Members

This functionality is Sneak Peek status. This means that the **APIs may change**. We welcome your feedback, ideas, and code, but please use caution if you use this new functionality.

On This Page

Overview

The Infusion IoC system provides a mechanism for declaratively describing the public properties of a component. Members are typically used to store data values - in rare cases they may also hold functions, although these are best described using Invokers which allow for polymorphism. Often static values (which are expected to remain fixed over the component's lifetime) are stored and sourced from the components options directly, however it is good practice to make use of the members block if the values will be mutable or should be exposed at the top level. Mutable values which are expected to be observed or shared amongst multiple components should instead be stored in the component's model.

Declaring Members

Members can be assigned via literal values, IoC references, or the result of an expander.

Example:

The following example defines a component of type `xyz.widget`, with three members named `min, multiplier,` and initialValue. min is given a literal value of 0, multiplier sources its value through an IoC reference pointing to a value in the components options, and `initialValue` is given a value via an expander. The `original` and `multiply` invokers access these properties directly from the component `"{that}"`.

```
fluid.defaults("xyz.widget", {
    ...
    members: {
        min: 0,
        multiplier: "{that}.options.multiplier"
        initialValue: {
            expander: {
                funcName: "xyz.widget.getInitialValue",
                args: ["{that}.dom.input"]
            }
        },
        ...
    }
    multiplier: 2,
        invokers: {
        original: {
                        funcName: "fluid.identity",
            args: ["{that}.initialValue"]
        },
        multiply: {
                        funcName: "xyz.widget.product",
            args: ["{arguments}.0", "{that}.multiplier", "{that}.min"]
        }
    }
    ...
});

xyz.widget.getInitialValue = function (input) {
    return input.val();
};

xyz.widget.product = function (value, multiplier, min) {
    return Math.max(min, value * multiplier);
};
```