Define a namespace and create a closure

Previous: Set up your environment Up to overview **Next:** Pick a component type

Infusion code generally follows a few conventions that we recommend, and that we'll use in this tutorial. We'll start with two of them:

1. namespacing, and
2. closures

Namespacing
We define a namespace for our code: a single global variable that becomes the container for the code. This helps keep code contained, meaning there's less chance of bad interactions with other code: Anything we want to be public will be attached to this object, so all of our code will be qualified by the namespace.

Closures
By wrapping the rest of the code inside an anonymous function, we can separate private and public functions. Only objects or functions that are attached to the global namespace object will be publicly available. Anything else inside the anonymous function will be invisible to the rest of the world. In general we recommend against the use of private definitions since they inhibit other developers from getting value from your code. Write every function and piece of data as a public member of your namespace, with a suitable comment if you don't intend them to form a stable part of your API. Of course, make sure not to write any mutable shared state in public - in general, you should make sure any mutable state is packaged as part of a Fluid Component.

General Structure

So what does this look like in general?

```
(function ($, fluid) {
    fluid.registerNamespace("mynamespace");

    // a private function, only accessible to other things
    // inside this closure - this is discouraged
    var privateFunc = function () {
        ...
    };

    // a public function, attached to the namespace - this is recommended
    mynamespace.publicFunc = function () {
        ...
    };

})(jQuery, fluid_1_5);
```

You might like to think of the `fluid.registerNamespace` call as equivalent to a line such as `var mynamespace = mynamespace || {};` written at the global scope. It is less cumbersome and more expressive of your intention, as well as easily allowing you to declare nested namespaces in one definition. Use this framework utility unless your requirements are very sophisticated (that is, you are writing a framework which you expect to be version-managed independently of both jQuery and fluid - in which case you should supply your framework's global object as an argument to the overall file closure.

The parameters to the anonymous function, `$` and `fluid`, will be used as shorthand for the arguments that were passed in: `jQuery` and `fluid_1_5` resp ectively. This allow us, for example, to upgrade to the next version of Infusion (e.g. `fluid_1_5`) simply by changing the one argument, instead of having to change every single use of the word `fluid`.

Example

So what might this look like in the real world? Well, for our bar graph example, we might call the global namespace `visualizer` and create a public function called `barGraph` that can be used by anyone to instantiate a bar graph component:

```
(function ($, fluid) {
    fluid.registerNamespace("visualizer");
    // put any private things the bar graph will need here


    // EITHER: the public bar graph creation function - this is discouraged
    visualizer.barGraph = function () {
        ...
    };
    // OR: a creator function automatically managed by Infusion - this is recommended
    fluid.defaults("visualizer.barGraph", {
        ...
    });

})(jQuery, fluid_1_5);
```

Previous: Set up your environment Up to overview **Next:** Pick a component type