**Encapsulation: Part 1**

The first step to code organization is separating pieces of your application into distinct pieces; sometimes, even just this effort is sufficient to lend

**The Object Literal**

An object literal is perhaps the simplest way to encapsulate related code. It doesn't offer any privacy for properties or methods, but it's useful for eliminating anonymous functions from your code, centralizing configuration options, and easing the path to reuse and refactoring.

**Example 10.1. An object literal**

var myFeature = {

myProperty : 'hello',

myMethod : function() {

console.log(myFeature.myProperty);

},

init : function(settings) {

myFeature.settings = settings;

},

readSettings : function() {

console.log(myFeature.settings);

}

};

myFeature.myProperty; // 'hello'

myFeature.myMethod(); // logs 'hello'

myFeature.init({ foo : 'bar' });

myFeature.readSettings(); // logs { foo : 'bar' }

The object literal above is simply an object assigned to a variable. The object has one property and several methods. All of the properties and methods are public, so any part of your application can see the properties and call methods on the object. While there is an init method, there's nothing requiring that it be called before the object is functional.

How would we apply this pattern to jQuery code? Let's say that we had this code written in the traditional jQuery style:

// clicking on a list item loads some content

// using the list item's ID and hides content

// in sibling list items

$(document).ready(function() {

$('#myFeature li')

.append('<div/>')

.click(function() {

var $this = $(this);

var $div = $this.find('div');

$div.load('foo.php?item=' +

$this.attr('id'),

function() {

$div.show();

$this.siblings()

.find('div').hide();

}

);

});

});

If this were the extent of our application, leaving it as-is would be fine. On the other hand, if this was a piece of a larger application, we'd do well to keep this functionality separate from unrelated functionality. We might also want to move the URL out of the code and into a configuration area. Finally, we might want to break up the chain to make it easier to modify pieces of the functionality later.

**Example 10.2. Using an object literal for a jQuery feature**

var myFeature = {

init : function(settings) {

myFeature.config = {

$items : $('#myFeature li'),

$container : $('<div class="container"></div>'),

urlBase : '/foo.php?item='

};

// allow overriding the default config

$.extend(myFeature.config, settings);

myFeature.setup();

},

setup : function() {

myFeature.config.$items

.each(myFeature.createContainer)

.click(myFeature.showItem);

},

createContainer : function() {

var $i = $(this),

$c = myFeature.config.$container.clone()

.appendTo($i);

$i.data('container', $c);

},

buildUrl : function() {

return myFeature.config.urlBase +

myFeature.$currentItem.attr('id');

},

showItem : function() {

var myFeature.$currentItem = $(this);

myFeature.getContent(myFeature.showContent);

},

getContent : function(callback) {

var url = myFeature.buildUrl();

myFeature.$currentItem

.data('container').load(url, callback);

},

showContent : function() {

myFeature.$currentItem

.data('container').show();

myFeature.hideContent();

},

hideContent : function() {

myFeature.$currentItem.siblings()

.each(function() {

$(this).data('container').hide();

});

}

};

$(document).ready(myFeature.init);

The first thing you'll notice is that this approach is obviously far longer than the original — again, if this were the extent of our application, using an object literal would likely be overkill. Assuming it's not the extent of our application, though, we've gained several things:

* We've broken our feature up into tiny methods. In the future, if we want to change how content is shown, it's clear where to change it. In the original code, this step is much harder to locate.
* We've eliminated the use of anonymous functions.
* We've moved configuration options out of the body of the code and put them in a central location.
* We've eliminated the constraints of the chain, making the code easier to refactor, remix, and rearrange.

For non-trivial features, object literals are a clear improvement over a long stretch of code stuffed in a $(document).ready() block, as they get us thinking about the pieces of our functionality. However, they aren't a whole lot more advanced than simply having a bunch of function declarations inside of that $(document).ready() block.