Tutorial - A Simple IoC Component

DRAFT

On This Page

See Also

- IoC - Inversion of Control
- Declarative Configuration
- Understanding Infusion Components
- Component Grades
- Events
- DOM Binder
- Infusion Framework Best Practices

Still need help?

Join the infusion-users mailing list and ask your questions there,
or hang out in our IRC Channel.

This tutorial is based on the Five-Star Widget code that is used as part of our Keyboard Accessibility Plugin demo. It assumes that:

- you are already familiar with HTML, Javascript and CSS,
- you're familiar with Inversion of Control (IoC) in general,
- you want to see how the Infusion Framework uses IoC in a component.

While the Five-Star Widget is used in a demo of our Keyboard Accessibility jQuery plugin, the Widget is, by itself, a fine example of an IoC component, and it's not necessary to know about the plugin to follow this tutorial.

Overview

The Five-Star Widget is a simple Infusion Component that allows users to "rank" something. The widget displays 5 stars and responds to user selection, internally storing the selection. In our Keyboard Accessibility Plugin demo, it's used to rank images in an image viewer, but it could be incorporated into a movie database, restaurant review site, etc.

The code for the Five-Star Widget embodies one of the main goals of the Infusion Framework's Inversion of Control system: To reduce component creation to (ideally) configuration information only.

Component Defaults

The core of the component definition is the declaration of the component defaults. This is carried out by a call to fluid.defaults(). The definition is show in its entirety below. We'll walk through each part in turn, providing links to more documentation about the various concepts and functionality as we go.

```
fluid.defaults("demo.fiveStar", {
    gradeNames: ["fluid.viewComponent", "autoInit"],
    model: {
        rank: 1
    },
    listeners: {
        onCreate: [{
            funcName: "demo.fiveStar.setARIA",
            args: "{that}"
        }, {
            funcName: "demo.fiveStar.bindChangeListener",
            args: "{that}"
        }, {
            "this": "{that}.stars",
            method: "mouseover",
            args: {
                expander: {
                    funcName: "demo.fiveStar.makeStarHandler",
                    args: ["{that}", "{that}.hoverStars"]
                }
            }
        }, {
            "this": "{that}.container",
            method: "mouseout",
            args: "{that}.refreshView"
        }, {
            "this": "{that}.stars",
            method: "click",
            args: {
                expander: {
                    funcName: "demo.fiveStar.makeStarHandler",
                    args: ["{that}", "{that}.setRank"]
                }
            }
        }]
    },
    selectors: {
        stars: "[class^='star-']"
    },
    starImages: {
        blank: "../images/star-blank.gif",
        hover: "../images/star-orange.gif",
        select: "../images/star-green.gif"
    },
    members: {
        stars: "{that}.dom.stars"
    },
    invokers: {
        setRank: {
            func: "{that}.applier.requestChange",
            args: ["rank", "{arguments}.0"]
        },
        renderStarState: {
            funcName: "demo.fiveStar.renderStarState",
            args: ["{that}.stars", "{arguments}.0", "{that}.model.rank", "{that}.options.starImages"]
        },
        hoverStars: {
            func: "{that}.renderStarState"
        },
        refreshView: {
            func: "{that}.renderStarState",
            args: 0
        }
    }
});
```

Examining the Component Defaults in Detail

Infusion components are created by passing the component name and component configuration options to fluid.defaults(), as seen on line #1:

```
fluid.defaults("demo.fiveStar", {...});
```

All components must define one or more grades using the `gradeNames` option, as see on line #2:

```
    gradeNames: ["fluid.viewComponent", "autoInit"],
```

The `demo.fiveStar` component is defined to be a view component, one of the grades provided by the Infusion Framework. It also uses the `"autoInit"` keyword, which instructs the Framework to automatically define the creator function for the component (the creator function is how the component will be instantiated).

View components are model-bearing components, and therefore must define a model using the `model` component option. This is done on lines 3-5:

```
    model: {
        rank: 1
    },
```

For the Five-Star widget, the model is simple, containing only a single member, `rank`, which stores the user's selection.

On line #6, we find the `listeners` option, where event listeners are declared for various events:

```
    listeners: {...},
```

This is where a lot of a component's functionality is configured, so we'll examine this in more detail a bit later in the tutorial.

On line #37, we find the `selectors` option:

```
    selectors: {
        stars: "[class^='star-']"
    },
```

Since view components are concerned with the DOM, it must have a list of CSS-style selectors that identify whatever parts of the DOM it cares about. These selectors are used by the DOM Binder to facilitate easy lookup and manipulation of the DOM elements. The DOM Binder automatically scopes any use of the selector to the container of the component, preventing conflicts when more than one instance of a component is present on the page.

The widget only defines one selector, `stars`, which is used to find the actual images that users will interact with.

On line #40, we find a component-specific option called `starImages`:

```
    starImages: {
        blank: "../images/star-blank.gif",
        hover: "../images/star-orange.gif",
        select: "../images/star-green.gif"
    },
```

So far, all the options we've seen are expected by the Framework and use reserved option names. The `starImages` option is the first (and only) general option defined by the Five-Star widget. This option defines the path to the different `gif` files used for different states of the stars.

On line #45, we find the `members` option:

```
    members: {
        stars: "{that}.dom.stars"
    },
```

This option creates a property, or member, called `stars` at the top level of the component, and sets its value to the DOM Binder value created based on the `stars` selector (defined on line #38).

Finally, on line #48, we find the `invokers` option, where methods are added to the component:

```
    invokers: {...}
```

We'll examine how invokers are configured in more detail a bit later in the tutorial.

Event Listeners

The `listeners` component option defined functions to attach to component events as listeners. The `listeners` option for the Five-Star widget is shown below:

```
    listeners: {
        onCreate: [{
            funcName: "demo.fiveStar.setARIA",
            args: "{that}"
        }, {
            funcName: "demo.fiveStar.bindChangeListener",
            args: "{that}"
        }, {
            "this": "{that}.stars",
            method: "mouseover",
            args: {
                expander: {
                    funcName: "demo.fiveStar.makeStarHandler",
                    args: ["{that}", "{that}.hoverStars"]
                }
            }
        }, {
            "this": "{that}.container",
            method: "mouseout",
            args: "{that}.refreshView"
        }, {
            "this": "{that}.stars",
            method: "click",
            args: {
                expander: {
                    funcName: "demo.fiveStar.makeStarHandler",
                    args: ["{that}", "{that}.setRank"]
                }
            }
        }]
    },
```

The first thing to note is that this component attaches an array of listeners to a single event, the `onCreate` event:

```
    listeners: {
        onCreate: [...]
    },
```

The `onCreate` event is one of the component lifecycle events defined automatically for every component. It fires after the component has been fully created. When it fires, each of the listener functions in the array will be called.

MORE TO COME...

Invokers

STILL TO COME...