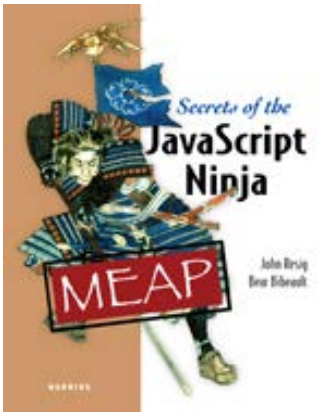
MANNING PUBLICATIONS



Secrets of the JavaScript Ninja
By John Resig and Bear Bibeault

Closures are one of the defining features of JavaScript. Without them, JavaScript would likely be another hum-drum scripting experience, but, since that's not the case, the landscape of the language is forever shaped by their inclusion. In this article based on chapter 4 of Secrets of the JavaScript Ninja, the authors explain how closures work.

How Closures Work

A closure is a way to access and manipulate external variables from within a function. A function is able to access all the variables and functions declared in the same scope as itself. The result is rather intuitive and is best explained through code, as in listing 1.

Listing 1 A few examples of closures

```
var outerValue = true;

function outerFn(arg1){
  var innerValue = true;

  assert( outerFn && outerValue, "These come from the closure." );
  assert( typeof otherValue === "undefined",
    "Variables defined late are not in the closure." );

  function innerFn(arg2){
    assert( outerFn && outerValue,
  "These still come from the closure." );
    assert( innerFn && innerValue && arg1,
  "All from a closure, as well." );
  }

  innerFn(true);
}

outerFn(true);

var otherValue = true;

assert( outerFn && outerValue,
  "Globally-accessible variables and functions." );
```

Note that listing 1 contains two closures: function `outerFn()` includes a closured reference to itself and the variable `outerValue`. Function `innerFn()` includes a closured reference to the variables `outerValue`, `innerValue`, and `arg1` and references to the functions `outerFn()` and `innerFn()`.

It's important to note that, while all of this reference information isn't immediately visible (there's no "closure" object holding all of this information, for example), there is a direct cost to storing and referencing your information in this manner. It's important to remember that each function that accesses information via a closure

immediately has a *ball and chain*, if you will, attached to them carrying this information. So, while closures are incredibly useful, they certainly aren't free of overhead.

Private variables

One of the most common uses of closures is to encapsulate some information as a *private variable* of sorts. Object-oriented code, written in JavaScript, is unable to have traditional private variables (properties of the object that are hidden from outside uses). However, using the concept of a closure, we can arrive at an acceptable result, as in listing 2.

Listing 2 An example of keeping a variable private but accessible via a closure

```
function Ninja(){
  var slices = 0;

  this.getSlices = function(){
    return slices;
  };
  this.slice = function(){
    slices++;
  };
}

var ninja = new Ninja();
ninja.slice();
assert( ninja.getSlices() == 1,
  "We're able to access the internal slice data." );
assert( ninja.slices === undefined,
  "And the private data is inaccessible to us." );
```

In listing 2, we create a variable to hold our state, `slices`. This variable is only accessible from within the `Ninja()` function (including the methods `getSlices()` and `slice()`). This means that we can maintain the state, within the function, without letting it be directly accessed by the user.

Callbacks and timers

Another one of the most beneficial places for using closures is when you're dealing with callbacks or timers. In both cases, a function is being called at a later time and within the function you have to deal with some, specific, outside data. Closures act as an intuitive way of accessing data, especially when you wish to avoid creating extra variables just to store that information. Let's look at a simple example of an Ajax request, using the jQuery JavaScript Library, in listing 3.

Listing 3 Using a closure from a callback in an Ajax request

```
<div></div>
<script src="jquery.js"></script>
<script>
var elem = jQuery("div");
elem.html("Loading...");

jQuery.ajax({
  url: "test.html",
  success: function(html){
    assert( elem, "The element to append to, via a closure." );
    elem.html( html );
  }
});
</script>
```

A couple of things are occurring in listing 3. To start, we're placing a loading message into the div to indicate that we're about to start an Ajax request. We have to do the query, for the div: once to edit its contents and would typically have to do it again to inject the contents when the Ajax request completed. However, we can make good use of a closure to save a reference to the original jQuery object (containing a reference to the div element), saving us from that effort.

Listing 4 has a slightly more complicated example, creating a simple animation.

Listing 4 Using a closure from a timer interval

```
<div id="box" style="position:absolute;">Box!</div>
<script>
var elem = document.getElementById("box");
var count = 0;

var timer = setInterval(function(){
  if ( count < 100 ) {
    elem.style.left = count + "px";
    count++;
  } else {
    assert( count == 100,
  "Count came via a closure, accessed each step." );
    assert( timer, "The timer reference is also via a closure." );
    clearInterval( timer );
  }
}, 10);
</script>
```

What's especially interesting about listing 4 is that it only uses a single (anonymous) function to accomplish the animation, and three variables, accessed via a closure. This structure is particularly important as the three variables (the DOM element, the pixel counter, and the timer reference) all must be maintained across steps of the animation. This example is a particularly good one in demonstrating how the concept of closures is capable of producing some surprisingly intuitive, and concise, code.

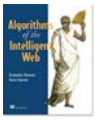
Summary

We saw how closures work in JavaScript. We looked at how they're implemented and then how to use them within an application. We looked at a number of cases where closures were particularly useful, including the definition of private variables and in the use of callbacks and timers.

Here are some other Manning titles you might be interested in:

Ext JS in Action
Jesus Garcia

jQuery in Action, Second Edition
Bear Bibeault and Yehuda Katz



Algorithms of the Intelligent Web
Haralambos Marmanis and Dmitry Babenko

Last updated: July 28, 2011