Contexts and Demands

Previous: Subcomponents Up to overview **Next:** Loading Templates and Other Resources

On This Page

See Also

`fluid.demands`
Contexts
Demands Specifications
Demand Resolution

Contexts

Subcomponents may have different requirements depending on the *context* in which they are operating. For example, a subcomponent might operate differently when running on a production server versus when testing locally off the file system, and differently still when operating in the context of automated tests.

You can find more detailed information on contexts by following the "See also" links in the right sidebar, but in general, the *context* is made up of

- the various other components that have already been instantiated (e.g. ancestor components, siblings, etc), and
- any special "name-only" components that have been defined (e.g. a 'flag' indicating a test environment).

When subcomponents do have different requirements, these can be declared using a *demands block*. This is a call to the Framework function `fluid.demands()` specifying:

- the name of the demanding component,
- the name(s) of the components in the specific context, and
- the particular demands for that context.

```
fluid.demands("tutorials.subcomponent", ["tutorials.parent1", ...], {
    ...
});
```

A demands block is a way of saying:

> *"When X is being used in the context of Y, use this special configuration."*

Demands for Component Options

The default values for a component's options may be different in different contexts. Demands blocks can be used to provide different values for these options.

Example: Test vs. Production

Consider the example of a data source component responsible for fetching data from a server. One of the options for this component would likely be the URL of the server. The **defaults** for the data source might include the following:

```
fluid.defaults("tutorials.dataSource", {
    gradeNames: ["fluid.littleComponent", "autoInit"],
    baseUrl: "http://some.server.com/data/",
    ...
});
```

During development, you would not want to query the server each time you run tests – at the very least, you wouldn't want to query the production server! Demands blocks allow you to specify a different base URL for the test context, without having to to write complicated `if/then` blocks.

In your unit test files, register a special test-only mini-component using the `fluid.registerNamespace` function. This namespace will then exist when your tests run, but not in the production environment.

```
// declare the test environment
fluid.registerNamespace("tutorials.testEnvironment");

// when creating a dataSource in the context of the test environment,
// override the default base URL
fluid.demands("tutorials.dataSource", ["tutorials.testEnvironment"], {
    options: {
        baseUrl: "../testdata/"
    }
});
```

Demands for Component Implementations

Demands also allow you to specify a completely different implementation for a given subcomponent in different contexts:

> *"When someone asks for X in the context of Y, use function Z"*
> *"When someone asks for X in the context of P, use function Q".*

Example: UI Options Preview Component

In our discussion of Subcomponents, we looked at the example of the User Interface Options (UI Options). One of UI Options' subcomponents was a Preview component. This preview is different in each of the three different versions of UI Options. These different configurations can be created using demands blocks:

```
// when instantiating the preview subcomponent in the context of the Fat Panel UI Options,
// Use the 'livePreview' function
fluid.demands("preview", ["fluid.fatPanelUIOptions"], {
    type: "fluid.uiOptions.livePreview"
});

// when instantiating the preview subcomponent in the context of the Full With Preview UI Options,
// Use the 'preview' function
fluid.demands("preview", ["fluid.fullPreviewUIOptions"], {
    type: "fluid.uiOptions.preview"
});

// when instantiating the preview subcomponent in the context of the Full, no Preview UI Options,
// disable preview by using the 'emptySubcomponent'
fluid.demands("preview", ["fluid.fullNoPreviewUIOptions"], {
    type: "fluid.emptySubcomponent"
});
```

Previous: Subcomponents Up to overview **Next:** Loading Templates and Other Resources