Computer Science E-75: Building Dynamic Websites
Harvard Extension School
Spring 2008

# Section 10: Frameworks (jQuery)

jQuery is a javascript library that is free to download and use, and it simplifies many of the things we already learned how to do in this course like traversing XHTML documents, performing animations, defining event handlers, and working with Ajax. We're going to cover some of the basics of jQuery in section notes this week. For those interested in CakePHP, please consult the lecture notes and slides for more information.

## GETTING STARTED

Let's download the library. Take your favorite web browser to http://jquery.com/ and click one of the links to download the library (any of the choices should be fine).

Now lets set up the example page that we'll be working with. Make it look something like this:

```
<html>
<head>
<script type="text/javascript" src="PATH_TO_JQUERY_LIB"></script>
<script type="text/javascript">
//<![CDATA[

// This is where the code we talk about goes!

//]]>
</script>
</head>
<body>
<a href="cs75.net">CS E-75 Homepage</a>
</body>
</html>
```

Granted, this is not proper XHTML (which all of our "real" pages should be), but it's fine for testing purposes. You will notice we're linking to the library within the first set of <script> tags. The path to your library goes in the value for the src attribute.

Now, we're ready to get rolling!

## DEFINING CODE TO RUN ON DOCUMENT READY

Javascript programmers (such as yourself!) should be with the following line of code:

```
window.onload = function(){ alert("hello world!") }
```

This defines a function to be run when the window loads. This function simply issues an alert that says hello world. Unfortunately, Javascript only runs after all images are loaded, so the alert will not be issued until after that time. Oddly, the reason why we use the onload method on the window is that the XHTML document isn't ready when we first want to run the code. But if this code doesn't run anyway until after the images are loaded, what good is it? Let's use a different statement, defined by jQuery as the ready event.

```
$(document).ready(function(){
   // Code to be executed
});
```

This code checks that the XHTML document is ready to be modified, and when it is ready, the function runs. We will put the code that follows in these section notes within that function definition (where "// Code to be executed" is).

**EVENT HANDLERS**
Now let's define a simple event handler. Remember, this goes within the code explained above!

```
$("a").click(function() { alert("You clicked me!") });      (1)
```

Let's take this line apart. The $ is an alias for the jQuery class. Thus, in essence, $() defines a new class. The whole unit, $("a"), is a jQuery selector which selects all a elements (like the <a href=".."> of a link). You can then call methods on that class like click, which takes as its argument a function to be called when the event triggers. There are many other event handler methods that can be applied, like change(), blur(), and focus(). Consult the API Reference for Events for more (http://docs.jquery.com/Events). The above code is similar to the following code, which you should recognize:

```
<a href="" onclick="alert('You clicked me!')">Link</a>      (2)
```

The difference between (1) and (2) is that (1) applies to all **a** tags, whereas (2) applies only to that specific instance of the <a> tag. In this way, we can separate Javascript and XHTML in the same way that we strive to separate CSS and XHTML.

**SELECTORS**
More complicated selectors are also available. For example,

```
$("#orderedlist")
```

will select tags with id="orderedlist". The dot operator can be used for classes, just as in CSS.

Virtually anything that you can do to traverse a DOM is also available to you with jQuery. X-path-like [expression] syntax is also available. For example,

```
$("a[@name=’link’]")
```

will select all <a> tags with a name attribute with “link” as its value.  For more complete documentation, check out the API Reference at http://docs.jquery.com/Selectors.

**ADDING A CLASS**
Adding a class to a tag is simple.  Just select the tag, and add the class using the addClass method with class’s name as the argument.  For example

```
$(“a”).addClass(“test”);
```

is virtually equivalent to putting a class=”test” in every <a> tag.  Now, to select all of these, you can just use

```
$(“a.test”)
```

**SPECIAL EFFECTS**
jQuery defines several methods that help you easily add “special effects” to your page.  Let’s take a look at the following code

```
1    $("a").click(function(event){
2      event.preventDefault();
3      $(this).hide("slow");
4    });
```

Line 1 selects the <a> tags on a page and defines a function to be called when they are clicked.  Line 2 prevents the default action of the click of an <a> tag (taking a user to the link specified by the href attribute within the <a> tag).  Line 3 uses the selector this to select the <a> tag itself, and then calls the hide special effect method.  “Slow” specifies the speed at which <a> tag should be hid.  There are different special effect methods (hide, show, etc.) and different speeds (“slow”, “normal”, etc.).  For more complete documentation, check out the API Reference at http://docs.jquery.com/Effects.

**CHAINABILITY**
Chainability is a wonderful feature of jQuery.  This feature stems from the quality that all methods within jQuery return the query object itself.  This means that you can “chain” methods together into long strings that read (relatively clearly) the way they will act.  Let’s take a look at the following code

```
1    $("a")
2      .filter(".clickme")
3        .click(function(){
4          alert("You are now leaving the site.");
5        })
6      .end()
7      .filter(".hideme")
```

```
8       .click(function(){
9         $(this).hide();
10        return false;
11      })
12    .end();
```

which works with the following XHTML:

```
<a href="http://google.com/" class="clickme">I give a message
when you leave</a>
<a href="http://yahoo.com/" class="hideme">Click me to hide!</a>
<a href="http://microsoft.com/">I'm a normal link</a>
```

A couple things to note.  Line 2 uses the method filter, which filters the previous selector to the class “clickme”, making whatever follows only apply to <a> tags with class=”clickme”.  Line 6 declares the end() method, which changes the set of matched elements to the previous state.  Basically, we go back to the $(“a”).  This allows us to define another event on a different class all within the original <a> tag selector (which we do in lines 7-12).  Check the API Reference on Traversing for more information. http://docs.jquery.com/Traversing

**CALLBACKS and AJAX**

jQuery does a nice job of covering AJAX, and streamlining many of the more annoying parts about using it. Let's look at a callback:

After including the jquery library...we have a form:

```
<input type="text" id="number1" value="0" /> +
<input type="text"id="number2" value="0" />
<input type="button" onclick="addNumbers();" value="=" />
```

Here's the callback:

```
function addNumbers() {
 var number1 = $('#number1').attr('value');
 var number2 = $('#number2').attr('value');
     $.get("giveMeSomething.php", {
          number1:number1,number2:number2 },
               function(data){
               alert("Data Loaded: " + data);
          });
     }
```

And "giveMeSomething.php" could be this...

```
<?
echo $_REQUEST['number1'] + $_REQUEST['number2'];
?>
```

Pretty self explanatory, right? For the rest of jQuery's AJAX functionality, I refer you to their (expansive) spec/API. Take a look!

http://docs.jquery.com/API/1.2/Ajax

Here are some highlights:

```
jQuery.post( url, data, callback, type )
```
Load a remote page using an HTTP POST request.

```
jQuery.getScript( url, callback )
```
Loads, and executes, a local JavaScript file using an HTTP GET request.

```
jQuery.get( url, data, callback )
```
Load a remote page using an HTTP GET request.

```
jQuery.ajaxSetup( options )
```
Setup global settings for AJAX requests.

As always, dive in and try these things out to REALLY learn how they work! Good luck with your projects!