INTRODUCING JQUERY

JavaScript is a scripting language developed by Netscape to allow basic dynamic operations to be performed in browsers. Originally created to be easily learned by web developers to add basic interactivity to an HTML page, JavaScript has since evolved into a tool most developers can't live without. However, because of various cross browser and legacy issues, many operations programmers would consider simple in environments such as app development are comparatively complex.

**Enter jQuery.**

jQuery offers the JavaScript developer more power and flexibility while simplifying and reducing the amount of code required. Started as a way to simplify object selection by John Resig back in 1995, jQuery has become an indispensable tool for any JavaScript developer.

jQuery is designed to offer cross browser support, allowing you to write your code once and have it run properly on every browser. jQuery makes it much easier to perform operations such as selecting individual or groups of items for manipulation, changing elements or the entire DOM in an HTML page, or making calls to the server to access remote resources.

## OPTIONS FOR ADDING JQUERY TO A PAGE

To add jQuery to a web page, it's as simple as adding a reference to the jQuery JavaScript file. Remember that you always want to use the minified version, signified by **min** in the name, as this will reduce the amount of data passed to the user, saving both bandwidth and loading time.

As we've already discussed, it's possible to download from jQuery, or to use a package manager such as NuGet or npm. Similarly, you have multiple options when it comes to sourcing the JavaScript file in your page.

The first option is to serve the file from your own server, just as you would serve any other image or other script file. Because you've already download the file, it's easy to simply ensure it's part of your project, deployed to your server, and then referenced in a page. However, this is yet another file that your server has to send to the user, which increases both the number of requests the server needs to handle and the amount of bandwidth required for the server to serve a page to a user.

As an alternative, you can use a content delivery network, or CDN. A CDN is a set of servers hosted by a third party, such as Microsoft or Google, that hosts and serves common files used for creating web pages. In fact, most every common JavaScript library, including Knockout, AngularJS, and Bootstrap, are available on CDNS.

By using a CDN you save your server from having to send down the file; the user's will download the file from the CDN servers rather than yours. For heavily trafficked sites this can mean a large improvement in performance.

When using a popular CDN, such as the one from Microsoft, there's a very good chance the user will have already downloaded the file. Browsers cache files based on path names, so if the user has visited a site that used the same file from the same CDN, the browser won't need to download that file again. As a result, your page will load that much faster.

## UPDATING YOUR PAGE TO INCLUDE A REFERENCE TO JQUERY

Below is the code you'll need to add jQuery to a page. Keep in mind that the URL of the JavaScript file will depend on where the file is located, and if you're using a CDN.

<html>

<head>

<title>Demo page</title>

</head>

<body>

<!-- your page -->

<script src="[location of minified jQuery JavaScript file]"></script>

<script>

// your custom scripts that use jQuery

</script>

</body>

</head>

THE MAGIC OF THE $

In jQuery, the $ character is the shortcut to the jQuery factory, meaning that it provides access to the jQuery object and all of the power that jQuery offers. You can use the $ to access global objects and functions, and, more commonly, elements and other components in the DOM.

The first operation you'll typically perform in any page that uses jQuery is to register a document.ready event handler. The document.ready event handler executes after everything on the page has loaded, ensuring that the entire DOM is now in memory, and the jQuery library is now available.

There are a couple of ways to register this event handler. The first is to access the document object in jQuery, and register an event handler with the jQuery ready event.

**$**(document).ready(function() {

// code here

});

Because registering an event handler with jQuery is so common, jQuery provides a shortcut. Simply passing a function into the constructor will register that function as the event handler for document.load.

**$**(function() {

// code here

});

Both of the code blocks above are semantically identical.

## ATTRIBUTE BASED SELECTORS

The HTML specification defines many attributes, such as class and style as well as allowing developers to add their own attributes. By using CSS selector syntax, you can locate items based on both custom and predefined attributes.

To find elements in which an attribute is set to a specific value, you use the equality syntax. Note that the value you wish to compare must be in quotes, which means you need to use both single and double quotes to create the appropriate string. While the actual quotes you use are up to you, I and most JavaScript developers I've worked with, use single quotes for normal strings in JavaScript, and then use double quotes when we need to indicate a quoted value inside of a string.

// selects \*\*all\*\* elements with an attribute matching the specified value

**$**('[demo-attribute="demo-value"]')

// selects all \*\*h1\*\* elements with an attribute matching the specified value

**$**('h1[demo-attribute="demo-value"]')

### Locating items by partial attribute values

Many frameworks, such as Bootstrap, make their magic happen by having developers add certain classes or other attributes to their HTML. Often, the values you'll use for classes or attributes have consistent patterns or prefixes. jQuery allows you to select items by searching inside of attribute values for desired sub-strings.

If you wish to find all elements where the value starts with a string, use the ^= operator.

**$**('[class^="col"]')

If you wish to find all elements where the value contains a string, use the \*= operator.

**$**('[class\*="md"]')

There are several more operators available. See [the documentation jQuery.com](http://api.jquery.com/category/selectors/attribute-selectors/) for a full list.

## POSITIONAL SELECTORS

Oftentimes you need to located elements based on where they are on the page, or in relation to other elements in the DOM. For example, an a element inside of a nav section may need to be treated differently than a elements elsewhere on the page. CSS, and in turn jQuery, offer the ability to find items based on their location.

### Parent/child relationships

The simplest location selector is one for parent/child. The > between selectors indicates the parent/child relationship. With this relationship, the second item listed must be a direct child of the first item, with no other elements between the two.

Consider the following script:

// Selects all a elements that are direct descendants nav element

**$**('nav > a')

In the following HTML, the first link would be selected, but not the second. This is because the first link is a direct child, but the second is inside of a div element.

<nav>

<a href="#">(First) This will be selected</a>

<div>

<a href="#">(Second) This will \*\*not\*\* be selected</a>

</div>

</nav>

### Descendants

To select elements where the targeted element is a descendant of another element, regardless of how many levels exist between the two, simply put a space between the two selectors.

// Selects all a elements that are descendants nav element

// The elements can appear anywhere inside of the element listed first

**$**('nav a')

While the difference in syntax is subtle, it makes a big difference in regards to the items selected. Using the same HTML from above, both a elements would be selected by using the syntax $('nav a).

<nav>

<a href="#">(First) This will be selected</a>

<div>

<a href="#">(Second) This will be selected</a>

</div>

</nav>

## SELECTING ITEMS BY POSITION

When creating a dynamic UI, you might not know anything else about an item other than its position. jQuery offers you the ability to both locate an item by its position in a collection, or to find out the index location of an item.

#### Finding the index of an item

If you have a reference to an object, but need to know where it sits in a collection of items, you can use the index method. The index method will return the zero based (ordinal) location of the item, or -1 if the item isn't found.

var currentElement = **$**('some selector');

var parent = **$**('some selector');

var index = parent.children().index(currentElement);

#### Finding an item by its position

If you know the zero based (ordinal) location of an item in a collection, you can obtain a reference to that item by using the get method.

var parent = **$**('some selector');

var element = parent.children().get(index);

**get return type**

One important note about get is it returns a JavaScript DOM object, **not** a jQuery object. This means the various jQuery methods, such as attr and text are not available. In order to call jQuery methods on the object, you must convert it to a jQuery object. Fortunately, this is just as easy as passing the object into the jQuery constructor. The above code would become:

var parent = **$**('some selector');

var element = parent.children().get(index);

var jQueryObject = **$**(element);

Or, you could distill it down to two lines

var parent = **$**('some selector');

var jQueryObject = **$**(parent.children().get(index));

Or, I suppose, down to one line.

var jQueryObject = **$**(**$**('some selector').children().get(index));

It's all just a matter of personal preference.

## ADDING AND REMOVING CLASSES

As we've already seen, jQuery leans on CSS syntax for quite a bit of functionality. Keeping with this theme, jQuery also makes it very easy to manipulate the classes an element is currently using. In fact, you'll notice most libraries that use jQuery to manipulate the UI will also come with a stylesheet that defines the set of classes their code will use to enable the functionality.

#### Adding a class

Adding a class to an element is just as easy as calling addClass.

currentElement.addClass('class-name');

#### Removing a class

Removing a class from an element is just as easy as calling removeClass. If the element in question was not already decorated with the class you're trying to remove, the method will simply return.

currentElement.removeClass('class-name');

## WORKING WITH ATTRIBUTES

As we've discussed, HTML allows you to both work with existing attributes, as well as add your own attributes as needed. This allows you to decorate elements with various flags and notes that you can use to enable the functionality you desire.

#### Retrieving an attribute value

To retrieve an attribute value, simply use the attr method with one parameter, the name of the attribute you wish to retrieve.

alert(**$**('selector').attr('attribute-name'));

#### Modifying an attribute value

To update an attribute value, use the attr method with two parameters, the first being the name of the attribute and the second the new value you wish to use.

**$**('selector').attr('attribute-name', 'new value');

## MODIFYING CONTENT

Beyond just working with classes and attributes, jQuery allows you to modify the content of an element as well.

#### Updating text and HTML

jQuery offers you the ability to update the text inside of an element by using the text method, and the HTML inside of an element by using the html method. Both methods will replace all of the content of an element.

The main difference between the two methods is html will update (and parse) the HTML that's passed into the method, while text will be text only. If you pass markup into the text method, it will be HTML encoded, meaning all tags will be converted into the appropriate syntax to just display text, rather than markup. In other words, < will become &lt; and just display as < in the browser. By using text when you're only expecting text, you can mitigate cross-site scripting attacks.

**Examples**

// update the text

**$**(item).text('Hello, world!!');

// update the HTML

**$**(item).html('

## HELLO, WORLD!!

');

#### Basic event handlers

Earlier we saw how to register an event handler for document.ready. jQuery allows you to access almost any other event that is raised via the JavaScript DOM, as well as several others. While a deeper discussion of event handlers will begin in Module 2, let's take a look at a few to get us one step closer to creating real applications using jQuery.

#### Registering event handlers

To register an event handler, you will call the jQuery method that matches the event handler you're looking for. For example, if you wanted the click event, you'd use the click method. Methods for wiring up event handlers allow you to pass either an existing function, or create an anonymous method. Most developers prefer to use an anonymous method, as it makes it easier to keep the namespace clean and not have to name another item.

Inside of the event handler code, you can access the object that raised the event by using this. One important note about this is it is **not** a jQuery object but rather a DOM object; you can convert it by using the jQuery constructor as we've seen before: $(this).

**Examples**

// click event

// raised when the item is clicked

**$**(item).click(function() {

alert('clicked!!');

});

// hover event

// raised when the user moves their mouse over the item

**$**(item).hover(function() {

alert('hover!!');

});

// mouseout

// raised when the user moves their mouse away from an item

**$**(item).mouseout(function() {

alert('mouse out');

});

## WHAT HAPPENS WHEN SOMETHING ISN'T FOUND?

Imagine the following HTML.

<html>

<head><title>Demo</title></head>

<body>

<div class='blue'>Sample content</div>

<!-- scripts go here -->

</body>

<html>

Now imagine we updated the page to add a reference to jQuery, and then added the following script:

**$**('blue').click(function() {

alert('You clicked the div element!');

});

If you ran the page and clicked on the div tag, guess what would happen...

**Absolutely nothing!!**

Why? Well, it's subtle, and a very simple typo, but take another look at the selector syntax:

**$**('blue')

Notice how blue is missing which is needed to indicate a class? Because that's not there, the CSS selector is instructing jQuery to look for an element named blue, rather than a class named blue. Leaving off a or a# is probably the number one mistake developers make when using jQuery.

#### But, that seems like a bad thing

At first glance, that might seem like a design flaw for jQuery. However, this can be very advantageous. In fact, you'll see this as a common pattern throughout jQuery - if it can't find an item, it simply moves to the next line of code.

Remember removeClass? removeClass doesn't throw an error if the element in question doesn't have the class. What's nice about that is we don't have to remember if the class was there in the first place, or write code to check for it in advance.

While you will certainly run into cases where leaving off a causes an unexpected bug, that behavior will help make your code a bit simpler when you're dealing with dynamic HTML, which is exactly what jQuery is all about.

## JQUERY OBJECTS AND DOM OBJECTS

When you get down to it, jQuery is a wrapper around the JavaScript DOM, both enhancing and simplifying it. One thing to remember, though, is you are always writing JavaScript. As a result, depending on where you are in code, you might be switching back and forth between jQuery and JavaScript.

Take the following code as an example:

**$**('#some-button').click(function() {

// code here

});

As we've already discussed, we can add code to a click event by using the syntax above. And, we can access the object that raised the event by using the this keyword, as this will give us a reference to the object that raised the event. But, this is a DOM object, not a jQuery object. As a result, any of the jQuery methods we'd like to use aren't available, unless we convert it to a jQuery object by using the code below.

var variableName = **$**(this);

Obviously, that's not challenging, but it can be a bit confusing as to when the object is a jQuery object or a DOM object. More than anything, that comes with experience. The more you use jQuery, the more natural that will become. That said, one quick way to see if you have a reference to a jQuery object is to take advantage of some form of auto-complete, such as that provided through IntelliSense in Visual Studio. You'd notice in Visual Studio the IntelliSense wouldn't show the different jQuery methods, which is a quick reminder that you don't have a jQuery object and need to perform the conversion.

## JQUERY'S EACH FUNCTION

As powerful as the selectors are in jQuery, there will still be times when you need to loop through the collection of items returned. Fortunately, jQuery does support methods such as index and get, and each collection that is returned can be accessed as an array, meaning there is a length property available. However, rather than using a traditional for loop, you may decide it's easier to loop through a collection of objects using for/each.

For/each is a common construct available in most programming environments. In a nutshell, what for/each does is instructs the runtime to grab each item in the collection, assign it to a variable that you declare, and then allow you to operate on that variable. Consider the following code:

**$**('.some-class').each(function(item) {

// item will be a variable that will represent each object

});

In the above code, the item variable will automatically be assigned to every item in the collection. The first time through it'll be the first item, the second time the second, etc. This can be much easier than programming with a for statement.

The syntax for jQuery's each function is as follows:

* A collection (in the example we used a collection returned by a selector)
* The each function
* A callback function with a parameter. The name you use for the parameter will become the variable name assigned to each item in the collection.