Fluid Component API

This page describes the Fluid standard conventions for defining components of different types. Following these standards to a greater or lesser extent will result in differing levels of support and interoperability with other parts of the Fluid Framework.

General recommendations

A Fluid component should be

- **DOM agnostic**: components should work with a variety of markup, and not make assumptions about the structure of the DOM. Any assumptions made by the component should be expressed in the form of selectors, supported by the framework's DOM Binder.
- **Accessible**: components should be equally usable with either the mouse or the keyboard. ARIA roles and states should be used to ensure support for assistive technologies. Use the framework's keyboard-a11y plugin and the jQuery UI ARIA support.
- **`this`-free**: Avoid use of JavaScript's `this` and `new` keywords, in favour of units and `that`-ism .
- **Testable**: Components should be readily testable using the jqUnit testing toolkit. Tests should be provided for all component functions.

On this Page

See Also

- DOM agnosticism
- DOM Binder
- Infusion Event System
- Component Model Interactions and API
- How to Create a Fluid Component

Still need help?

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

Basic component API

This section describes the basic level of support required for a Fluid component.

Creating a component

A component should provide a creator function that is responsible for creating a new instance of the component. This function should accept at least two arguments:

1. The container for the component. This should always be the *first* argument.
2. An `options` object containing configuration options for the component. This should always be the *last* argument.

Containers

The first argument to a component's creator function represents a container for the component.

It is assumed that

- A `container` is a jQueryable element; a string representing a selector, a DOM element, or a jQuery.
- All markup belonging to the component, and all DOM nodes that the component will directly modify are nested below the `container` node
- When this DOM node is hidden, all visible evidence of the existence of the component will also be hidden.

Options

The last argument to a component creator will be `options`, a JavaScript object specifying the details of configuration of the component. Some of the structure of `options` is standardised; at bare minimum, it should contain the following properties:

Property path	Type	Description
`selectors`	hash of strings	a set of selector strings which, when scoped to the `container` passed as the first argument, will nominate parts of the DOM which are "of interest" to the component. For information on naming conventions, see Class Name Conventions.
`styles` (optional)	hash of strings	Unlike `selectors`, these need to be globally namespaced, since they are expected to be referenced in style sheets to apply styles to parts of the component markup. For information on naming conventions, see Class Name Conventions.
`mergePolicy` (optional)	object	A "policy object" whose keys are EL paths into the options structure itself, and which encodes any special instructions to the framework to be followed when merging the user's options against any registered defaults. Documented on its own page at Options Merging for Infusion Components
`listeners` (optional)	hash of functions	a set of listener funtions to attach to supported events.
`strings` (optional)	hash of strings	a set of strings to inject into the user interface.

`options` and defaults

The `options` structure is laid out almost identically to a "master options structure" which is registered using the `fluid.defaults` utility. This structure defines the default values to be used in the case that implementors do not fill in an option. For example, for a component named `inlineEdit`, the defaults registration might begin like this:

```
fluid.defaults("inlineEdit", {
    selectors: {
        text: ".text",
        editContainer: ".editContainer",
        edit: ".edit"
    },
```

This example defines default selectors for the Inline Edit component. As the componenent's creator starts up, it will merge together the user's instance `options` with the version from `defaults` using `jquery.extend` to produce its runtime options.

Example of a component creator function

```
// A component creator function:
fluid.myComponent = function(container, options) {

};
```

Runtime structure of a Fluid component

The `that` for a Fluid component, once it is created, will contain the following structure at top level:

Member	Description
`container`	A DOM node holding the component (identical to the `container` argument to the component, see description above)
`options`	The "merged" set of options to the component (see description of `defaults` above)
`events`	event firers for any events that the component supports (see Infusion Event System for information)

Runtime structure of a "model-bearing" component

A Fluid component may be structured around a "model" (in the MVC sense) - conditions on "reasonable models" are described on Component Model Interactions and API. Model-bearing components have more requirements on the runtime structure of their `that` - in addition to the two basic members above, they will also possess the following:

Member	Description
model	A "reasonable model" for the component (that is, one that consists of pure data)
refreshView	A function which can accept zero arguments, which when invoked will refresh all of the visible structure of the component to bring it in step with the current state of `model`
modelFirer	A structure as dispensed from `fluid.event.getEventFirer` - this allows listeners to subscribe to updates to the model, as well as update events to be fired.