**Writing Stateful Plugins with the jQuery UI Widget Factory: Part 4**

**Cleaning Up**

In some cases, it will make sense to allow users to apply and then later unapply your plugin. You can accomplish this via the destroy method. Within the destroy method, you should undo anything your plugin may have done during initialization or later use. The destroy method is automatically called if the element that your plugin instance is tied to is removed from the DOM, so this can be used for garbage collection as well. The default destroy method removes the link between the DOM element and the plugin instance, so it’s important to call the base function from your plugin’s destroy method.

**Example 8.11. Adding a destroy method to a widget**

$.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);

if (this.options.value == 100 ) {

this.\_trigger("complete", null, { value: 100 });

}

},

destroy: function() {

this.element

.removeClass("progressbar")

.text("");

// call the base destroy function

$.Widget.prototype.destroy.call(this);

}

});

**Conclusion**

The widget factory is only one way of creating stateful plugins. There are a few different models that can be used and each have their own advantages and disadvantages. The widget factory solves lots of common problems for you and can greatly improve productivity, it also greatly improves code reuse, making it a great fit for jQuery UI as well as many other stateful plugins.