**Writing Stateful Plugins with the jQuery UI Widget Factory: Part 2**

**Adding Methods to a Widget**

Now that we can initialize our progress bar, we’ll add the ability to perform actions by calling methods on our plugin instance. To define a plugin method, we just include the function in the object literal that we pass to jQuery.widget. We can also define “private” methods by prepending an underscore to the function name.

**Example 8.6. Creating widget methods**

$.widget("nmk.progressbar", {

options: {

value: 0

},

\_create: function() {

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

this.element

.addClass("progressbar")

.text(progress);

},

// create a public method

value: function(value) {

// no value passed, act as a getter

if (value === undefined) {

return this.options.value;

// value passed, act as a setter

} else {

this.options.value = this.\_constrain(value);

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

this.element.text(progress);

}

},

// create a private method

\_constrain: function(value) {

if (value > 100) {

value = 100;

}

if (value < 0) {

value = 0;

}

return value;

}

});

To call a method on a plugin instance, you pass the name of the method to the jQuery plugin. If you are calling a method that accepts parameters, you simply pass those parameters after the method name.

**Example 8.7. Calling methods on a plugin instance**

var bar = $("<div></div>")

.appendTo("body")

.progressbar({ value: 20 });

// get the current value

alert(bar.progressbar("value"));

// update the value

bar.progressbar("value", 50);

// get the current value again

alert(bar.progressbar("value"));

**Note**

Executing methods by passing the method name to the same jQuery function that was used to initialize the plugin may seem odd. This is done to prevent pollution of the jQuery namespace while maintaining the ability to chain method calls.

**Working with Widget Options**

One of the methods that is automatically available to our plugin is the option method. The option method allows you to get and set options after initialization. This method works exactly like jQuery’s css and attr methods: you can pass just a name to use it as a setter, a name and value to use it as a single setter, or a hash of name/value pairs to set multiple values. When used as a getter, the plugin will return the current value of the option that corresponds to the name that was passed in. When used as a setter, the plugin’s \_setOption method will be called for each option that is being set. We can specify a \_setOption method in our plugin to react to option changes.

**Example 8.8. Responding when an option is set**

$.widget("nmk.progressbar", {

options: {

value: 0

},

\_create: function() {

this.element.addClass("progressbar");

this.\_update();

},

\_setOption: function(key, value) {

this.options[key] = value;

this.\_update();

},

\_update: function() {

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

this.element.text(progress);

}

});