Invokers

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

Invokers: Dependency Resolution At Invocation

The Infusion IoC system provides a mechanism for creating public component functions whose arguments are resolved from the environment at invocation time. This differs from subcomponent, whose dependencies are resolved a single time, at creation. With invokers, the dependencies are "re"-resolved each time the function is invoked.

Declaring Invokers

Defaults

Invokers can declared similarly to subcomponents: Using a default/option called `invokers`:

```
fluid.defaults("component.name", {
    invokers: {
        invokerName1: {
            funcName: "name.of.implementation.function1",
            args: [...]
        },
        invokerName2: {
            funcName: "name.of.implementation.function2",
            args: [...]
        }
        ...
    }
});
```

When dependencies are initialized using `fluid.initDependents`, the invoker names (`invokerName1` and `invokerName2` in this example) will be used to create functions that will be attached to the component `that` object. The functions will resolve the values specified in the arguments and pass them to the function named by `funcName`.

Demands for Invokers

As with subcomponents, demands can be registered for invokers using `fluid.demands`. In this case, the declaration in the defaults is slightly different:

```
fluid.defaults("component.name", {
    invokers: {
        invokerName1: "demandingName"
        ...
    }
});
fluid.demands("demandingName", "component.name", {
    funcName: "name.of.implementation.function1",
    args: [...]
});
```

The form of the demands specification is the same for invokers as for subcomponents.

No arguments

If the implementing function requires no arguments, it can be declared directly in the defaults:

```
fluid.defaults("component.name", {
    invokers: {
        invokerName1: "implementing.function"
    }
});
```

This is essentially shorthand for:

```
fluid.defaults("component.name", {
    invokers: {
        invokerName1: {
            funcName: "implementing.function"
        }
    }
});
```

Argument to Invokers

The arguments list specified in an invoker declaration can include:

- references to the parent component, which will be resolved at invocation time;
- references to the parameters to the invoker, in the format `"@i"` where `i` is the index of the parameter.

Examples of Arguments to Invokers

```
invokers: {
    myFunc: {
        funcName: "nspace.myFuncImpl",
        args: ["@0", {parent}.field, "@1"]
    }
}
```

In this example, if

```
that.field = "CATT";
```

and the invoker is called with

```
that.myFunc("foo", "bar");
```

then the invoker will call the implementation function with

```
nspace.myFuncImpl("foo, "CATT", "bar);
```

Examples

```
fluid.defaults
("cspace.
autocomplete.popup",
{
    invokers: {
        treeBuilder:
{

funcName: "cspace.
autocomplete.
modelToTree",
            args:
["{popup}.model",
"{popup}.events"]
        }
    },
    ...
};
```

In this example, the popup component's `that` object will have a function called `treeBuilder`, i.e. `that.treeBuilder()`. This function, when called, will evaluate the current {`popup` and `events` properties of the popup component and pass them to the function `cspace.autocomplete.modelToTree()`.

```
fluid.defaults("test.
testComponent", {
    template: "Every
{0} has {1} {2}(s)",
    invokers: {
        render: {
            funcName:
"fluid.formatMessage",
            args:
["{testComponent}.
options.template",
"@0"]
        }
    },
    ...
```

In this example, the component's `that` object will have a function called `render`, i.e. `that.render()`. This function, when called, will evaluate the current {`options.template` property of the test component and pass it and the first parameter to `that.render()` to the function `fluid.formatMessage()`.

The following two methods of declaring an invoker are equivalent:

```
fluid.defaults("fluid.testUtils.invokerComponent", {
    template: "Every {0} has {1} {2}(s)",
    invokers: {
        render: {
            funcName: "fluid.formatMessage",
            args:["{invokerComponent}.options.template", "@0"]
        }
    }
});
```

In this version, the invoker `render` is declared directly, in the `invokers:` block.

```
fluid.defaults("fluid.testUtils.invokerComponent", {
    template: "Every {0} has {1} {2}(s)",
    invokers: {
        render: "stringRenderer"
    }
});
fluid.demands("stringRenderer", "fluid.testUtils.invokerComponent", {
    funcName: "fluid.formatMessage",
    args:["{invokerComponent}.options.template", "@0"]
});
```

In this version, the invoker `render` is declared through a demands block registered to the demanding name `stringRenderer`.