**Writing Stateful Plugins with the jQuery UI Widget Factory: Part 1**

**Note**

This section is based, with permission, on the blog post [Building Stateful jQuery Plugins](http://blog.nemikor.com/2010/05/15/building-stateful-jquery-plugins/) by Scott Gonzalez.

While most existing jQuery plugins are stateless — that is, we call them on an element and that is the extent of our interaction with the plugin — there’s a large set of functionality that doesn’t fit into the basic plugin pattern.

In order to fill this gap, jQuery UI has implemented a more advanced plugin system. The new system manages state, allows multiple functions to be exposed via a single plugin, and provides various extension points. This system is called the widget factory and is exposed as jQuery.widget as part of jQuery UI 1.8; however, it can be used independently of jQuery UI.

To demonstrate the capabilities of the widget factory, we'll build a simple progress bar plugin.

To start, we’ll create a progress bar that just lets us set the progress once. As we can see below, this is done by calling jQuery.widget with two parameters: the name of the plugin to create and an object literal containing functions to support our plugin. When our plugin gets called, it will create a new plugin instance and all functions will be executed within the context of that instance. This is different from a standard jQuery plugin in two important ways. First, the context is an object, not a DOM element. Second, the context is always a single object, never a collection.

**Example 8.3. A simple, stateful plugin using the jQuery UI widget factory**

$.widget("nmk.progressbar", {

\_create: function() {

var progress = this.options.value + "%";

this.element

.addClass("progressbar")

.text(progress);

}

});

The name of the plugin must contain a namespace; in this case we’ve used the nmk namespace. There is a limitation that namespaces be exactly one level deep — that is, we can't use a namespace like nmk.foo. We can also see that the widget factory has provided two properties for us. this.element is a jQuery object containing exactly one element. If our plugin is called on a jQuery object containing multiple elements, a separate plugin instance will be created for each element, and each instance will have its own this.element. The second property, this.options, is a hash containing key/value pairs for all of our plugin’s options. These options can be passed to our plugin as shown here.

**Note**

In our example we use the nmk namespace. The ui namespace is reserved for official jQuery UI plugins. When building your own plugins, you should create your own namespace. This makes it clear where the plugin came from and whether it is part of a larger collection.

**Example 8.4. Passing options to a widget**

$("<div></div>")

.appendTo( "body" )

.progressbar({ value: 20 });

When we call jQuery.widget it extends jQuery by adding a method to jQuery.fn (the same way we'd create a standard plugin). The name of the function it adds is based on the name you pass to jQuery.widget, without the namespace; in our case it will create jQuery.fn.progressbar. The options passed to our plugin get set in this.options inside of our plugin instance. As shown below, we can specify default values for any of our options. When designing your API, you should figure out the most common use case for your plugin so that you can set appropriate default values and make all options truly optional.

**Example 8.5. Setting default options for a widget**

$.widget("nmk.progressbar", {

// default options

options: {

value: 0

},

\_create: function() {

var progress = this.options.value + "%";

this.element

.addClass( "progressbar" )

.text( progress );

}

});