Component Grades

On This Page

Overview

A component **grade** extends the notion of component defaults (fluid.defaults). In fact, every **fluid.defaults** directive introduces a **grade** into the system of Fluid Infusion components, which can be built on to derive further grades/components. This derivation occurs by mentioning the name of the original grade within the **gradeNames** section of the derived component.

Developers can create their own **fluid.defaults / grades** as well as use them to build upon each other, and compose them as needed.

The Infusion Framework already contains several predefined component grades that normally form the initial building blocks for external components and grades. The following table describes these grades and how they relate to each other.

Grade Name	"Relay" Version [*]	Description
`autoInit`		A special directive grade that instructs the framework to automatically construct a globally named creator function (with the same name as the grade) responsible for the construction of the component. NOTE: for the Infusion 2.0 release this grade will become redundant as it will be the default for every grade
`fluid.littleComponent`		A "little" component is the most basic component: it supports options merging with defaults (Little Components). All Fluid components are derived from this grade, and in general all things not derived from this grade are non-components (e.g. plain functions, or model transformation transforms, etc.)
`fluid.modelComponent`	`fluid.modelRelayComponent`	A "model" component is already a little component that additionally provides supports for a component's model, defined in the components options, and operations on it (Tutorial - Model Components).
`fluid.eventedComponent`		An "evented" component is already a little component that additionally instantiates event firers based on default framework events (onCreate, onDestroy, onDetach) and events declared in the options (Tutorial - Evented Components).
`fluid.standardComponent`	`fluid.standardRelayComponent`	A compound of `fluid.modelComponent` and `fluid.eventedComponent`
`fluid.viewComponent`	`fluid.viewRelayComponent`	A "view" component is a fluid.standardComponent is bound to a DOM container node, holds a DOM Binder and supports a view (Tutorial - View Components).
`fluid.rendererComponent`	`fluid.rendererRelayComponent`	A "renderer" component is already a vew component that bears a renderer. There are additional features provided by this component grade specified on the Useful functions and events section of the Tutorial - Renderer Components page

[*] About the special "relay" grades - as part of the work on the New ChangeApplier coming up to the 1.5 release of Infusion, every standard grade descended from fluid.modelComponent has acquired a parallel version including the word "relay" that allows access to the new ChangeApplier on an "opt in basis". During the course of 2014 we will be incrementally updating each Infusion component to the new "relay" grades, and once this work is complete, the "relay" grades will be renamed back to their standard (left column) names and the old ChangeApplier implementation will be abolished.

Specifying Parent Grades

A component's grades should be specified using the `gradeNames` option in the components defaults block, as shown in the examples below. The `gradeNames` option holds a string or Array of Strings.

NOTE: In the examples below, the `autoInit` flag is not actually a grade, but is added to the `gradeNames` array to control how the component is created. See #Initializing Graded Components below for more information about the `autoInit` flag. The `autoInit` flag will soon become the default. *Always* use the `"autoInit"` flag, unless you have a very good reason not to.

```
fluid.defaults("fluid.uploader.demoRemote", {
    gradeNames: ["fluid.eventedComponent", "autoInit"],
    ...
});
```

```
fluid.defaults("cspace.messageBarImpl", {
    gradeNames: ["fluid.rendererComponent", "autoInit"],
    ...
});
```

```
fluid.defaults("cspace.util.relationResolver", {
    gradeNames: ["fluid.modelComponent", "autoInit"],
    ...
});
```

Initializing Graded Components

The Framework offers support for automatic initialization of graded component through `autoInit`. When the `autoInit` flag is added to the `gradeNames` array, the Framework will create the creator function automatically – the developer does not need to write a creator function.

To use the `autoInit` flag, add it to the array of `gradeNames`, as shown below:

```
fluid.defaults("fluid.uploader.fileQueueView", {
    gradeNames: ["fluid.viewComponent", "autoInit"],
    ...
});


var that = fluid.uploader.fileQueueView( ... ); // The framework has automatically generated this function
since the component is autoInit
```

NOTE: The `"autoInit"` flag should always be used if you expect the grade to be directly instantiated as a component. It can be omitted if the only use of the grade is as an "add-on" ("mixin") to another grade hierarchy.

Combining Grades

Since the `fluid.defaults` directive introduces a grade into the system, various components can be composed to create new ones. Options, fields and methods introduced by the ancestor grades will be merged. The merging happens, firstly in hierarchical order (grades comprising the ancestor grade are resolved before the actual component grades resolution) and secondly in the left-to-right order (defaults from the grade on the right taking precedence over the defaults from the grade on the left, more details can be found at the JIRA FLUID-5085). For example:

```
fluid.defaults("examples.componentOne", {
    gradeNames: ["fluid.modelComponent", "autoInit"],
    model: {
                field1: true
        },
        option1: "TEST"
});



fluid.defaults("examples.componentTwo", {
    gradeNames: ["fluid.modelComponent", "autoInit"],
    model: {
                field1: false,
                field2: true
        },
        option2: "TEST2"
});

fluid.defaults("examples.combinedComponent", {
    gradeNames: ["examples.componentOne", "examples.componentTwo", "autoInit"]
    // The resulting defaults for component examples.combinedComponent will look like this:
    // model: {
    //     field1: false,
    //     field2: true
    // },
    // option1: "TEST",
    // option2: "TEST2"
});
```

NOTE: All the material from the component defaults will be merged by the framework, including records such as `events`, `listeners`, `members`, `components`, `invokers` and `model`. Some of these, e.g. `listeners` will receive custom merging algorithms sensitive to their context - for example showing awareness of listener namespaces.

Dynamic Grades

Grades supplied as arguments to a constructing component in the `gradeNames` field will be added into the grade list of the particular component instance, as if a new `fluid.defaults` block had been issued creating a new "type" in the system - however, the main `type` of the component will not change. This facility could be thought of as a form of "type evolution" or Schema evolution. All dynamic grades take precedence over (that is, are merged in after) all static grades.

Delivering a dynamic gradeName as a direct argument:

There are numerous ways that these additional gradeNames could be delivered - for example, as a direct argument to a component's creator function:

```
var myCombinedComponent = examples.componentOne({
    gradeNames: "examples.componentTwo"
    }); // creates a component that behaves exactly (except for its typeName)
        // as if it was created via examples.combinedComponent() above
```

Delivering a dynamic gradeName via a subcomponent record:

Another possibility is to supply the additional gradeNames via a subcomponent record - for example

```
fluid.defaults("examples.rootComponent", {
    components: {
        myCombinedComponent: { // This component also behaves (except for typeName)
                               // as if was created via examples.combinedComponent
            type: "examples.componentOne",
            options: {
                gradeNames: "examples.componentTwo"
            }
        }
    }
});
```

Delivering a dynamic gradeName via an options distribution:

Perhaps one of the most powerful possibilities is to distribute dynamic gradeNames to one or more components via a distributeOptions record:

```
fluid.defaults("examples.distributingRootComponent", {
    distributeOptions: {
        record: "examples.componentTwo",
        target: "{that examples.componentOne}.options.gradeNames}"
    },
    components: {
        myCombinedComponent1: {
            type: "examples.componentOne",
        },
        myCombinedComponent2: {
            type: "examples.componentOne",
        }
    }
});
```

In the above example, **every** subcomponent of `examples.distributingRootComponent` which had a grade content of `examples.componentOne` would automatically have mixed in a grade of `examples.componentTwo`, causing them all to behave as if they were instances of `examples.combinedComponent`

Raw Dynamic Grades

Another very powerful framework facility is the use of *raw dynamic grades*. In this scheme, the gradeNames list for any component may include any standard IoC reference which may resolve to either a String or Array of Strings directly holding one or more grade names, or else a zero-arg function which can be invoked to obtain such a value. In this way, the developer can specify additional grade names based on dynamic material (potentially not known at the time of definition) such as a function (method or invoker) or a property in component options. Note that use of this facility should be discouraged in favour of any of the other techniques on this page - e.g. standard dynamic grades or grade linkage - in future versions of the framework the use of raw dynamic grades may impose a big performance penalty.

For example:

```
fluid.defaults("fluid.componentWithDynamicGrade", {
    gradeNames: ["fluid.littleComponent", "autoInit", "{that}.getDynamicGradeName"],
    invokers: {
        getDynamicGradeName: "fluid.componentWithDynamicGrade.getDynamicGradeName"
    }
});

// When resolved our fluid.componentWithDynamicGrade will have all the functionality of a fluid.modelComponent
grade.
// NOTE: developers can also return an array of grade names. These grade names can be custom grade names.
fluid.componentWithDynamicGrade.getDynamicGradeName = function () {
    return "fluid.modelComponent";
};
```

Grade Linkage

A powerful scheme for producing components whose grade content depends on combinations of other grades, or else to "advise" an already existing grade to append further grades into its components without redefining it, is ***grade linkage***. In the current framework this is an experimental and extremely expensive (in CPU) facility which is only available on an "opt-in" basis, although it will be optimised and refined in future versions of the framework. The original implementation and the need for it are described in the JIRA FLUID-5212. This rarely-required facility covers the part of the framework's responsibilities that used to be covered by the now withdrawn demands blocks facility which are not covered by any other scheme. These mostly relate to situations where some form of multiple dispatch is required.

A component opts in to the grade linkage system by including the framework grade `fluid.applyGradeLinkage`. This grade may itself be supplied dynamically by any of the schemes listed earlier in this page. Whenever an instance of a component which has opted in starts to construct, the complete list of all of its grades, both static and dynamic, are checked against **all** grade linkage records which have been registered into the system holding the grade `fluid.gradeLinkageRecord`. Each of these linkage records will have the grades that they list in the field `contextGrades` checked against the component's total set of grades - if ***every one*** of them appears in the component, then all of the grades mentioned in the record's `resultGrades` entry will be added into the constructing component's set of grades. The dynamic grade resolution process will then kick off again until no further grades arrive.

The following example demonstrates the process from end to end:

```
// Sets up a component which opts into the grade linkage system and has a parent grade of "examples.
componentOne"
fluid.defaults("examples.gradeLinkageComponent", {
    gradeNames: ["examples.componentOne", "fluid.applyGradeLinkage", "autoInit"],
});

// Sets up a "grade linkage record" that states that any opted-in component which comes to
// have both "examples.componentOne" AND "examples.componentTwo" as grades through any route,
// will automatically be also given grade "examples.componentOneAndTwo"
fluid.defaults("examples.oneTwoLinkage", {
    gradeNames: ["fluid.gradeLinkageRecord", "autoInit"],
    contextGrades: ["examples.componentOne", "examples.componentTwo"],
    resultGrades: "examples.componentOneAndTwo"
});

// Construct an instance of our component supplying an additional dynamic grade of "examples.componentTwo" -
// this will activate the grade linkage system and automatically supply the further grade of "examples.
componentOneAndTwo"
var that = examples.gradeLinkageComponent({
    gradeNames: "examples.componentTwo"
});
fluid.hasGrade(that.options, "examples.componentOneAndTwo"); // true
```

Once the implementation for this feature has bedded down, the framework grade `fluid.applyGradeLinkage` will be removed and all components will be opted-in by default.