Simple Text Inline Edit API

This page is being upgraded to the new look and feel.

Production Status: PRODUCTION The **Inline Edit** component allows users to edit content within the context of their work rather than going to an "edit mode". It can be applied to any text, assuming a very simple contract is maintained:

1. The elements that are part of the Inline Edit component have some kind of container element.
2. The text you wish to make editable is within the component container.

You can optionally provide your own markup that will be used for the edit mode, but if not, default markup is provided.

See Also

Inline Edit Component Description
Simple Text Inline Edit Component Description
Inline Edit API
Dropdown Inline Edit API
Rich Text Inline Edit API
Tutorial - Simple Text Inline Edit
About Infusion Components

Creator	`fluid.inlineEdit(container, options)` Creates a single inline edit.
Supported Events	**event** Event description
Methods	*none*

Options	**`listeners`**
	See Supported Events for information
	
	`opt`
	Option description
	
	`opt`
	Option description

Creator	`fluid.inlineEdits(container, options)` Creates an array of inline edit components.
Supported Events	**event** Event description
Methods	*none*

Options	**`listeners`**
	See Supported Events for information
	
	`opt`
	Option description
	
	`opt`
	Option description

Creators

back to top Use the following function to create a single Simple Text Inline Edit component:

Method	`fluid.inlineEdit(container, options);`
Description	Instantiate a single Inline Edit component.

Parameters	*`container`*
	A CSS-based selectors, single-element jQuery object, or DOM element that identifies the root DOM node of the Inline Edit markup.
	
	`options`
	An optional data structure that configures the Inline Edit component, as described below

Returns	The Inline Edit component

Examples	
	```
	var myEdit = fluid.inlineEdit("#myContainer", {
	    strings: {
	        defaultViewText: "Edit here"
	    }
	});
	```

Use the following function to create multiple Simple Text Inline Edit components:

Method	`fluid.inlineEdits(container, options);`
Description	Instantiate multiple Inline Edit components.

Parameters	*`container`*
	A CSS-based selectors, single-element jQuery object, or DOM element that identifies the root DOM node where the UI Options interface should be placed.
	
	`options`
	An optional data structure that configures the Inline Edit components, as described below

Returns	An array of Inline Edit components

Examples	
	```
	var myEditors = fluid.inlineEdits("#myContainer", {
	    strings: {
	        defaultViewText: "Edit here"
	    }
	});
	```

Supported Events

back to top

`afterInitEdit`

Description	This event is fired by the Rich Text Editors when the editing interface is fully initialized.
Type	default
Parameters	*editor* the instance of the editor component

`afterBeginEdit`

Description	This event fires after the editing view has initialized and is ready for editing.
Type	default
Parameters	*none*

`onCreateEditView`

Description	This event fires when the Inline Edit component creates the editing view.
Type	default
Parameters	*none*

`modelChanged`

Description	This event fires when the value of the editable field has changed.
Type	default
Parameters	*model* The current (new) value of the "model" structure representing the editable state of the component *oldModel* A snapshot of the old value of the model structure before the current edit operation started *source* An arbitrary object which optionally represents the "source" of the change, which can be checked by listeners to prevent cyclic events. Can often be undefined.

`onBeginEdit`

Description	This event fires before the Inline Edit component switched into edit mode.
Type	preventable
Parameters	*none*

`onFinishEdit`

Description	This event fires before the Inline Edit component is switched out of edit mode back into view mode.
Type	preventable

Parameters	*`newValue`*
	see parameters for `modelChanged` (model, oldModel)
	
	`oldValue`
	see parameters for `modelChanged` (model, oldModel)
	
	`editNode`
	A DOM node which holds the concrete editable control - this may vary in interpretation for different embodiments of the InlineEdit control but may, for example be an `<input>`, `<textarea>` or `<select>` node
	
	`viewNode`
	A DOM node which holds the read-only representation of the editable value.

`afterFinishEdit`

Description	This event fires when the Inline Edit component has been switched out of edit mode back into view mode.
Type	default

Parameters	*`newValue`*
	see parameters for `modelChanged` (model, oldModel)
	
	`oldValue`
	see parameters for `modelChanged` (model, oldModel)
	
	`editNode`
	A DOM node which holds the concrete editable control - this may vary in interpretation for different embodiments of the InlineEdit control but may, for example be an `<input>`, `<textarea>` or `<select>` node
	
	`viewNode`
	A DOM node which holds the read-only representation of the editable value.

Options

back to top The second argument to the creator function is the options argument. This is a JavaScript object containing name/value pairs: The name is the name of the option and the value is the desired setting. Components define their own default values for options, but integrators can override these defaults by providing new values using the options argument. For technical information about how options are merged with defaults, see Options Merging.

```
var myEdit = fluid.inlineEdit(".myContainer", {
    <option1Name>: <option1value>,
    <option2Name>: <option2value>
    ...
});
```

The options supported by Inline Edit are described below.

selectors

strings

listeners

styles

paddings

applyEditPadding

submitOnEnter

displayModeRenderer

editModeRenderer

displayAccessor

displayView

editAccessor

editVIew

lazyEditView

blurHandlerBinder

Description	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.
Default	null

Example	`fluid.inlineEdit("#myContainer", {`
	`    blurHandlerBinder: ""`
	`});`

selectOnEdit

Description	Indicates whether or not to automatically select the editable text when the component switches into edit mode.
Default	false

Example	`fluid.inlineEdit("#myContainer", {`
	`    selectOnEdit: true`
	`});`

defaultViewText

Description	The default text to use when filling in an empty component. Set to empty to suppress this behaviour.
Default	"Click here to edit"
Example	`fluid.inlineEdit("#myContainer", {     defaultViewText: "" });`

`useTooltip`

Description	Indicates whether or not the component should display a custom ("invitation") tooltip on mouse hover
Default	false
Example	`fluid.inlineEdit("#myContainer", {     useTooltip: true });`

`tooltipText`

Description	The text to use for the tooltip to be displayed when hovering the mouse over the component
Default	"Click item to edit"
Example	`fluid.inlineEdit("#myContainer", {     tooltipText: "Click to edit" });`
See also	`useTooltip`

`tooltipID`

Description	The id to be used for the DOM node holding the tooltip
Default	"tooltip"
Example	`fluid.inlineEdit("#myContainer", {     tooltipID: "myTooltip" });`
See also	`useTooltip`

`tooltipDelay`

Description	The delay, in ms, between starting to hover over the component and showing the tooltip
Default	1000
Example	`fluid.inlineEdit("#myContainer", {     tooltipDelay: 500 });`
See also	`useTooltip`

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 }`

Inline Edit Overview

The Inline Edit allows users to edit content within the context of their work rather than going to an "edit mode". It can be applied to any text, assuming a very simple contract is maintained:

1. The elements that are part of the Inline Edit component have some kind of container element.
2. The text you wish to make editable is within the component container.

You can optionally provide your own markup that will be used for the edit mode, but if not, default markup is provided.

Creation

Creating a single Inline Edit

```
fluid.inlineEdit(componentContainer, options);
```

Return: The Inline Edit component object.

Creating Multiple Inline Edits

```
fluid.inlineEdits(componentContainer, options);
```

Return: An array of the Inline Edit component objects.

This function will find any elements within the given container that are identified as 'editables' and apply the Inline Edit component to them.

Status
This component is in Production status

On This Page

See Also

- Inline Edit Component Description
- Simple Text Inline Edit Component Description
- Inline Edit API
- Dropdown Inline Edit API
- Rich Text Inline Edit API
- Tutorial - Simple Text Inline Edit
- About Infusion Components

Still need help?

Join the infusion-users mailing list and ask your questions there.

Parameters

componentContainer

The `componentContainer` parameter is a selector, a single-element jQuery, or a DOM element specifying the root DOM node of the Inline Edit markup.

options

The `options` parameter is an optional data structure that configures the Inline Edit component(s), as described below in the fluid:Options section.

Supported Events

The Inline Edit component fires the following events (for more information about events in the Infusion Framework, see Events):

`afterInitEdit`

Description	This event is fired by the Rich Text Editors when the editing interface is fully initialized.
Type	default
Parameters	*editor* the instance of the editor component

`afterBeginEdit`

Description	This event fires after the editing view has initialized and is ready for editing.
Type	default
Parameters	*none*

`onCreateEditView`

Description	This event fires when the Inline Edit component creates the editing view.
Type	default
Parameters	*none*

`modelChanged`

Description	This event fires when the value of the editable field has changed.
Type	default

Parameters	*model* The current (new) value of the "model" structure representing the editable state of the component *oldModel* A snapshot of the old value of the model structure before the current edit operation started *source* An arbitrary object which optionally represents the "source" of the change, which can be checked by listeners to prevent cyclic events. Can often be undefined.

`onBeginEdit`

Description	This event fires before the Inline Edit component switched into edit mode.
Type	preventable
Parameters	*none*

`onFinishEdit`

Description	This event fires before the Inline Edit component is switched out of edit mode back into view mode.
Type	preventable

Parameters	*`newValue`*
	see parameters for `modelChanged` (model, oldModel)
	
	`oldValue`
	see parameters for `modelChanged` (model, oldModel)
	
	`editNode`
	A DOM node which holds the concrete editable control - this may vary in interpretation for different embodiments of the InlineEdit control but may, for example be an `<input>`, `<textarea>` or `<select>` node
	
	`viewNode`
	A DOM node which holds the read-only representation of the editable value.

`afterFinishEdit`

Description	This event fires when the Inline Edit component has been switched out of edit mode back into view mode.
Type	default

Parameters	*`newValue`*
	see parameters for `modelChanged` (model, oldModel)
	
	`oldValue`
	see parameters for `modelChanged` (model, oldModel)
	
	`editNode`
	A DOM node which holds the concrete editable control - this may vary in interpretation for different embodiments of the InlineEdit control but may, for example be an `<input>`, `<textarea>` or `<select>` node
	
	`viewNode`
	A DOM node which holds the read-only representation of the editable value.

Functions

These functions are defined on the central `component` object returned from the inlineEdit creator function - for example with

```
var myEdit = fluid.inlineEdit(componentContainer, options);
```

```
myEdit.edit();
```

Switches the component into edit mode. The events `onBeginEdit` and `afterBeginEdit` will fire.

```
myEdit.finish();
```

Switches the component out of edit mode into display mode, updating the displayed text with the current content of the edit field. The events `onFinishEdi` and `afterFinishEdit` will fire. If the model value has changed, there will be a call to `modelUpdated` in between these calls.

```
myEdit.cancel();
```

Cancels the in-progress edit and switches back to view mode.

```
myEdit.isEditing();
```

Determines if the component is currently in edit mode: Returns true if edit mode is shown, false if view mode is shown.

```
myEdit.refreshView(source);
```

Updates the state of the inline editor in the DOM, based on changes that may have happened to the model. `source` is an optional source object identifying the source of the change (see ChangeApplier documentation)

```
myEdit.tooltipEnabled();
```

Returns a boolean indicating whether or not the tooltip is enabled.

```
/**
  * Pushes external changes to the model into the inline editor, refreshing its
  * rendering in the DOM. The modelChanged event will fire.
  *
  * @param {String} newValue The bare value of the model, that is, the string being edited
  * @param {Object} source An optional "source" (perhaps a DOM element) which triggered this event
  */
myEdit.updateModelValue(newValue, source);
```

Updates the component's internal representation of the text to a new value. If the value differs from the existing value, the `modelChanged` event will fire and the component will be re-rendered.

```
/**
  * Pushes external changes to the model into the inline editor, refreshing its
  * rendering in the DOM. The modelChanged event will fire.
  *
  * @param {Object} newValue The full value of the new model, that is, a model object which
  *      contains the editable value as the element named "value"
  * @param {Object} source An optional "source" (perhaps a DOM element) which triggered this event
  */
myEdit.updateModel(newValue, source);
```

Similar to `updateModelValue`, only accepts specification of the overall model object (housing the editable value under the key `value`).

```
myEdit.model
```

Not a function, but a data structure. This directly represents the "model" or state of the editable component. External users should consider this structure as read-only, and only make modifications through the `updateModel` call above.

Options

The following options to the creator functions can be used to customize the behaviour of the Inline Edit component:

selectors

strings

listeners

styles

paddings

applyEditPadding

submitOnEnter

displayModeRenderer

editModeRenderer

displayAccessor

displayView

editAccessor

editVIew

lazyEditView

blurHandlerBinder

Description	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.
Default	null

Example	`fluid.inlineEdit("#myContainer", {`
	`    blurHandlerBinder: ""`
	`});`

selectOnEdit

Description	Indicates whether or not to automatically select the editable text when the component switches into edit mode.
Default	false

Example	`fluid.inlineEdit("#myContainer", {`
	`    selectOnEdit: true`
	`});`

defaultViewText

Description	The default text to use when filling in an empty component. Set to empty to suppress this behaviour.
Default	"Click here to edit"

Example	`fluid.inlineEdit("#myContainer", {`
	`    defaultViewText: ""`
	`});`

`useTooltip`

Description	Indicates whether or not the component should display a custom ("invitation") tooltip on mouse hover
Default	false

Example	`fluid.inlineEdit("#myContainer", {`
	`    useTooltip: true`
	`});`

`tooltipText`

Description	The text to use for the tooltip to be displayed when hovering the mouse over the component
Default	"Click item to edit"

Example	`fluid.inlineEdit("#myContainer", {`
	`    tooltipText: "Click to edit"`
	`});`

See also	`useTooltip`

`tooltipID`

Description	The id to be used for the DOM node holding the tooltip
Default	"tooltip"

Example	`fluid.inlineEdit("#myContainer", {`
	`    tooltipID: "myTooltip"`
	`});`

See also	`useTooltip`

`tooltipDelay`

Description	The delay, in ms, between starting to hover over the component and showing the tooltip
Default	1000

Example	`fluid.inlineEdit("#myContainer", {`
	`    tooltipDelay: 500`
	`});`

See also	`useTooltip`

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 }`

Skinning

This component can be skinned "out of the box" when you include the component's CSS files. Just be sure to put the following in your document:

```
<link rel="stylesheet" type="text/css" href="components/inlineEdit/css/InlineEdit.css" />
```

Dependencies

The Inline Edit dependencies can be met by including the `MyInfusion.js` file in the header of the HTML file:

```
<script type="text/javascript" src="MyInfusion.js"></script>
```

Alternatively, the individual file requirements are:

```
<script type="text/javascript" src="lib/jquery/core/js/jquery.js"></script>
<script type="text/javascript" src="lib/jquery/ui/js/jquery.ui.core.js"></script>
<script type="text/javascript" src="lib/jquery/ui/js/jquery.ui.widget.js"></script>
<script type="text/javascript" src="lib/jquery/ui/js/jquery.ui.position.js"></script>
<script type="text/javascript" src="lib/jquery/plugins/tooltip/js/jquery.ui.tooltip.js"></script>
<script type="text/javascript" src="framework/core/js/Fluid.js"></script>
<script type="text/javascript" src="framework/core/js/jquery.keyboard-a11y.js"></script>
<script type="text/javascript" src="framework/core/js/FluidDocument.js"></script>
<script type="text/javascript" src="framework/core/js/DataBinding.js"></script>
<script type="text/javascript" src="framework/core/js/FluidView.js"></script>
<script type="text/javascript" src="framework/core/js/FluidIoC.js"></script>
<script type="text/javascript" src="components/tooltip/js/Tooltip.js"></script>
<script type="text/javascript" src="components/inlineEdit/js/InlineEdit.js"></script>
<script type="text/javascript" src="components/undo/js/Undo.js"></script>
```