How to Define a Unit

What is a Unit?

In Fluid, a *unit* is a bundle of data and functions representing a "type" or "class" of object in JavaScript. Units are defined using plain old objects and functions. JavaScript has a very loose, dynamic type system where objects can be modified and augmented freely at run time. The language doesn't have the concept of classes, and many familiar attempts to impose a classical system on the language have proved awkward and unworkable. Units address this by embracing a simple and functional approach to packaging up objects.

On this Page

In idiomatic JavaScript, inheritance is prototypal. An object can inherit from another object simply by creating a new instance and modifying its functionality. No classes, no static types. Used carefully, this is a good thing.

Lots of things can be defined as a unit. All Fluid components are themselves units, but are also composed of other smaller, more specialized units. These are all just plain old JavaScript objects, wrapped in functions for privacy.

Is this another attempt at creating a class system?

No. Units are defined using a functional idiom. This isn't an attempt to impose a class-based system on JavaScript, but rather a convention for defining formal objects in a simple way. And in a way that avoids bugs in the language.

Here we are putting the emphasis on the aspect of "classes" as things we can create multiple, similar instances of, rather than the other aspects relating to being able to categorise these units as we find them in the wild. In a dynamic language, categorisation should be done using duck typing instead of type checking.

The standard JavaScript approach to defining types using constructor functions, prototypes, and the `new` keyword can be confusing and error-prone. For example, if you forget to invoke a constructor function using the `new` keyword, the result is a silent failure. No error, and nothing works. Just confusion. Worse yet, using `.prototype` allows other code to dynamically modify types, breaking the contract your code depends on. This has the potential to cause all kinds of hard-to-debug errors. Avoiding these features of JavaScript will make your life easier.

Fluid's object definition idiom ("that-ism") is designed to remove the confusing and buggy aspects of JavaScript and replace them with simple *creator functions* that return regular objects. Plain old functions provide a private scope within which types can be defined, and they don't require the use of the `new` keyword. Once you get used to the syntax, the result is a simple and straightforward way to define new sorts of objects.

This idiom was originally defined by Douglas Crockford, and was derived from Chapter 5 of his *JavaScript: The Good Parts* book.

How do I define a Unit in code?

Here's a simple template you can use when defining your own units:

```
// First, define your own unique namespace.
var fluid = fluid || {};

// Next, create an anonymous function for privacy. This closure represents a bundle of related code.
// Pass in any common dependencies, such as the jQuery "$" shorthand, as arguments to avoid conflicts.
// Also pass in the fluid variable, to allow versioning of this namespace
(function ($, fluid) {

    // Inside the closure, define any private, shared utility functions.
    var usefulPrivateFunction = function (args) { ... };


    // Then, define a creator function for your object in your public namespace.
    fluid.tabs = function(args) {
        // Create your new object instance.
        // This could be done using object literals, or by calling another creator or constructor function.
        var that = {};

        // Define any private methods here.
        var myPrivateMethod = function () {
          doSomethingPrivate(that);
        };

        // Then define any public instance methods.
        that.select = function(tabToSelect) {
            that.selectedTab = tabToSelect;
        };

        // Finally, return your new object instance to the world.
        return that;
    };

// Invoke your private closure now, with the appropriate context arguments
})(jQuery, fluid);
```