Invokers

On This Page

Overview

The Infusion IoC system provides a mechanism for creating the public functions (or "methods", in OO terminology) of a component. Invokers can bind free functions, IoC resolved functions, and "this" based functions to the component, and to the context of the component. Invokers allow the signature of the bound function to differ arbitrarily from the signature applied by the caller. As well as allowing the traditional OO facility for shifting the "this" (or "that") argument representing the component itself in and out of the argument list, this also allows for much more powerful reuse of existing functions where signature elements can be sourced freely from around the component tree and within the argument list.

Types of Invokers

Standard invoker binding to a function using `funcName`, `func`

An invoker can specified with either the **`funcName`** property to reference a free function by its global name (e.g. fluid.copy, console.log, etc.) or the **`func`** property to reference an existing function (perhaps another invoker) from elsewhere in the component tree.

Property	Description
`funcName` or **`func`**	**Required** type: `string` **`funcName`** - A string representing the name of a free function. **`func`** - A string representing an IoC reference to a function which is attached to the component tree
`args`	**Optional** type: `array` An array of arguments to be passed into the function specified by the `func` or `funcName` property. The values in the array can be of any type, including IoC references. In this context, you may also use references of the form `{arguments}.n`, where n is an integer indexing from 0 which refers to the position of the originating argument, to use arguments passed in when the invoker is called. If no `args` are specified, all of the arguments passed into the invoker are sent to the underlying function unchanged.
`dynamic`	**Optional** type: `boolean` By default, the values specified in the `args` property are cached, with the exception of those defined by `{arguments}.n`. If any of these arguments reference a value that may change between invocations of the invoker, this flag should be set to `true`. Note that using this option will reduce performance.

An invoker record in context

The following skeleton example defines an invoker named invokerName attached to a component of type component.name. When a component of the type is instantiated, for example with a line such as `var that = component.name()`, the invoker will then be available as a function directly attached to the instance, callable under the name `invokerName` - e.g. as `that.invokerName(....args....)`

```
fluid.defaults("component.name", {
    ...
    invokers: {
        invokerName: {
            funcName: <fully namespaced string name of function>,
            args: <array of arguments>,
            dynamic: <boolean>
        },
        ...
    }
    ...
});
```

Example:

The following example defines a component of type `xyz.widget`, with two invokers named `addTwo` and `subtractTwo` - the former binds onto a free function named `xyz.widget.add`, the latter binds onto another invoker elsewhere in the tree, which is not shown. Note that there is no necessary relationship between the name of a component and the name of a free function which is bound to it. If there is a close relationship between the purpose of a function and a related component, it is a helpful convention to prefix its name with the component name, as here - however, any free function may be bound via its global name to any component.

```
fluid.defaults("xyz.widget", {
    ...
    invokers: {
        addTwo: {
            funcName: "xyz.widget.add",
            args: [2, "{arguments}.0"]
        },
        subtractTwo: {
            func: "{parent}.subtract",
            args: ["{arguments}.0", 2],
        }
    }
    ...
});

xyz.widget.add = function (a, b) {return a + b;};
```

Compact Format

Alternatively, invokers can be specified in a compact single line format. However, arguments specified in the invoker can only be strings or IoC References. Strings which can be converted into Numbers or Booleans will be so converted before being interpreted. Dynamic invokers are specified with an "!" before the arguments (equivalent to the `dynamic: true` annotation in the full syntax)

```
fluid.defaults("component.name", {
    ...
    invokers: {
        invokerName: "<fully namespaced string name of function>(<comma-separated ioc references>)",
        dynamicInvokerName: "<fully namespaced string name of function>!(<comma-separated ioc references>)",
        },
        ...
    }
    ...
});
```

Example:

```
fluid.defaults("xyz.widget", {
    ...
    invokers: {
        // regular invokers:
        addVal: "xyz.widget.add({that}.staticVal, {arguments}.0)",
        subtractVal: "{parent}.subtract({arguments}.0, {that}.staticVal)",
        // dynamic invokers:
        addMax: "xyz.widget.add!({that}.dynamicMaxVal, {arguments}.0)",
        subtractMax: "{parent}.subtract!({arguments}.0, {that}.dynamicMaxVal)"
    }
    ...
});

xyz.widget.add = function (a, b) {return a + b;};
```

Example

```
fluid.defaults("fluid.uploader.fileQueue", {
    ...
    invokers: {
        start: {
            funcName: "fluid.uploader.fileQueue.start",
            args: "{that}"
        },
        startFile: {
            funcName: "fluid.uploader.fileQueue.startFile",
            args: "{that}.currentBatch",
            dynamic: true
        },
        ...
    },
    ...
}
```

Invoker triggering a model change to some component's model

If the invoker's record contains the field `changePath` (rather than the standard `func/funcName`) this is recognised as a special type of invoker triggering a change to a particular component's model via its ChangeApplier. This type of record is documented in its own section with the ChangeApplier API. The compact syntax may not be used with this variety of record.

"this"-ist invoker binding to a OO-style JavaScript function referencing "this"

Specifying an invoker with a **`"this"`** property allows the invocation of functions whose body makes a reference to the special JavaScript value `"this"`. These are generally functions external to the Infusion framework, since it is a Fluid community standard to write "that"-ist functions whose execution is independent of the calling context. These can be any functions, but will most often be used for jQuery methods. See Declarative this-ism in IoC for more details. Note that the string `this` must always be quoted when appearing as a key as it is a JavaScript keyword.

property	Description
`"this"`	**Required** type: `object` The object used referenced by "this" internally in the function. For a jQuery method, this is usually an jQuery element.
`method`	**Required** type: `string` The name of the "thisist" function (attached to the `this` object designated above) to call.

`args`	**Optional** type: `array` An array of arguments to be passed into the function specified by the `func` or `funcName` property. The values in the array can be of any type, including IoC references. In this context, you may also use references of the form `{arguments}.n`, where `n` is an integer indexing from 0 which refers to the position of the originating argument, to use arguments passed in when the invoker is called. If no `args` are specified, all of the arguments passed into the invoker are sent to the underlying function unchanged. Identical to role in `func/funcName` style invokers.

Format

```
fluid.defaults("component.name", {
    ...
    invokers: {
        invokerName: {
            "this": <reference to an object>,
            method: <string name of method on the object>,
            args: <array of arguments>
        }
    }
    ...
});
```

Example:

```
fluid.defaults("xyz.widget", {
    ...
    invokers: {
        setAriaLabel: {
            "this": "{that}.dom.elm",
            method: "attr",
            args: ["aria-label", "{arguments}.0"]
        }
    }
    ...
});
```

Example

```
fluid.defaults("fluid.uploader.html5Strategy.browseButtonView", {
    ...
    invokers: {
        ...
        disable: {
            "this": "{that}.dom.fileInputs",
            method: "prop",
            args: ["disabled", true]
        }
        ...
    },
    ...
}
```

Compact format

"this"-ist invokers have no equivalent compact syntax.