer`, do not over-ride this selector.
New in v1.3: `strings`	Configuration of short messages and strings which the component uses in its UI	key-value structure with `string` values	`strings: { textEditButton: "Edit text %text", editModeInstruction: "Press Escape to cancel, Enter or Tab when finished." }`
`listeners`	JavaScript object containing listeners to be attached to the supported events.	Keys in the object are event names, values are functions or arrays of functions.	See fluid:Supported Events

`styles`	Javascript object containing CSS style names that will be applied to the Inline Edit component.	The object can contain any subset of the following keys:
		`text`
		`edit`
		`invitation`
		`defaultViewStyle`
		`emptyDefaultViewText` (New in v1.3)
		`focus`
		`tooltip`
		`editModeInstruction` (New in v1.3)
		`displayView` (New in v1.3)
		`textEditButton` (New in v1.3)
		
		Any values not provided will revert to the default.
			`styles: {`
			`    text: "fl-inlineEdit-text",`
			`    edit: "fl-inlineEdit-edit",`
			`    invitation: "fl-inlineEdit-invitation",`
			`    defaultViewStyle: "fl-inlineEdit-emptyText-invitation",`
			`emptyDefaultViewText: "fl-inlineEdit-emptyDefaultViewText",`
			`    focus: "fl-inlineEdit-focus",`
			`    tooltip: "fl-inlineEdit-tooltip",`
			`editModeInstruction: "fl-inlineEdit-editModeInstruction", // New in v1.3`
			`    displayView: "fl-inlineEdit-simple-editableText fl-inlineEdit-textContainer", // New in v1.3`
			`    textEditButton: "fl-offScreen-hidden" // New in v1.3`
			`}`
			
			NOTE: The default `editModeRenderer` uses the default style `fl-inlineEdit-edit`. If you are using the default `editModeRenderer`, do not over-ride this style.

`paddings`	Javascript object containing pixel values that will configure the size of the edit field.	The object can contain any subset of the following keys:
		`edit`
		`minimumEdit`
		`minimumView`
		Any values not provided will revert to the default.
			`paddings: {`
			`    edit: 10,`
			`    minimumEdit: 80,`
			`    minimumView: 60`
			`}`

`applyEditPadding`	Indicates whether the values stored for `edit` and `minimumEdit` will be applied to the edit mode or ignored	boolean	`true`
`submitOnEnter`	Determines whether receiving an "Enter" keypress will cause the component to finish editing and commit the changed value. If this is given a direct value `true` or `false`, the value will be honoured. If the value is left at `undefined`, the default behaviour will be inferred from the tag peering with the `editField` selector - a `<input>`> tag will cause submission, whereas a `<textarea>` will not.	boolean	`undefined`
New in v1.3: `displayModeRenderer`	A function that calls upon the markup corresponding to the display mode of the component.	function - `InlineEditRenderer` See #InlineEdit Types for more info.	`defaultDisplayModeRenderer`, a function that creates the `displayModeRenderer`

`editModeRenderer`	A function that creates or recognises the markup corresponding to the editable view of the component. The function is intended to inspect the state of the existing markup, and if it is "incomplete" in some way, to fill in the required fields. In all cases, the function returns a structure
	`return {`
	`  container: [jQuery],`
	`  field: [jQuery]`
	`};`
	where `container` is a container element for the edit field and `field` is the editable field itself. The value held within `field` is intended to be stored and retrieved in the document using the `editAccessor` function, which is a ViewAccessor
		function - `InlineEditRenderer` See #InlineEdit Types for more info.	`defaultEditModeRenderer`, a function that creates the edit field based on the following template:
			`<span><input type='text' class='flc-inlineEdit-edit fl-inlineEdit-edit'/></span>`

`displayAccessor`	A ViewAccessor, or name of one, which is used to store and retrieve the editable value from the read-only view of the control, peering with the tag referenced by the selector `text`. The standard accessor uses the standard jQuery functions `val` or `text` depending on the tag type.	`ViewAccessor` See #InlineEdit Types for more info.	"fluid.inlineEdit.standardAccessor"
`displayView`	A InlineEditView, or name of one, which is used to operate the implementation of the `refreshView` method as applied to the read-only view of the component. This contains a single method named `refreshView` which brings the state of this view in line with the component's model, performing any work (in general, dealing with any default text) in addition to the raw value-fetching work done by the `displayAccessor`	`InlineEditView` See #InlineEdit Types for more info.	"fluid.inlineEdit.standardDisplayView"
`editAccessor`	As for `displayAccessor`, but for use when the editable view of the component is shown	`ViewAccessor` See #InlineEdit Types for more info.	"fluid.inlineEdit.standardAccessor"
`editView`	As for `displayView`, but for use when the editable view of the component is shown	`InlineEditView` See #InlineEdit Types for more info.	"fluid.inlineEdit.standardEditView"
`lazyEditView`	For `editRenderer` s which are particularly expensive, for example, those which instantiate a rich text editing component, it is valuable to delay their rendering until they are required. Setting `lazyEditView` to `true` will ensure that the `editRenderer` is not triggered until the component is sent into edit mode.	boolean	false
`blurHandlerBinder`	A function which acts on the overall component to bind a handler for the `blur` event received on the editable view. For integrations where the editable view is a complex collection of elements, such as dropdown inlineEdit, this needs to take an arbitrary form. A standard implementation is provided as `fluid.deadMansBlur` which will infer that focus is leaving a set of elements if none of them receives a `focus` after a `blur` within a 150 millisecond horizon	`function (that)`	null
`selectOnEdit`	Indicates whether or not to automatically select the editable text when the component switches into edit mode.	boolean	`false`
`defaultViewText`	The default text to use when filling in an empty component. Set to empty to suppress this behaviour	string	"Click here to edit"
`useTooltip`	Indicates whether or not the component should display a custom ("invitation") tooltip on mouse hover	boolean	`false`
`tooltipText`	The text to use for the tooltip to be displayed when hovering the mouse over the component	string	"Click item to edit"
`tooltipId`	The id to be used for the DOM node holding the tooltip	string	"tooltip"
`tooltipDelay`	The delay, in ms, between starting to hover over the component and showing the tooltip	number	1000

Additional options for Multiple Inline Edits

The options for the creation of multiple Inline Edits are the same as those for the creation of a single Inline Edit, with the addition of a selector for identifying the editable elements. The default selector is defined as follows:

```
selectors: {
    editables: ".flc-inlineEditable"
}
```

InlineEdit Types

Several of the InlineEdit configuration elements make use of various "Implicit" or "Duck Typed" objects which have particular structures or signatures.

Type name	Description	Layout
`ViewAccessor`	Appears as `displayAccessor` and `editAccessor`. Used to convey updates to and from the model to its representation in the DOM. Exposes a single function `value` with the same semantics as `jQuery.val()`.	`value: function([optional value]) }`
`InlineEditView`	Appears as `displayView` and `editView`. Used to wrap the action of the relevant ViewAccessor as it maintains synchrony between the model and DOM. For some views, especially where there is some "default text" to invite the user to edit, there is extra formality to displaying the model which is InlineEdit-specific, rather than markup-specific. Such logic goes in this class, and is less frequently user-configured.	`{ refreshView: function (that, source) }`
`InlineEditRenderer`	Appears as `editModeRenderer`. Actually a function, rather than a structure, with a fairly complex contract. Is passed the entire component `that` in order to inspect the current markup situation at startup time, to manipulate it if necessary to render and initialise the editable component view, and return the relevant nodes which it has either created or discovered.	`function (that) -> { container, field }`