jQuery Fundamentals

Course Introduction

It's no secret that jQuery's one of the most popular JavaScript libraries out there, and for good reason. You can do an amazing number of things with it, increase your productivity, and quite honestly, have a lot more fun along the way; especially if you have to deal with cross-browser types of issues. Well if you've been wanting to learn jQuery, or have been putting it off and didn't know where to start, you're in the right place. I'd like to welcome you to the jQuery Fundamentals course. My name's Dan Wahlin with Pluralsight, and throughout this course I'm going to be covering all the key fundamentals that you need to know to get started on maximizing your productivity with jQuery, dealing with cross-browser issues more effectively, and many other things along the way. With jQuery we can select items in the DOM, we can handle events, we can do AJAX calls, and the list goes on and on and on. And we'll be highlighting and covering some of those key features throughout this course. Now to get the most out of this course, you do need to have a background in JavaScript, so as long as you're comfortable with the overall syntax you'll be fine. You do need a background in HTML and CSS as well. Now, you don't have to be an expert at any of it, but as long as you're comfortable with what an HTML tag is, what CSS is, what a CSS class is and those types of things, then you're getting quite a bit out of this course because jQuery is all about manipulating HTML, CSS, and other aspects of your web page. So let's go ahead and get started and jump in to module one and talk about why would we want to use jQuery in the first place.

Introduction to jQuery

In this first module we're going to talk about some of the reasons you'd want to use jQuery, and some of the features it provides out of the box. And then we're going to go into how we actually reference jQuery, and there's a couple different ways you can do it, you'll see. We can actually load a script locally in a web page, or we can use something called a CDN, or Content Delivery Network, and I'll talk about the pros and cons of both approaches. From there we're going to talk about after you have jQuery referenced in a page, how do you actually know when that page is loaded at runtime in the browser so that you can then use jQuery to interact with the page; and it's called the Document Ready, and we'll talk about how that works and why you'd need it. And then we'll wrap up with a quick tour of the jQuery documentation out there; it's a very good thing to know, because, although I'll teach you all the key fundamentals you'll need in this course, after you get done there's certainly going to be additional features you might want to add, and you need to know where to go look those up and get more information about them. So we'll talk about that in the jQuery documentation section. So let's go ahead and get started by talking about why use jQuery, and what are some of the different features it provides.

Why use jQuery?

Let's start things off by asking that all-important question of what is jQuery, and why would I want to use it in the first place? Well, I asked the same exact question many years ago when I got started, and the good news is, the more I dug into it the more impressed I was. So jQuery is just a single JavaScript file, that's all it is; nothing more, nothing less, but it's a really powerful single JavaScript file, and it's cross-browser, and that's why jQuery, at least in my opinion, took off many years ago when it first came out, because we had to deal with IE 6, and then Firefox, and then Chrome started to come on the scene later, and just a lot of issues there with cross-browser coding. Well jQuery makes it so you can focus on your app; you don't have to worry if you're on Safari or Chrome or whatever it is, so unless you have a very specific browser feature, jQuery's going to take care of it for you; whether it's finding things in the DOM, or handling events, or making AJAX calls, it'll take care of the behind the scenes heavy lifting to figure out what browser it is and how to deal with it. Now as far as features go, one of the most powerful features it offers is called Selectors; this is the ability to select things in the DOM and then manipulate those. And you'll see there's an entire Selector language with different expressions we can write that we'll be covering later in the course, so it's an awesome feature that literally will save you hours over what we did in the old days. When it comes to events, jQuery has a very rich event infrastructure, and as I mentioned, it deals with browsers very nicely. You don't have to worry about Safari, Firefox, and all the others, you can just write a single code base and know that your events are going to work and be handled appropriately. In addition to that, we have very rich support for AJAX: Asynchronies JavaScript And XML. And although we don't really use XML that much nowadays, you can really use jQuery to call back to a server and get whatever data you'd like. Nowadays it's usually JSON data: JavaScript Object Notation, and later in the course we'll be talking about that and how AJAX works with jQuery. And then finally in addition to these features, and actually many others that I won't have time here to go into, there's a very rich community of free jQuery plug-ins. So if you need something like a light box to display pictures, or you want to touch-friendly way to swipe through pictures, or you want an auto-complete textbox that as they type it pops down things that match up with what they've typed. You could certainly write all that by hand, but there's thousands of plug-ins out there that you can plug into your page really quickly and get started right off the bat without writing a lot of code. Now, why use jQuery? Well, the cross-browser issue is probably one of the biggest reasons, but there's a lot of different reasons actually, and I think when you go look at simple things like how would I find multiple div's that have a class called foo on them? Well if you have modern browsers, you could use the new Query Selector functionality, but if you have older browsers, there's really not a great way to do that. Well jQuery makes that a piece of cake; with one simple and very concise line of code, you could find all the div's that have a specific class on them, and you don't even have to write the looping code to do it. Another question: how do you apply styles to multiple elements with one line of code? Now it can be done with just pure vanilla, plain JavaScript, but you're definitely going to have to do some looping and other operations. Well the jQuery with just a single, again very concise line of code, you can apply styles or classes to multiple elements in the DOM. How do you handle events in a cross-browser manner? Well, most of the modern browsers nowadays have a standard way to do that; we can just add an event listener and it's pretty easy and not a lot of code. But, if you do still have to deal with some of the older browsers, you'll know that things like IE 6 doesn't support events in the same way that they should've, according to the spec anyway. So jQuery takes that headache out of the picture, and really, we just don't want to deal with these cross-browser issues at all; we want to write our application, focus on the logic, focus on the UI, and have fun doing it. And that to me is where jQuery really shines; it makes client-side development fun again, because we don't have to worry as much about the browser.

Getting Started with jQuery

So how do you get started adding jQuery into a web page? Well, you can head off to jquery.com, and there's two different options you can download. If you're going to be supporting IE 6 - 8 then you're probably going to want to go with jQuery 1, and then there's 1.9, and the versions will change. If you don't have to support IE 6 - 8 and you only want to go with modern browsers, then you can go with jQuery 2.x, and that'd give you a more slimmed down version, it's more modular, and it's not going to have all that overhead associated with some of the older browsers; so it's kind of a more lean script if you will. Now from there all you have to do is reference the script in your web page; very, very simple, so go to jquery.com, grab the appropriate version based on your requirements, and then add that script into your web page. Now the editor that you use to do it can be really anything; it can be something as simple as notepad, or WordPad, or whatever you like. There's a lot of great editors out there; Visual Studio, Eclipse, Aptana, I like Sublime Text a lot, WebStorm, and many, many others. Really you just need a text editor; the same thing you would use with HTML for that matter, or CSS. So now that we've talked about how to get started, let's run off to jquery.com real quick and take a look at it. To work with jQuery you want to head over to jquery.com in your browser, and from the home page you'll see a lot of the different features; Lightweight Footprint, CSS3 Compliant, Cross-Browser, as I've mentioned, and right here is the important part though, we can download jQuery. And you'll notice that there's actually two versions; now these versions will change over time of course, but right now it's v1.9.1, and 2.0. And you may look at that and go, well, why are there two, and which one would I pick? Well, the 1.9.1, or really any 1.x I'm going to say, is designed to be backward compatible, all the way back to IE 6. Anything with a 2.x is going to be IE 9 or higher, and any of the modern browsers out there. So if you need IE 6 through 8 support, you're going to go with a 1.x; otherwise, if you don't need that and you just want to target the modern browsers out there, you can go with the 2.x, whatever that is. So if I click on this, you'll see that I can now come on in and download either the 1.x or the 2.x. Now you have a couple options here as well; the compressed version is going to be very, very small, it's what we call the minified or minimized version, and it's all compressed up, unnecessary white space has been removed, variables have been shortened, things like that to make it really small for production. If you want to debug though and step through jQuery from your code, you might want to grab the uncompressed, more readable version, and that's this version here. And we have that for the 1.x, and you can see the 2.x. So let's assume that at work I still have IE 8 hanging around, which is often the case in a lot of big corporations. Then I might grab the 1.x version, so I'm going to grab this one here, and I'm just going to Save Link As, and then I can save it to wherever I'd like. And I'm just going to call this jquery.js to keep it really simple, and save that. Now, all I have to do is go into my web page, and I have a very simple one here, and either up in the head or at the end of the body, add my script tag. So if you've ever worked with script tags, you'll be pretty familiar with the following. We'll simply come in and say script; we can say type is text/javascript. You don't technically have to put that if you're doing HTML 5 apps, but I'll add it. Source equals, and then I'd point to the path of my jQuery's; we'll say jquery.js for this example, and then add my script tag. And that's all you have to do to actually get started with jQuery; it's very, very simple. The important thing though to understand is again, 1.x is for IE 6 through 8 and the new browsers, 2.x is only for the modern or new browsers, so keep that in mind when you pick the version of jQuery you'd like to work with.

Using Content Delivery Networks

In addition to loading jQuery from your own server, you can also load it from a Microsoft, Google, or other server, and we call these external servers Content Delivery Networks. Now there's actually several benefits if you want to load your jQuery script from Microsoft, or from Google, or even from the jQuery site. If you actually go to a site, and let's say you went to say a new site, and they use this URL right here for Google; well if you go to that site, that script is now cached for that domain. If you then go to another site that also uses this exact same URL, then the script is already there; it's already cached. So one of the big benefits of using the CDN is you can get some caching benefits because many of the popular sites out there, instead of hosting jQuery locally, will grab it off a Microsoft or Google site. Now there's a couple other benefits too. Microsoft and Google have datacenters all around the world, so if you're in Australia, odds are you're actually going to be grabbing that jQuery script from a server in Australia, instead of having to go wherever your server's located to grab that script. So you can get some speed benefits there. Number two, you also get something called parallelism; browsers limit the number of simultaneous ACDP calls they can make to a given domain. So, because this is going to be in a different domain from your actual web server, the browser will allow this to load in addition to whatever you're loading that's local. So this provides those kind of big benefits: caching, I call it regionality, or closer relationship to the data services out there and servers, and then finally the parallelism of the browser actually being able to load multiple scripts from different domains. Now this is great and at first glance you go, well geeze, how can I lose? You know, I get those three big benefits, and maybe some others, so why would I not want to do this? Well, what happens if, due to the network, or maybe even due to the Content Delivery Network, that you can't get the jQuery script? That would certainly cause a big problem, especially if your app is based on jQuery, because if it can't load that script, you obviously are not going to be able to get to all the cool jQuery functionality that we're going to be covering throughout this course. Well, what you can do in that case is you can provide a fallback, and this is a nice little trick, very simple to do. You can go out to, in this case Google, and we'll try to grab jQuery, and here we would list a specific version of it, such as 1.9.1, or 2.0.0. Then what we do is if that loads successfully, the Window object in the browser will have a new property off of it called jQuery. Now that happens as the jQuery script actually loads. Now, if the jQuery script can't load, then we're going to have a fallback. So if the jQuery script loads appropriately, we'll have this jQuery property; otherwise, we'll fall back, and we're going to do a document.write, and write out a script tag that points to a local copy of jQuery. Well now we can have the best of both worlds; we can leverage the caching and the regional servers, and potential parallelism, and stuff like that with the CDN, but for some reason we can't get that CDN script, we can actually just load it locally. And that's an example of working with Content Delivery Networks, and it's definitely something that I recommend you look into, especially in Internet type scenarios. Now if you're on an intranet app, I'd say it really depends; if the users are all local it probably makes sense actually just to hit a local copy of jQuery off of your server, if you have users distributed all over the place, maybe around the world for instance for your company, then you might want to look at the CDN; it just depends on your network requirements, firewall constraints, and things like that. So that's some information on how we can use the CDN, and how we can have a nice safe fallback to load a local script if we need it.

Using the jQuery ready() Function

Once you've loaded jQuery into a page, either with a local copy, or by calling out to a CDN, you can then take it up a notch and start to use jQuery of course, and that's, after all, what we're after. So in this section we're going to talk briefly about a little function jQuery provides, which is very, very useful, and one that you'll find yourself using a lot called ready. Now, one of the important things to understand with jQuery is that when it's loaded, the window object, as mentioned earlier, has a little property on it called jQuery. Now you can use that object directly or you can use an alias of it, which is simply the dollar. And so notice here, I have $(document), and what we're doing is passing the document object from the browser into jQuery, and then we're going to call the jQuery ready function. Now, this doesn't wait until everything's loaded, and what I mean by that is with things like window onload in JavaScript, it's not called until everything's loaded; images, CSS, and all of those types of files. With the document ready, it detects once the DOM hierarchy has been loaded, so before all the images and CSS have been loaded. And that allows you to start manipulating the DOM, and that could include things like I want to show or hide, or add different things into the DOM; I can do that right when the DOM's ready; I don't have to wait for all the images to download and things like that. Now, all we have to do is say $(document), and then notice we're passing that in to this little jQuery function, so again the dollar here, you're going to see over and over and over throughout the class, and it's simply an alias for the word jQuery. Now if you prefer to type the word jQuery you could do that too. Now, myself and many others out there, we just prefer to go with the dollar because it works and saves a little bit of typing. So we're going to pass the document in that wraps it with a special jQuery wrapper. Now we can get to jQuery functionality and know when the document is ready. Now when it is ready, what do you want to do? Well you have to give jQuery your own custom functions, and notice we have an in line anonymous function here, and this is where you perform the action that you should be doing. So we could be adding things to the DOM, we could maybe trigger an AJAX call to go get some data from the server, there's a lot of different things we could do. So let's take a look at the document ready and how we can use it in a live web page, and see when it's called and compare it to the window onload. Now that jQuery's available in this simple page here, let's take a look at the difference between window onload, and the document ready that jQuery provides. So I'm going to add our script tag in, and we'll go ahead and add our type just to be official here, and then we'll add our closing. Now what I'm going to do first is kind of the standard JavaScript way to detect a load that you'll see out there a lot. Now there's multiple ways to do this, but this is what I see a lot of people doing. So we're going to say window.onload, and we're going to call our own custom function here, and then in this function we're just going to do a simple alert, just to say hey, the window loaded. So we'll say alert(‘Window loaded'). Alright, now that's kind of the standard way to do it, so let me save that, we'll run off to the browser here, refresh, and you can see when the loader was called. Alright, so nothing real fancy at this point. Well, we can also come in and do it the jQuery way. So I'm going to come in, in fact, let me just put it after to kind of prove the fact that timing will be faster when it comes to using jQuery. So because jQuery's already loaded, we could either do this: jQuery(document), and then we could do our ready; or I can simply replace the keyword jQuery with $. And in all the demos I'll be doing throughout this course, I'll be using the alias of $; it provides a little bit of a shortcut, don't have to type as much. So we're going to pass the document into jQuery, so we're literally invoking a jQuery function here, and it's going to return something called a rapset. Now you don't really need to know what that is, but we'll be covering that in the next module on Selectors a little bit more. So we pass in the document, and because this is now a jQuery object, we can now get to the jQuery ready. Now ready is of course defined up in our script. Now what I'm going to do here is say okay, when it's ready what do I want to do? Well, just like we did with window onload we need to give it a function, so we're going to say function, and then we'll go ahead and end our function, and then we need to end the parenthesis here for our ready, so we'll put that in semicolon, and here, I'm going to say alert, and we'll say ‘Document loaded' so that we know it's a little bit different from the window load to the document load. Now really what this is, in fact I'm going to change it, it's really DOM has loaded; so not all the images have loaded, not all the CSS has loaded, but the DOM, the HTML DOM has loaded, so if we had div down in here, and multiple spans and tables, that's all ready to go, that's how we know that jQuery could then be used to manipulate this. Now, another option would be I could just move the script below at the very bottom of --- right before the closing body tag, then I would know that this stuff would be loaded first, then the script would execute, and that would be another alternative. But if you want to put it up in the head like we've done here, then you'll want to use the document ready. Alright, now let's see which one's going to be called for, so window.onload should be called and do an alert here, and then document.ready should be called, and it also should do an alert. So let's go back and refresh, and notice that the DOM loaded was called first, now that is correct, because it doesn't wait for images and all this other stuff; now you might say, but there are no images; true, but in a real web page, odds are you have images and style sheets and other things; this is going to be called as the HTML DOM is loaded, and not wait on all that stuff. Then you can see the actual window is loaded. Well now we can use jQuery to do some cool things, so I'm going to give a quick segue into what we'll cover in the next module. So let's say we had an Id down here on a div of "myContent", and inside of here we have this My jQuery enabled Page!. So what we can start to learn with jQuery is, instead of doing these simple alerts, I'm just going to comment this one out, we can actually come in and start to manipulate things. The DOM is loaded, so we can go find things. So I'm going to do our very first selector, and this is going to be our hello world to get it out of the way early on. And I'm going to say go find myContent. Now this is called an Id selector, and we're going to cover this officially in the next module, but for now, this will go in and do the equivalent of a document get element by Id; that's what the hash or the pound means here. myContent is obviously the Id. Then I want to change the HTML here, so just like with ready off of a jQuery object, we can also do things such as change the HTML, and I'm just going to say ‘Hello World' here. So now, once jQuery detects that the DOM is loaded, we'll go find our div with the Id of myContent, and we're going to change the HTML to Hello World, so let's go ahead and run that, and now, once we run this, we'll be off and running with our first little jQuery example, and you can see it worked; we get Hello World. Now, I'll certainly admit that's not very impressive, but we've got to start somewhere. So in the next module we'll be talking much more about Selectors and not only Id's but classes, and some more complex Selectors, and that's where you're really going to see the power of jQuery. But at this point you've now seen how to detect when the DOM was loaded through the document ready, and a simple example of how we can even manipulate the DOM using jQuery.

Getting to Know the jQuery Documentation

Now that you've seen how jQuery can be added into a page and the fundamentals that can be used to get started with it, let's take a little bit of time to run over to the jQuery documentations site. Now this is a site that I have bookmarked, and something that I use constantly in my applications, because I'd love to say I've memorized every single function that's available, but some of them I just don't use that much; some of them I use every day, so it really depends on the application. I can tell you that if you want to get good at jQuery, you really do need to spend some time getting good at the jQuery documentation. By doing that, you're going to really speed up your productivity, and it's going to be a lot less of a frustrating process. So let's go ahead and head over to jquery.com and take a look at what we can get to as far as docs go. In addition to getting the script from jquery.com, you can also get to the API Documentation. I'd argue this is one of the most important things you need to know about, especially if you're going to be using jQuery a lot. When I first started jQuery right after it came out, I didn't know a whole lot about the different functions, and luckily they did have some docs back then, and I spent a lot of time in those, because I'd come up with some new thing and it felt like I was climbing up a cliff because I had no idea what to do. Well fortunately, the documentation over the years has gotten better and better, and so let's just take a quick look at what we can get to and what they provide. So you'll notice here on the left, once I go to the API docs here, that I have some different categories; so if we want to work with AJAX, which we'll have a whole module on later in the course, or CSS functions, working with Attributes, some of the Core functionality, and I can scroll through and get to effects for things like animations and events, and you can see it kind of goes on and on and on, including DOM manipulation. Well, a lot of this stuff will be used in applications. Chances are you're probably going to be manipulating the DOM with jQuery. You're probably going to be making AJAX calls if that's how your application is set up, and if that's how your server is set up. So, we're going to be covering things like events and DOM and all of that in this course, and so you can see that we can just scroll through and get to kind of a vast amount of information, so if we click on Selectors, for instance, which will be the next module of the course, we can see all the different things related to Selectors. Now if we go back to, let's say the DOM manipulation, let's say I want to insert something into the DOM, then you'll notice there's several different methods for doing it; I showed the HTML a little bit earlier in this module, but we can append things, we can pre-pend things, we can change the text, and what's nice about these is once you click on it, find the function you need, you'll get a nice little explanation of it; there's some different options on how you can call these functions, and they'll even tell you when it was added; which version. So if you're using maybe an older version of jQuery for whatever reason, then you'll know what version this particular function or property is supported in. Now if we come on down, what I really like about it is you can get to some info on how to use these different functions or properties. So in this case, it shows us how we can retrieve HTML content, or if we come on down, you can see how we can actually update HTML or retrieve and use some other functionality; we can even run a live demo here. So there's a lot of great stuff in here for how to retrieve HTML; this now shows us how we can set the value of HTML, which I demoed a little bit earlier with our Hello World, and we should see down here another example, and there it is. And you can see that they're actually adding a span into the HTML. And that's the type of documentation that you'll see for most of the different functions and properties available in jQuery. Now, there's certainly some that have less docs, and some have a whole bunch, but in general, this is where you need to go to get started. And it's really going to give you a nice jumpstart if you learn how to navigate this kind of left hand side here. So definitely, I would recommend get good at the docs, spend some time in the docs, and it's really going to help your jQuery development process.

Summary

In this module you've seen an overview of what jQuery is and what it offers; we know it's a very good cross- browser JavaScript library; you can go to jquery.com to get it, and there's different versions you can grab based on which browsers you want to support. Now, once you have jQuery we can add it into the page, and then I showed how we can use things like document ready to detect when the DOM is loaded, and then we can manipulate DOM elements like I showed with the jQuery HTML function. The other thing I showed at the end was the jQuery Documentation, and I want to emphasize again how important it is to at least feel comfortable navigating through that Documentation to find things, because it'll really speed up and make you a more productive developer by learning it. So now that we've talked about the fundamentals of jQuery, it's time to jump in to some more fun with it. In the next module, we're going to talk about jQuery Selectors, and how we can use Selectors to find DOM elements.

Using jQuery Selectors

What are Selectors?

Welcome to module 2, Using jQuery Selectors. My name is Dan Wahlin with Pluralsight, and in this module we're going to be talking about one of the very key things you need to know as a jQuery developer. So what are we going to cover? Well, we're going to start off by first talking about what are selectors and why you care about them as a developer. From there we're going to go into a lot of the different selectors that you need to know. So we're going to talk about how to select tags, how to select notes based upon ID, based upon class name, different ways to select attributes, input nodes such as forms tags, and then we'll wrap up by talking about some other more advanced selector features that are really, really nice to know about. So let's go ahead and jump into the first section, which is what are selectors and why you care about them as a jQuery developer. Well, a selector is simply a way to select a node from the DOM, and when I say node, you can think of it as a tag, an element, whatever you want to call it, but when the browser loads all the different HTML, it loads that up into memory into the document object model and jQuery uses selectors to allow you to actually go into that memory, the DOM, select those different nodes, and then do something with them. Now the cool stuff comes into play once you actually find the node and you start doing something with it, but you need to know how to select a node before you can actually do something with it, and that's what we're going to be covering in this module. So all the key stuff you need to know to do things such as come into a div and find it based upon ID, or find the div if it didn't have an ID based upon the class. Or maybe you want to find the span that's a first child of a given div with an ID of CustomersDiv, there's all kinds of different things you can do. Now you can certainly do all of this just using normal JavaScript. However, you're going to write a lot more code, you're going to have to worry about some cross-browser issues in some cases, and it's just a lot more work in general. So, by using jQuery, you're going to see we number one, have a very, very compact syntax, we work cross-browser, and it's just super easy to find things based upon ID, class name, styles, attributes, all kinds of stuff. So, very, very nice once you learn it, and this, quite honestly is the first thing you need to get good at. So what's jQuery selector syntax look like? Well, it actually can look one of two ways. I normally prefer to do the dollar, which you saw in the previous module when we showed the dollar document ready, but the dollar is the same thing as jQuery, and I pointed that out earlier, but most people you'll see will use the dollar, and then what you do is in quotes you'll put what it is you want to find. So in the next sections in this module, we're going to talk about how to find IDs, how to find classes, how to find tag names, all that good stuff, and show you how easy it is, and it really is easy once you learn the fundamentals of how to do it. So the next thing up is going to be how to find things by tag name. So let's go ahead and take a look at how that works.

Selecting Nodes by Tag Name

If you've ever needed to select a specific tag out of your page as it loads, then you can do that with document.getelementsbytagname using regular JavaScript. And that works cross-browser, there's not really a problem there at all. A little bit long in my opinion, but it still works. Well, in jQuery you can do the same thing, but it's actually much more compact and I think it's just plain easier, and that's what we're going to cover in this section. So here's an example of selecting the paragraph and the anchor tags using a jQuery selector. So you'll notice that we start with the jQuery object, we put the dollar, we then call into a function, and we pass in the selector syntax, which in this case is simply the name of the tag or element that you want to call. Now what this will do is go up into the DOM, it'll look for all the tags that are paragraph or anchor tags, and return those back. So this one up top here would return a collection of paragraph tags, this one will return a collection of anchor tags. So very, very simple and definitely more compact than the longer document.getelementsbytagname. In addition to that, what if we wanted to select not only the paragraph tags, but also all the anchor tags, but in just one statement, we want to merge those into one collection. Well we can do that as well, we just simply do a comma separated list of the tag names. So in this example here, we're going to again start with the jQuery object, call into the function for the selector, and pass in that we want to select the paragraph, the anchor, and the span tags. Now it's going to return all of those in one single collection, and then you could loop through those, you could apply styles, you could do all kinds of stuff once you get that collection back, very simple to work with. Now, you can even go in and instead of just saying find me all these specific tags, what if you just want to go in and find specific tr tags that are nested directly within a table tag for instance. Well, you can combine that as well using the ancestor descendant syntax, which will go in and select the table tag, and then you'll notice that we put a space, and a space then will separate the descendant that we'd like to select. So in this example we'd say, go find the table, but specifically don't return the table tag, just return all of the tr tags that are descendants of that particular ancestor, that particular parent. So that'll go in it and grab those tr tags and allow you to manipulate the children and go even deeper if you'd like. So let's take a look at an example of using selectors and show you how easy it is to work with.

Demo

Let's take a look at how we can select tag names within our HTML page very easily with jQuery. So first off I have a really simple page, and this is loaded inside of Visual Studio. Now I want to emphasize that jQuery works with just plain HMTL, PHP, Ruby on Rails, ASP.NET, Web Forms, MVC, whatever you may want to use. So, although this is my tool of choice for now, which is Visual Studio, you can certainly do this in Notepad, you could use Aptana, and there's all kinds of different editors, Eclipse would work great. So nothing specific to Microsoft or any of the Microsoft technologies, but I will say that things like ASP.NET MVC play very nicely with jQuery out of the box. So I have an ASP.NET MVC application in front of us here. And you'll notice in my Scripts folder I have the jQuery script. I also have the vsdoc script so I get some IntelliSense, and that was covered a little bit in module 1. So what I'm going to do is come in and we're going to do a real simple selector. So I've already loaded the script and I have the document ready function all wired up and ready to go, so when the DOM is loaded we're able to go in and get the tags that we want out of it. So all we're going to do, is like I showed earlier in the slides, we can come in and simply say I would like to find all the, for instance, p tags, or let's find that img tag. And that's going to return a collection of images actually, because there might be more than one of those. So let's go ahead and just to keep it simple, we'll do an alert on that collection, we'll say what is the length of it. So let's go ahead and run that now and when it runs we'll hopefully see that we, at this stage, have one image that pops up, and you can see that works appropriately. Let's go in and there's also some div tags. So let's change this to divs, we'll change this guy to divs, and then we'll find all the div tags. Now I happen to have three that are actually in the document, so I have a div there and then there's two more divs right here. There's one that'll be injected dynamically at runtime as well, so we should see about four of them that appear once we run this, and you'll see we do get back four divs. Now at this stage, what do you do once you've selected a tag? Well, in a separate module I'm going to cover some cool things you can do, but let me show you a few of those as we get going here. So one of the things we could do is we could actually go and for each div, we might want to go in and change for instance the style. Now this is a class, but we could also go in and directly influence CSS styles. So if you've been doing this the custom way with JavaScript, and just out of the box type JavaScript, you'd probably assume we'd come into the style and we'd go from there, but that's not how it works in jQuery. In fact, you won't actually see styles show up in the IntelliSense here. This is a jQuery object that's being returned, so it's actually a DOM object, which represents all the divs wrapped inside of a jQuery object. Now how do we know that? Well, because this is the jQuery object, this is a call out to the jQuery function, and that returns us a jQuery, in this case, collection. So what we could do is we want to change, for instance, the background color of all the divs. You'll notice currently I have blue and red and that's about it, this top one has nothing. Well, we can go in and say find all the divs, and you'd probably think, well then you've got to loop through to apply the styles. Well in jQuery that's one of the nice features, you don't have to. We could come in and say I'd like to change the CSS, and you'll notice I get some really nice IntelliSense here, and I want to change the property called background-color, and I'd like to change it to, let's just say green to make it really obvious. That's all I'd have to do. If we find four divs, three divs, whatever it is, it's going to automatically iterate through those, go in, and apply this style onto each of those divs. Now it's actually going to influence the style attribute that you would normally type down in the HTML. So when it gets done, it's like we did this. It would be something like that, but it'd be up in memory because we're doing it with the DOM. So that'd be the basic syntax, and you'll notice how simple that is. No looping, very compact, very easy to use. So let's go ahead and run that and we'll see what we have here. Now it doesn't look phenomenal, but you'll see that that was one div, here was our second div, and then here was our third div, and you'll see it changed those to green and you can see how easy it was to not only find the divs, but to also do some work on those divs. And we'll have a lot more I'll be covering in later modules about how to work with the data once you get it. Now one more I want to show you real quick though, is what if we wanted to see the HTML inside of the divs, or maybe we want to change the HTML inside of the divs. Well again, normally what most people would do is they'd come in and make the collection, and then they'd do something like a for loop or they do the for, and we can do the loop, start at 0 and loop through each div. Well, much, much easier to do in jQuery. So I'm going to go ahead and start from scratch so we'll do our selector. Now I'm going to say for each item that's returned, call a function and do something in the function. And I'm going to keep it real simple since we're not really dealing a lot right now with these functions, I just want to show you a couple of them. And I'm going to do an alert on the div's HTML so that we can see the HTML inside of that div. Now, as it loops through each item, it's going to pass you that item. And the way we can get to it is you do a jQuery wrapper around this. Now this represents the item you're looping through. So we would say this.html to do it the jQuery way. Now if you've ever done the innerHTML before using traditional raw JavaScript, then that type of stuff can be done, but in this case we'd actually have to go this way. So if I run this, what'll happen is as we iterate through each div, it's now going to pop up, so this represents the table in the upper left, here's the Blue div, that's the contents, here's the red div you can see, and then there's a hidden one that's inside of here that is written out for us. And there we go. So, there's many, many things we can do once you do the selector on a tag, you could apply styles, you could change class names, you can iterate through each one, you can toggle them, and there's a lot more we're going to cover in the upcoming modules, but that's one of the first things I wanted to show you. So this particular syntax with the selector works with anything you want. You can find p tags, you can do all that kind of stuff, but we can also do more as I mentioned earlier. If we want to find, I don't have any anchors, but let's say we want to find the div and the span tags. And let's see how many, we had four divs we know, and we should have I believe two spans in here. So we'll probably get around 6 total items back if we do it this way. So let's do an alert on the collection and so we'll say collection.length, and it should be in the ballpark of about six. We'll see what we get, there might be one I missed, but yeah, there we go, so we get six. So we get the three divs here plus the kind of hidden one, plus inside of each of these two colored divs we have a span tag, and that's how we can get those. So you can see it's very, very easy to select all kinds of tags. Now what's cool about this, the same exact technique that I showed you earlier. If we wanted to say change the CSS of some property, we could say propName, value, and that would now not only change the propName on just the divs, but it would also iterate through all those span tags. So, very, very compact, very easy to work with. So that's an example of using tag selectors, or element selectors, whatever you'd like to call them to select specific tags in the DOM that your webpage has.

Selecting Nodes by ID

The fastest type of selector you can do with jQuery is what's called an ID selector. Oftentimes in your HTML page you'll go in and you'll give IDs, identifiers, to your specific tags to make it easy to find, and that's pretty common, been around for a long, long time. And normally you'd use document.getelementbyid to find those specific IDs. Well, in jQuery it's even more compact than that. What we can do is we'll, just like we learned with the tag selectors in the previous exercise in the previous part, you come in and use the jQuery object, so the dollar, call into the selector function, and you put the name of your ID. So what we have here down at the bottom is we have a paragraph tag with an id of myID, but if we just put myID in here as we just learned, it would look for a myID tag, and that's obviously not what we want, we want to select it based upon the ID. So what this will do is if you put the hash, or the numeric-type symbol, then it'll automatically go in and know to look for something that's an ID, and that's kind of jQuery's way of knowing that you don't just want to tag, you want a specific ID, and you've got to remember, especially if you're new to this, to put that in front or it's not going to work right. So let's take a look at an example of using an ID selector and how easy it is to work with those.

ID Selector Demo

So back in Visual Studio, I have my document ready function and you'll notice if I scroll on down that we have a couple different tags in here, we have a form tag with an ID, and that's about it, we don't have a whole lot going on as far as IDs right now. So let's go ahead and add in just right here at the top, let's go ahead and add in a div since forms aren't real exciting at this point, and we'll just call this TestDiv. And inside of that we might have, This is my div text. So if we want to find that particular div, TestDiv, all we'd have to do is once we know the DOM is loaded, we can come on in, we'll start with our jQuery selector, and we have to do the pound, or the hash, and do the name of it, TestDiv. And then we can come in and let's say we want to, well let's grab the HTML and we'll actually write that to an alert. So HTML again is a jQuery function, and we'll just update that in an alert box. So let's pop that up and you'll see This is my div text, you can see that it found it. Now this is by far the fastest way if you're going to be using selectors in jQuery, it's by far the fastest way to find your different DOM elements. If I do something like we did in the previous module, or section I should say, and just do a p tag, well that's going to scan the entire DOM and find all of the paragraph tags. So, by doing the ID, it knows exactly what to look for and it's optimized in the document object model to be able to find that. So that's all you have to do to find those particular items. We could come down to wrap up this section and come to form1, put that in there, and that should return for the form all this table HTML. So let's go ahead and run that, and you can see it did. We have some stuff that was injected, but here is our table tag that was added into the DOM, which represents our form down here. So that's how easy it is to use ID selectors. The nice thing is, as shown in the previous section, in addition again, the HTML, I could certainly modify styles, I could change classes, I could even add children if I wanted, all programatically and it's very, very compact, and I just think it's a lot more easy to work with, it's a lot easier to work with I should say than getElementById, and then we put in TestDiv, which is the traditional way to do it. So, kind of take your pick there, but you get a lot of benefits by getting the jQuery object instead.

Selecting Nodes by Class Name

The next type of jQuery selector we're going to look at is the class name selector. It's very similar to the ID selector, but a little bit different character syntax. So to use the class name selector, you're actually going to use the period character inside of your selector name. So for example, we have the jQuery object you can see here, we call into the selector function, and instead of putting the pound or the hash, whatever you want to call that symbol, we're going to put the period. Now this says to jQuery, go find any element in the DOM that has a class attached to it called myClass. So that would match up with a paragraph tag, class equals MyClass, it would match up with an anchor tag, class equals myClass, we're not telling it which type of class. So the downside of this particular approach here is that it is going to scan the whole DOM, it's going to have to look through all the different objects that are loaded up in the memory, analyze if they have this specific class name, and then go ahead and return that set. Now, the upside is it's really convenient, you can see how easy this is. There is no equivalent way to do this with normal JavaScript. You'd have to do getElementsByTagName or you'd have to just loop through all for the DOM, loop through every single node, and then check the class name on it to make this work. So it is very compact, very easy to use. Now you can also combine this and do things like multiple class name searches. So in this example, you'll notice that we're just going to do very similar to what we showed earlier with selecting multiple tag names, but in this example we're going to grab the BlueDiv and the RedDiv, and that set of items, the set of DOM nodes that matches those class names will automatically be returned. So this would give you everything that has those class names out of the DOM. Now the other thing you can do, and this is good for performance in general, is you can go in and if you don't know the ID, it's always best to go through the ID if you happen to know that, but in this example you'll see we have an anchor tag, and then we want to find not all anchor tags, but an anchor tag, or multiple, that have the class myClass on it. So that is definitely way more efficient than just doing .myClass as in the previous slides, and the reason for that is we don't have to scan the whole DOM. This particular example will automatically allow us to go in and then we can grab all of these items and manipulate the classes. So really, really nice, really easy to work with, but I would recommend be smart about it. Just because jQuery selectors are very concise and very easy to work with doesn't necessarily mean they're going to give you the best performance in all situations. So, in this case, not too bad because we're locking it down, but just be aware of that as you're selecting your different items. So let's take a look at an example of using the class selector with jQuery within our webpage.

Class Name Selector Demo

Our webpage has two different divs, and these divs have some classes applied to them. So we have Blue div and a Red div and each in turn, you can see the syntax Visual Studio shows here has div.BlueDiv and div.RedDiv, and actually that looks very much like the jQuery selector syntax. So what I'm going to do is we're going to come into our source and find those divs, they're right here, and you can see the class names on them. So if we wanted to go in and find all the blue divs, then all we'd have to do is apply that class selector syntax. So we'd just simply come in, put the dot, which represents a class name, say BlueDiv, and then let's say we want to change the CSS style on that div. So this isn't modifying the class, but it is going in and actually adding a style. So let's say we want to change the border to 2 pixels solid red border around that. So what that'll do is go on in, find that div, and any div that matches that class, and in this case we're going to go ahead and apply that border, and you can see it works. Now I said find that div, and that's actually not entirely accurate. If we come back, this doesn't care if it's a div or a span or whatever it may be, we can come down and let's just add in a span to demonstrate the concept, and we'll do BlueDiv and just put This is my blue span. Let's go ahead and run that now. And because my selector didn't say what to find, you'll notice it also highlighted the second span, the second class that's applied within the webpage. So, not only is it very easy to do, but it could highlight some things you don't want necessarily, so you've got to kind of know what you're doing and what you want to select. Now to make this really efficient, let's say that, let's come back and I'm actually going to put that back in. And let's say we only want to target the div, we don't want to find everything that has BlueDiv. So well, as I showed you before, we'll come in and say div.BlueDiv, and now only the div will be highlighted with that particular style. You'll notice the span here doesn't have that particular style associated with it, so we're doing what we want here. Likewise, we could come on in and do the span, and what that'll do is as you'd expect, ignores the div because now we're being very specific in what the selector is doing. So, that's how easy it is to work with selectors using class names, something that's very powerful, but as I've mentioned a few times in this section, be really careful with it because it is going to scan the whole DOM when you just do things like this, so very nice. Now the final thing to wrap up, I also showed that you can do multiples of these. So we can put a comma and we could say RedDiv, and if we scroll on down, so for BlueDiv we have this and of course we have our span that has BlueDiv, but we also have another div with RedDiv applied. So this will now apply that border, the red border, which you probably won't be able to see too good in the RedDiv, because it's already red, but it'll apply that border around all three of those now, and so yeah, you can't really see it on the red because it's red on red, but very nice, very easy to work with, especially compared to doing this by hand where you'd go in, you'd have to loop through all these DOM elements and all that and what's really nice is if we just wanted to target the divs, we could also prefix both of those and say only focus on divs that have those class names. Now when we run it, you'll see that the span is left alone. So that's an example of using class name selectors with jQuery.

Selecting Nodes by Attribute Value

To this point, we've looked at how we can select different nodes out of the DOM by tag name, by ID or even by style class name, but there may be times when it's not enough just to find it by class name or by tag name, or even by ID, you actually want to go in, check a specific attribute for a value, and then find that based upon the value. So that's what we're going to learn about here is how we can start to combine a little more sophisticated syntax in our jQuery selectors. So, it's very easy to work with. In fact, if you've ever worked with something called XPath, which has been around for a long time with XSLT and some XML technologies, it uses a very similar syntax in its language. So what we're going to learn here is instead of just saying hey, let's go find all the anchor tags, what if we want to go say go find me an anchor tag that has a title attribute on that anchor tag. Well, we can do that with the following syntax. So again, jQuery object, call into the selector function, we're going to pass in the tag that we want, but we're going to say where the anchor tag has a title attribute. Now notice it doesn't say where title is equal to something or has a starts with or ends with value, it just says has a title. Now you can also go in though if you wanted to search for a very specific title, you could come in and say yeah, go find the anchor tag where, and every time I see these brackets here, I think in my mind where, the title attribute equals ProgrammingInfo. Now yes this is JavaScript, and normally you can do the double equals for comparison, but in this case we don't, it's a very specific syntax that you have to use with jQuery, so we just use a single equals here. So this is actually a really, really useful selector because there's oftentimes where for instance you might have a lot of different divs that have specific text that you know to look for. So you might have states as an example and you want to find all the divs that have the value AZ as the value of the text as an example. Well, you could do that very easily using this syntax. So again, before we run into the demo, think anytime you see the anchor or the div or whatever the tag is, if you see square brackets next to it, think where whatever it is. Now this one only shows title, but if for instance we had input type equals text, for a textbox, then you could go highlight all textboxes. Now there's other ways you can do that as well, but that would be an example of what you can do out of the box with this particular functionality.

Attribute Selector Demo

So let's take a look at how we can actually apply what you've just seen with attribute selectors into our page and actually select some things. So we're going to come in and do our normal jQuery selector. Now if I just went in and did div, we know that would just find all the divs, but I want to go in and say find me all the divs where there's a title attribute first off, like you saw earlier. So let's go ahead and we'll wrap an alert around that so we can see the length. And right now you'll notice I have a div here with a title, and that should be the only one I have in the page. So let's go ahead and run that as is and you'll notice that we get, there's one available, which is correct. Alright, so now that we've done that, let's take it up one notch and let's go find a specific one. Let's add on this div a title as well, we'll just say Contains Table. So now we don't want to just find the div where there's a title because that's going to return both of them, but we want the specific div that has Div Title. Now at first glance you may look at this and say why would you do that because if you know it's Div Title, this div has an ID. Well let's just assume that maybe there's a couple divs that have that same title, and we want to find all of those divs, we don't know the ID necessarily. So we search for Div Title and it does match the exact type of string, and let's just go ahead and put that in a divs collection and we'll do an alert on, let's do divs, yeah, we'll just do length again, and run that. And so what's going to happen now is you'll see we just find the 1. And let me change this up a little bit though. Let's change the case of the div to be a different case than what you see here and run this now and let's see what happens. You can see it is case sensitive, so that's definitely something you'll have to watch out for, it's definitely something I've been bitten with personally when I wasn't paying attention and not really worrying about the uppercase or lowercase version of it. So out of the box you'll need to know about that, but that makes it really nice. Now that just finds divs with titles. Let's go in and find any input tag that we might have, and we could just do input tag, but let's say I want to find an input tag that type is text. Now if we scroll down to the bottom of the form, you'll notice I have a couple of those. I have one right there and one right there with type as text. And so now we'll say go find all the input tags where the type of the input tag, the attribute, has a value of text. And I believe we'll get two of those back unless I have some others I'm not thinking about. So yeah, we get two of those back now you can see, and that matches up with that selector. And what's nice about this is this isn't divs anymore, it's inputs I guess. And now based on the little bit of syntax I've shown you, this is a jQuery object, so now it has full access to the jQuery API and so we could do things like I showed earlier, and maybe we want to change the background- color to say yellow of those particular inputs. So let's go ahead and run it and you'll see that we've now changed our textboxes to be yellow and it was super easy to work with. So that's an example of using ID selectors. Now at the end of this module, I'm going to have even more because you may want to find where it starts with text or ends with text or it just contains the word text. Right now it has to be an equivalent match, it has to match character for character, and as I showed you, the case matters. There's some other more flexible things we can do though and at the end of this module, I'll run into that as well, but that is an introduction to using selectors in jQuery, and it's something that I use actually a fair amount, very useful to know about, and something you'll definitely be applying as you work with jQuery more.

Selecting Input Nodes

Next up in our tour of jQuery selectors is selecting input nodes, very useful if you're working with forms, textboxes, textareas, selects, those types of things where the end user can input data into your application. So there's a new character syntax that you haven't seen up to this point that can be used in a lot of different ways, but one of the ways is with input elements, and you'll notice that if you look up at the top here, we have our jQuery selector function, and we're going to not just pass in input, which as we just covered in the previous section would only select the input tags, we want to actually select all input tags. Now you might say well what else is there. Well keep in mind that things like textarea and select don't have an input tag of course, that is the tag, textarea and select. Well the :input is different from the standard input in that :input selects all of those things, inputs, selects, textareas, buttons, even images, radio buttons and those types of things, checkboxes. So you can get the whole picture of things that can be on a form that are used to input data and click on buttons and things. Now, this will return all of those elements. So that means if you wanted to get radio buttons, you could just do this, you could say :input, but we want to filter it by the type of input and say the type is radio. Now that is going to target all radio buttons, but is that the most efficient way to do it based on what we've covered up to this point in the module. And the answer is no, it's not, because this is going to select all of the things that qualify as an input, input, select, textarea, button, all those types of things, and as a result, you're going to have to filter through more things to find where the type is radio versus if you just went in and took out the colon and put input, now we're only going to filter based on just the input tags. So we'd say grab the input where the type is radio. So it's a very subtle difference, but you've got to keep in mind that anytime you do a selector, it's going to grab this whole set, which in this case is the set of input, and then it's going to go ahead and filter based upon the type of radio. So you need to be careful of that so you can make your app as efficient and performant as possible. So let's take a look at how we can use input tags though, and it's very useful any time you just want to quickly iterate through different items in a form, including textareas and selects and things.

Input Selector Demo

So back in our webpage, I have my document ready, and down at the bottom I have a form, you'll see that right here, and in that form I have an input type equals text for a textbox, you'll notice the value is hardcoded to John, we're going to access that value in just a moment. We have an input, which is type text for Last Name. And we have a textarea. Now we could add a select to really kind of fill in the whole gamut of input controls, but the textarea is definitely different than a standard input. Now if we came in and let's say we came in and, I'll tell you what, let's do this first. In the area of the textbox, let's just say, or the textarea, let's go ahead and say TextArea right there. And so you'll notice when we go to design view we see TextArea and we see John, and then this one's empty. Well, what we can do to get to the values is first off we can use the :input selector. So let's go ahead and we'll do that, we'll do our jQuery selector syntax, we'll say input. And then that'll give us back a collection, but I want to show something to be aware of when you work with collections that we haven't seen up to this point, and a very useful thing to know about. So we'll say inputs equals that, and we've seen that a couple times. So that'll grab all the textareas, all the buttons, all the images buttons, everything that is associated with basically a form. Well, let's go ahead and say we want to write out the value of, this one will be the second input, there's another one that will be generated that's a hidden by the page, but we'll ignore that one. So you would think I could probably say inputs 1, which will be get to the first child, and then just go directly in and get to maybe the value of that. And the way you do that in jQuery is you say val. That literally ties us into, in the case of something like an input, it ties us into the value, or in the case of a textarea, it'll still allow you to get the text of that particular textarea. Very simple to use and something that takes a little getting used to, but works great because there's only one method you have to go through, one function to be able to get to your values of your input controls. But there's a little bit of a problem with this that I want to illustrate. So let's go ahead and fire this up in Chrome. And you'll notice you don't really see anything here, but I'm going to run off to the Chrome Developer Tools. So we'll come down into our developer tools here and you'll notice on the console tab of the developer tools, there's Uncaught TypeError, Object, HTMLInputElement has no method val. Well the first time you see that you might wonder what's going on here with our particular input in getting the value. Well the answer is, right now input is a little bit different than you might think. In fact we could come in and we could say alert, and we could do typeof inputs1, and let's go ahead and try to run that guy. And you'll see it's just an object at this point, so that doesn't really help a lot. So what's happening is if you want to get to val, that's a jQuery function. In order to get to jQuery functions, you have to have a jQuery wrapper around your object in order for that to work properly. So what we're going to do here is we're going to wrap the dollar selector type syntax around the object. In fact, you'll see you get IntelliSense for this. Notice here the IntelliSense says you can give it an element, an elementArray, a jQuery object, even well at this point, we actually have an element. So we have an HTML object if you will, an HTML element up in memory, which represents our inputs 1. We're going to wrap that with the jQuery object and that allows us to get to the jQuery functionality and to work with this as we'd expect. Now inputs 1 just so happens to be this input right here, this textbox, which has a value of John. So let's go ahead and go back up and run this now, and you'll notice we're able to get to John. So what we could also do then to get to all the values of our form input is we could iterate through those. Now I could do a for loop through this and then put a jQuery wrapper around each one to easily get to the value, but we're going to do the each that I showed a little bit earlier. So we're going to come in and say get me all the input, and once we get the input we want to say each of those, so we want to instantly loop through each of those, and that's going to call an anonymous function. And then in here, we can kind of break this out into steps and we can say elem equals this. Now you'll notice that this is a jQuery wrapper around an object called this. Well what is this at this point? Well, as we loop through each input item, it's going to pass it into this each function and that will represent in this case the textbox as an example. We're going to wrap that though and now we have a jQuery wrapper. So I'm just going to put a comment, This is wrapped in a jQuery wrapper at this point. Now we can come in and say alert elem., and notice we're able to get to the full set of IntelliSense for jQuery now because it knows there is a jQuery wrapper. Well, we can get the val because that allows us to get to the value of our particular item, and then we'll just again show that in an alert. So let's go ahead and fill out the form here, well run it actually because it's running in document ready. Okay, that's a hidden, there's John, there's an empty one, there's our TextArea, and there's our Submit button. So you can see those are the different inputs it looped through. Now likewise, let's come down really quickly and let's put in a select inside of this. So we'll change this to a select, add some options to it, we'll just call it that. And inside of our select let's just say we want to add maybe a couple states or something. So we'll say option, and then we can come in and add the standard values. And so we can come in and say value equals AZ. And we'll do one for California, so let me just copy that. Okay, now if you've done this before with standard JavaScript, you'll know that with selects you usually have to check the selected item that they're on to get to the value. Well with jQuery it's very consistent based upon what is selected. So let me go ahead and select the second one. So we'll say selected and I'm going to add an official true, or you could do selected equals selected if you wanted, but I like to finish mine, although we could just put selected. And now let's see what happens as we loop through the different items. And I just want to point out how nice this val function is when you're working with input controls. So let's go ahead and do that and there's our hidden, there's our John from our textbox, there's some emptiness there, TextArea, California, you'll see that it did pick it up directly, and then we have our Submit value for our button, you see California is selected of course down here. Now, that's really nice because you don't have to now check, are we on a textarea, are we on a textbox, are we on a button, are we on a select, it just automatically allows you to get to an input control and return the value of it just by calling that. Now we could also set the value as well. So we could go through and say val equals Foo, and what this'll do is actually loop through each item and try to set that value. Now with the select it's not going to be as useful, but now it's going to loop through it. And you'll notice that first off everything was set to Foo. Now like I said, with select we didn't have that option so we'd actually have to add that option in, but you can see also how easy it is to assign a value to any type of input control. So to kind of back up and kind rehash what we talked about, the :input selects all input controls. So again, that includes textareas, select, and all the inputs, button, image button, all those things. We can then use the each function in jQuery to loop through each item, and then that'll pass each item into this anonymous function and then we can grab it by putting a jQuery wrapper around this, around the object that was passed in. From there, we can either get the value of an input or set the value of an input by calling the val function. Very, very easy and once you get the hang of it, especially if you're used to doing it the old way, which is you write the custom JavaScript and you check the control you're on, and all that fun stuff, with jQuery they really simplify that and make it super easy to work with input controls. So that's an example of working with those. Now if we wanted to go in on the input control and do other things like select other types of input controls, you could use input, and as I mentioned earlier, you could come in and say type equals radio as an example, but that wouldn't be that efficient, it'd be more efficient if we had a radio to do that, then we're locking it down to just the input tags. Well, there's some other stuff that's available. So what I'm going to do to wrap up this part of it, let's run off to the jQuery.com documentation. And on the left, I'm going to run over to selectors and all the stuff we've been talking about is available in the selectors area, but I want to point out a couple things. If you want to find all checkboxes, all buttons, you can do :checkbox, :button, you want to find all the stuff that's checked, you could just do that. Moving on down, you'll see you could find images this way, hidden elements, and then of course we have the input as well that we just looked at. As you click on these different items and go in, they'll have a nice example for you. It shows you all the stuff you can find, textareas, buttons, selects, inputs, all that stuff with a code sample of kind of how to work with that type of stuff. So this is a really nice, I don't know if it's a trick, but it's a really nice technique to know about if you want to iterate through all the inputs in a given form, because keep in mind in a form, especially my form here, it's not just as simple as saying go find the form and then go find all the inputs as children. So we could do this, we could say go find the form and then go find all the inputs. Well again, that's only going to find the inputs of that form. That won't find the textareas, the selects, and stuff like that. So what's nice about the :input, it'll find everything you need within this particular screen. And we could even do this, if you want to lock it down, if you only had one form, then we could say hey, go find that form and then go ahead and find all the inputs within that form. So if we run it, you'll see we get the same exact thing. And then thinking through this, taking it a step further, how can we make it even more efficient to find all the inputs. Well let's go down to our form. Notice this form has an ID, not a very good one, but it does have an ID, form1. So to make this even more efficient, we could do that. We could say go find form1 and then go find all the inputs on that particular form and you'll see that again, we get the same exact results. So it makes it really nice to work with this in a very, very simple way. So that's an example of using the :input selector with jQuery.

Additional Selector Features

The final thing we're going to talk about pertaining to jQuery selectors is some additional flexibility when it comes to selecting DOM nodes. There's going to be times where you want to select some nodes based on the text they contain, or the attribute starts with or ends with or contains different text. So to this point we've only looked at how can you select tags based on a specific tag name, or how do you select based on ID or class name. There was a little bit of flexibility when it came to attribute selection, since you could say select based on this string, but you had to have that whole string, it wasn't starts with or ends with or those types of things. So let's take a look at a few different features of jQuery when it comes to selectors that you can apply to your programs. So the first one is the contains. Now contains allows you to search a given element for specific text. So in this example, you'll notice that we're going to search for a div, but we don't just want all divs, we want to select a div where that particular div contains the text Pluralsight somewhere in the text. It could start it, it could end it, it could be in the middle, it doesn't really matter. Now it is a case sensitive search, so be aware of that because I've beaten my head against the wall for a few minutes figuring out that particular point before, but in this example, we have a div, we have Expert Pluralsight Training /div. Well what that'll do is that would match this particular div. Now if it's the only div that has the word Pluralsight in it, then that's all that would be returned. The other divs would simply be ignored because of course they don't contain that text. So that can be very useful when you need it. Another one that's also very useful, I tend to work a lot with rows and columns, columnar data in line of business applications, and oftentimes you might want to highlight, for either highlighting to make it easier to read or for some other purpose, the different rows. Well you could do a lot of this with CSS, but what if you want conditional highlighting, based on this condition, highlight the even rows or the odd rows or whatever it may be. Well, fortunately jQuery provides two really useful features, :odd and :even that can be applied. Now all rows are 0-based, so you have 0, 2, 4, 6, 8, that would be your even rows, and you have 1, 3, 5, 7, 9 and up, and that would be your odd rows, and simply by saying go find tr and then doing :odd, that will return the set of odd rows or vice versa, you could do even. Then you could apply styles or change the HTML of those or whatever it may be to do what your application requires. So I'll show an example of this in the demo in just a moment. Very, very useful though and something that I find a good technique to know about. Now the last three things I'm going to talk about are going to be related to attributes, very cool stuff for selecting text, but there's one final one I want to talk about with regard to elements before we move into attributes, and that is first-child, last-child, and you can even find children in the middle. So there may be times when you want to find some spans, but you only want to highlight maybe the first child, if it's a span, throughout your entire page for whatever reason. Well you'll notice in this particular example we're using yet another selector that we haven't seen to this point where we're going to say go find all spans, but we only want to find, we want to filter it down to the first child. And so if you look into this down here to the bottom, you'll see that we have two divs and each div has two spans, but what we want to do is just select that first span within each parent. Now it doesn't have to be just that, it could be a td tag, maybe you have a span inside of several columns and you want to highlight those as well. What matters though is that we're automatically allowing jQuery to filter based upon some criteria such as first-child, last-child or there's an eq, an equals to give it a specific position of the children, and I'll show some of this as we walk through the demos here. It makes it very easy to work with though to find your children without having to write hardly any JavaScript code as you can see. So the last three things I want to show you are related to attribute selectors. So we've already talked about attribute selectors, but everything we talked about, they had to contain a very specific string. So what I'm going to show you here is that you can select attributes based on attribute where the attribute starts with or ends with or contains a specific value. So here's the syntax, you'll do that kind of where clause as I refer to it as, where the attribute starts with. Now this is the carrot symbol, and if you've done regular expressions before, this'll look pretty familiar. If you haven't, just use the carrot symbol. So the carrot symbol followed by an equals allows you to do this starts with. So the example shown here is we want to go find all input where the value of the input starts with Events. And so if we had these, in this case three buttons, then because these all three start with Events, this would automatically allow us to go into those and work with them. It makes it really easy to do what you need to do there. Likewise, we can go in and say hey, I want to find where it ends with. So instead of using the carrot, you can just go in and do the dollar. Again, that comes from regular expressions. So now we'll say go find me an input where the value attribute ends with the word Events. Coming down to here, now it's nice because the World, National, and Local might dynamically be generated for us, maybe you're using PHP or Pearl or whatever it may be to generate the HTML and you don't have to worry about what it starts with, you just know it contains, or ends with in this case, the word Events, very simple. And then I just alluded to the final one, which is you can also go in and say hey, find me everything that contains the text Events. And to do that, you simply use a star. And so very similar, you say attribute value, or name of the attribute, star equals the value. So in this case we want to find all the input where the value attribute contains the text event. So you'll notice in this example we have World Events, National Events, Local Events, it would find all three of those and go ahead and return it. So that's an example of using contains outside of attributes with like divs. We looked at how you can do first-child, last-child, those types of things, and we also saw the odd and even functionality that's really nice when you have a set of children, and I showed it with trs, but keep in mind that could be used for all kinds of things, lis in the case of unordered list, and stuff like that. And then finally we talked about three different flexibility features when it comes to attribute selectors for starts with, ends with, and contains. So let's take a look at an example of using some of these in a demo application.

Additional Selector Features Demo

So let's take a look at some of the additional functionality available with contains and the other things we just talked about. So the first thing we'll do is you'll notice we have a div here and it has this text, This is my div text. So let's go ahead and see how easy it is to use contains. So I'm going to come on in and we're going to say let's go find a div that contains the word, or words even, let's do my div text, or let's just do my div right in the middle. Alright, so what that'll do is automatically go in, find any div that contains that text, and then return it. Now what I'm going to do to show that it worked is let's just do an alert around this and we'll pop up the inner HTML, which in jQuery as I showed earlier, you can simply do that to make it run. Now one thing to keep in mind, it is case sensitive. So, I know sometimes I've done an uppercase C not thinking about it and it is a lowercase c and it does matter, so be aware of that, it's case sensitive. And so, well that's actually not what I expected, so let's go back to that and we want div. Oh, and what's wrong, we should have put a colon, didn't mean to put a dot. So let's go run it again and there we go, this is my div text. So that works out perfect. So that's how you can use the contains. Now one of my favorite ones that you can do, and this one's definitely a visual feature that's nice, is let's say that all our rows here, we'd like to highlight those, maybe make them greenish for green reports or something going way back to the old days, and highlight it based on some color, maybe the user clicks a button or something that triggers this, but we'll just do it when it loads. So what I can do is I can come in and we can do a selector for tr, which that would just get all the trs, so let's go ahead and do that to start. And we'll do CSS like I showed earlier, and let's do green. So if we run that as is, that's going to go in and you'll see that all my rows are now green, it doesn't look too nice actually. Well, what if I want just odd or even? Well as I just showed you, you can come in and say odd and that'll come in and just highlight the odd rows. Remember it starts with 0, so 1, 3, and so on and so forth going down. You can see it work down here as well. And it just makes it really nice to highlight whatever you want to highlight with whatever color, don't pick the colors I'm using, but it makes it super easy to work with. Likewise I can do even and that'll automatically go in and highlight the even rows, 0, 1, you can see those types of things going on. Very useful and I've used that quite a bit actually to either make things more readable, or maybe when they click a button they can highlight it, maybe by default nothing's highlighted, but at the top of your data, you might have a way they can highlight the odd or even rows to make it more readable for them. So that's nice. Now we can also come in and leverage the first-child that I showed as well. And that'll come in and just highlight the first-child of each table. Now this one's highlighted because there's actually a nested table right there, so that is correct. You can see it's highlighting the first child of each table there, or tr. And then we could even go in and say which table, so let me, instead of doing first-child, let's go back to say odd, but you'll notice it was doing both tables. Well, let's come to this table right here. Let's say it had an ID and we'll just call it DataTable. Well, using the standard ID selector, we can first tell it select DataTable, then go find tr, but I only want to find the odd trs, then highlight with, let's go back to green, a little easier to see. And now the top table should be highlighted, and that's it. You'll notice the bottom table is left alone because we did a filter, and that's a very efficient selector. Any time you can start something off with either a specific tag or even faster than that, a specific ID, then that's a great way to go and work with things. Okay, so moving on, we also talked about different attribute selectors, adding a little more flexibility, starts with, ends with, contains, that type of thing. Let's go find a div where the title attribute, instead of doing equals, let's do ends with, which is a dollar, Title. And it is case sensitive. We'll end our bracket, which is kind of like ending our where clause and then we'll just write out the HTML because that makes it easy to see. So we'll do an alert on that. Alright, and let's run it and you'll see This is my div text, which is correct. If we come back to that div, there's the div, there's the title. Now if we change that to a starts with, which is the carrot symbol, then obviously we're going to get entirely different results here because I don't have one that I know of, and you can see we don't, it's null, so we don't have any object to be able to write out, there's no HTML there. Or, let's say that we had something like this and title is just hung in there somewhere, then we can do the star, which is the contains. And we'll do star equals, run that, and you can see it did match up on the div. Very nice. Now, to this point I've kind of been using alert a lot because really other than showing the HTML function, the CSX function, and the vowel function, I've tried to focus just on selectors. We'll get into some of these other functions in later modules, I want to keep it focused. But let's do this. If we find a div that has a particular title containing the text title, let's go ahead and set the HTML to a different value. Now I'm just going to be put Updated Div Value due to Title, something like that. So you can certainly go in and update your different code on the fly. I've only used HTML to this point to write it out, and let's face it, you're not going to be doing alerts a lot popping up HTML, and I realize that, I don't do it myself. But let's go ahead and run this now and you'll see that there we go, it updated that div live and updated the HTML. And this is nice because if you've worked with inner text versus inner HTML in something like Internet Explorer, which supports both of those, and you also have had to target Chrome or Firefox, you'll know that inner text doesn't work so well across those. In fact, it's only supported out of the box in Internet Explorer. Well, the nice thing about jQuery is that you have one function to know about, so once your selector does its match, then you can go in and either get the HTML or you could go in and update the HTML. And in some of the future modules in this course, I'll show you other ways you can manipulate HTML such as appending notes, removing notes, and things of that nature. So that's an example of some of the extended selector features that are available. And please do keep in mind, there's a lot more. If you go to the jQuery documentation, there's even other symbols and other combinations of ways you can do things, but this will give you a solid foundation to help you get started writing jQuery selectors.

Summary

So that's a wrap on the key selector features of jQuery that you need to know about to get started. Selectors are where it starts. You can't really do anything useful with jQuery unless you get good at selectors. So as a quick recap, we've talked about how to find tags, simply pass in the tag name into the dollar function method. We've also talked about different ID selectors, class selectors and the different characters you'd use such as the hash or the pound. And we've also talked about the dot for class selection. Some other characters we just talked about in the last section of this module were the carrot, the dollar, and the star for starts with, ends with, and contains. And those are very useful in cases where you need to search for specific string values and starts with or ends with. We've also talked about how you could find input elements. We've talked about how you can go in and then filter on those and how that returns not only input tags, but how it returns things such as text areas and selects, and I showed you during that demo how you could use the vowel function in jQuery to easily get the values out of a form tag or element that you want to get to. So as I mentioned, mastering selectors in jQuery really and truly is key to being good at jQuery. You have to get good at selectors and practice these in order to be able to use it and work with it. Now fortunately, there are some sites out there that'll make it easy for you to practice some of this without having to do your own HTML like I've been doing to this point, and one of those sites I'd like to show you now, it's super easy to work with and makes it really nice to test out your jQuery selector knowledge. So the URL for this site is codylindley.com/jqueryselectors, and it's what I always show in our jQuery classes that makes it really easy to work with things. So, what it allows you to do is you can come up here in the top area and you can actually input expressions and type your own. So this would say go find an li, which would be an unordered list, you can see we have some here, and go find nth-child 2. So if we hit toggle, it'll actually go in and you'll notice in this case it starts with 1, so we come in 1, 2, and then we also add a second level, there's 1, 2 of that one, and you can toggle those on and off to make it really easy to know if your expression is working. If I just wanted to go find all divs, then I could just toggle all divs that are in my particular page. Maybe I want to find the first div. So another way we can do that is as shown in the earlier demo, we could say equals 0, come toggle, and there it is, or you could do the nth-child as they just showed earlier. You can also come down and over here is all your HTML that you can work with, they have all kinds of stuff down here, but as you scroll down here, we can now find all code tags, so we can toggle. There's some code tags in there. And then you can even go to documentation directly about using element selectors. So they make it really convenient to be able to work with those. Here's an example of ID selectors. Here's class selectors that we talked about. This finds everything, the star is, in this case not super useful because it highlights every single thing. Here's an example of highlighting a tag, an ID selector, and a class selector and highlight those. So, I'll let you play with it, but it's something that's very, very useful as you want to experiment with the different characters and things you can work with in jQuery, and they even cover some of the ones that you won't use quite as much, some of the ones down here as an example, so I'll let you play with that. So definitely get good at working with selectors, they're very, very key. And that's a wrap on this particular module.

Interacting with the DOM

Introduction

Hi, I'm Dan Wahlin with Pluralsight. And in this module, we're going to talk about how you can interact with the document object module using jQuery APIs. Now up to this point in the course, you've learned how we can use selectors to find nodes in the DOM and we've talked about ID selectors and tag selectors, attributes, classes, and stuff like that, and I've even shown maybe 2 or 3 functions that you can use as you interact with selectors and get those notes back, but we're going to take it up a notch in this module and talk about some of the other items, and in some of those that I did cover, we're going to go into a little bit more detail. So this is a really good module if you want to dynamically influence the DOM, you want to add notes or remove notes, change styles, switch out classes, all those types of things. So the agenda that we're going to go over here is first off I'm going to start with the each function. Now I've already detailed this a little bit and demoed it in the previous module with selectors. However, there's a little bit more you can do with it, and I want to point that out real quick. Very, very useful function though. From there, we're going to talk about how to modify properties and attributes, and what the difference is there, and it's definitely something you need to know the difference because there's a different syntax you use, a different way you access the object whether you're going directly to the raw object or if you're going in through a jQuery object, so we'll talk about that. Then we'll get into what I find extremely useful. I do a lot of line of business applications where we'll make calls back to the server with AJAX to get our data as JSON, and then we'll update our page, so we might add notes, we might remove notes, it depends on what the user did of course. So we'll talk about the different add and remove APIs that are available and it's very, very simple. If you've used things like document.createElement, you'll know it's quite a bit of code actually if you want to add a lot of different HTML nodes or elements into the DOM. Well, with jQuery they've really simplified it and made it super easy to use, so I'll explain that when we get there. And then finally, we're going to wrap up this module by talking about styles and then we'll talk about classes. So you've already seen the CSS function a little bit in the Selectors module, but we'll go into a little more detail there, and then we'll go into some really good details if you're a big fan of CSS, and I hope you are, then we'll go into how we can add and remove classes, toggle classes, remove specific classes, all classes, there's all kinds of stuff that's really fun to do to influence how your page looks within the application. So let's go ahead and jump into the each function and talk about how you can use that in your jQuery apps.

Iterating Through Nodes

Let's talk about jQuery's each function in a little more detail. Now I've already covered it briefly and demoed it with the selectors earlier in the course, but if you didn't watch that part or if you want to get more details, let's talk about those. So the each function actually can pass you two parameters. Now for the most part, I generally don't even need these parameters, but it really depends on what you're doing. Sometimes you might need, for instance, the index. So it'll pass you the item it's on as you loop through, and then it'll also pass you the actual element that it's looping through. So, you kind of get the equivalent of what a for loop would do in traditional, just regular JavaScript where you might have i, i is less than the length, i++ type thing. The index will allow you to get that index, which is really useful because then you don't have to keep some type of internal counter going. Well, I don't use element a lot, but I'm going to show you that here at the bottom in just a moment. So, I do use index from time to time. So what we're doing in this example is we're going in and we're selecting all divs, we're going to iterate through those, and as we do that, we're going to tell jQuery call into this anonymous function. Now the function only takes a single parameter, you can see that's just the index, which is going to be basically an integer, and then you can simply use that integer directly. Now how do you get to the actual div though that's being looped through at this time? Well, the easiest way to do that is to wrap the this keyword because this will represent the raw DOM object that you're looping through, but we want to wrap that with the jQuery selector. And so you'll notice we have the typical jQuery function here where we start with the dollar, we wrap it with parentheses, and then we have this. So what that'll do is take the raw object, wrap it in a jQuery wrapper, I explained this a little bit in the Selectors module, but the reason we do that is we want to get access to the jQuery functions, jQuery API. If we just use this dot, we're not going to have access to methods such as text. And so in this example what we're doing is just grabbing all the text out of each div and basically appending that to the index. So it would be the number, equals, and then the text for that particular div. So very, very simple, very easy to work with, and it actually saves you some code over a for loop. Now you can certainly use a for loop as well, and that's just personal preference whatever one you want to go. Obviously anytime you use the raw JavaScript API performance wise you might save a couple milliseconds, it just depends on what you're doing though. Alright, now the second parameter that can also pass is the actual element being looped through. Now I quite honestly never use this one, I prefer to do this, it's four characters, easy to type, but let's say we had down here the same function as up above where we have the each, the function, and the index except we're going to name the second parameter elem. What we can do is wrap that just like we did this. So it's the same thing, elem equals this, and that's why I prefer to just use this with the each, it just makes it a lot easier to work I think. So now that we've talked about the different parameters that can be passed, let's take a look at a demo on how you can use the each function.

Iterating Through Nodes Demo

The each function provides a really easy way to loop through a set of nodes that you might have selected with the jQuery selector, so let's take a look at how we can do that. I have a couple divs here and these particular divs have some styles, you can see we have a red div and a blue div, but I'm just going to loop through all divs in this particular document that may have that particular selector, either blue or red. So to do that, we're going to come up into our ready, I'm going to strip out the code we had, and when the ready is called and the DOM is ready to go, then we're going to go ahead and do the selector. So we're going to go ahead and say I want to select the div.BlueDiv and div.RedDiv, and then we could say each. Now you'll notice as I do each, we can automatically, as I showed earlier in the slides, we can go in and do the function, and so the function can actually take two parameters. We could pass the index and we can pass the element that you might be wanting to grab. Now I mentioned earlier that I just prefer to use the this keyword, but you can do whatever you want there. So let's go ahead and end this. Alright, now from here we might want to go in and just pop up some information about that particular item. Now what I'm going to do though, instead of showing an alert, let's go ahead and add an ID here called OutputDiv, and that's just going to have nothing inside of it, but what I'm going to do is go in and update OutputDiv with the text inside of these two divs that we're going to select. So if we go down to those, you'll see that we just have Blue div and we have Red div, so not a whole lot going on there. But I want to show that when we update the HTML of this, there's different ways to do this. So, first thing I'm going to do is I could come in in the loop and say let's go find the ID selector called OutputDiv. Now that would work, and that would work great, except keep in mind, it's going to do that every single time that we loop through. It may not be what we want of course, and it's definitely not performant, so it would be better to cache this outside of the loop. And that's just one of those things when you get into jQuery, you need to be careful of that because you might be in a loop and you try to do some different types of selectors and if for instance you did a selector like this, just div for some reason inside of a loop, it's going to scan the entire document module, document object module, every single time you go through that loop, obviously not a good thing. So what you can do instead is we can cache that up top. Highly recommended that you do that. So what we'll do here is I'm just going to grab the text and we'll update the HTML. Now there's another tip you can do with this as well, and so I'm going to go ahead and do it one way and then I'll wrap up by showing a different way. So let's go do output.html, and then we're going to set the value of the HTML to whatever output.html is. We want to grab what the current value is plus let's go ahead and add in a br, plus we want to grab index plus, and then we'll grab the this to grab the value that we're currently iterating through. So we'll say this, and then we'll do text, that'll just grab the text value of this particular item. So we're going to grab whatever HTML had for the output, whatever the current value is, which the first time through will be nothing of course, then we're going to append a br, then we're going to add the index, and then we'll add the actual text. And just to make it a little bit more readable, let's go ahead and add in one item there. So to review, we're going to go in with a selector, we're going to select all BlueDiv divs and all RedDiv divs, and of course these are the CSS selectors, but we're going to be specific, we're then going to do an each on the set that's returned. Now that should only return these two items that you see right there. We should get that div and we should get that div. Once those divs are returned, we're going to loop through each one, that's going to pass us the index of each one. We're then going to write to an output div, but we're going to cache it, we're going to grab it first before we do the loop, update its HTML, but we don't want to lose what's already in there, so we're going to grab the current HTML, do a br, and then write out the index on the text. So barring any typos there, let's go ahead and run this, and you'll see it did work, we get 0 Blue div and 1 Red div, so not real impressive output, but it did work properly. Now, that's still a little bit messy and we're actually having to update the DOM of the HTML, which is ultimately in the DOM of the output every time. This is also not good for performance in general. What would generally be better is we might want to just say html, create an HTML variable here, and we can come in and say html, we could just do +=, a br, the index, and then this.text, and we'll get rid of that last bracket there, or parenthesis. Then when we're done, we can actually grab the output. Now we don't need it until the end here, so we're going to come and now we'll say output.html is, and then we'll give it our HTML. Now this is going to be even better than the previous one, because the previous one, although we had cached the OutputDiv and we weren't grabbing that selector every single time through the loop, that was good, but we were updating the HTML of that particular object, which means it has to touch the DOM every time it loops through. Now we're only looping through two things, but imagine you had a table and you're modifying the trs or tds or whatever it may be and you have a lot of those. That's definitely not the best performance. So, that's why we're going to go ahead and build up our string first as we look through, grab that index, grab the text, then once the string is built up, we'll go ahead and assign that to the HTML property. Now that gives us the best of both worlds. We've cached the object and we don't update the DOM until we're ready to update the DOM, so it's one shot, much faster. So let's go ahead and run that, we should get the same output, and you can see we do. I'm trying to think where the undefined come from here. Oh, it's because it was undefined at first, so let's set that to actually empty strings, run it again, and there we go. So we get the 0 and the red. Alright, so that provides an excellent way to easily iterate through notes. I also showed in the slides you certainly could pass the element that's being looped through here, but then you'd still have to wrap it. This ultimately would give us the same exact result, you can see there. But I don't know, my take on it is why put extra text there and there, have to match it, when you could just do this. So 99% of the people you see out there doing this, including myself, we'll just go ahead and do it this way. So that's an example of the each, and a couple tips and tricks for you as well when you're doing the each. So my advice is be really, really careful on what you put inside of your each block in the each anonymous function here because keep in mind, it's going to call that every time, so you want to keep that as minimal as possible.

Modifying Properties and Attributes

jQuery provides an easy way to interact with DOM objects directly, or using jQuery object wrappers, and we're going to talk about both of those. We're first going to talk about how we can modify DOM object properties directly, then we're going to go into how we can take a jQuery wrapper, a jQuery object, and call into the jQuery APIs to do other things. So two different scenarios, but they ultimately end up doing a very similar purpose, similar task. So the first thing we're going to talk about is we just covered the each function in the previous section and I talked about how as we loop through using the each, that function that gets called will pass in the index, and then we also get passed the raw object. Now you can get to it if you don't pass it as a parameter using this. Now if you just use the keyword this, then it represents the raw DOM object, and then we can get to the direct properties of that object just by using the dot syntax, so .title in this example. And then we're going to assign the title to a value of My Index equals, and then the index that was passed in. So because this represents the raw DOM object, we don't have access to the jQuery API at this point. Now we could, as I showed earlier in the selector areas of the course, we could go in and put dollar parentheses around this, and that would allow us to go ahead and use that as a jQuery object, but then you wouldn't have direct access to the properties like this because now it would think it's not a raw DOM object, it's actually a jQuery object. So the big thing to know, and here's kinda how I think of it is if you're using a tool that gives you IntelliSense or code help, Aptana, their tools, or Visual Studio or others, then if you do this dot and you don't see any of the jQuery API functions showing up, you're probably on the raw DOM object, and that's why you're not seeing those. So if you want to convert that into a jQuery object so you can do something else with it, then you'd have to wrap it with the dollar parentheses and use a selector type syntax. So that's an example of how we can modify properties using just the raw DOM object itself. Now we can also use jQuery though to go in and modify what we're going to call attributes of the object directly as well. Now the way that works is once you've found one or more objects, and this works not only on a single one like this ID selector, but even on a collection, which I'll show you in a moment, then you could come in and in this example we're going to say let's go find CustomerDiv, and we want to grab the title attribute of that particular element. So if you think of the HTML, the HTML might have a title attribute on the div and we want to go ahead and grab that. Now that's very similar in this case to the title property, but dollar CustomerDiv gives us a jQuery object. That's why we can then do .attr title. So a lot of people call this the attr function, very, very useful, something I use a lot in my jQuery apps because oftentimes I want to manipulate those attributes based on what's occurring with the end user. So in this example we'll simply select that div, and then we're going to grab the title and we're going to store it in a value. Now of course there's a lot of times where you just don't want to grab it and select it, you actually want to update it. So the attr function also takes the attribute name and the value if you'd like to set a value. Now this one provides kind of an interesting thing that jQuery does. We've talked about the each function and how I could say dollar img, do a tag selector and then do the each and loop through each one, then we could change the title as I just showed earlier using the raw DOM object. Well, with jQuery, once you've selected a collection or a set of DOM objects, now it's going to internally have a wrapper around those. And so if you do .attr title My Image Title, it would actually update the title attribute of each image within your page. Now obviously you may not want to do that depending on what it is you have going, but I find it pretty cool that it does that because I don't even have to loop through it, jQuery just magically, based on the fact that this might return more than one image, will automatically update the title attribute of each of those images, I don't even have to do an each loop, that's pretty cool. Now it also gets even better though. What if we don't want to just update title while we're looping through, while jQuery I should say, loops through each of the images, we might want to update title, maybe the style or something else. Well we can do that as well using what's called JSON, JavaScript Object Notation. Now I'm going to talk about that in just a sec, but let me show you a simple example to start. So we have the similar scenario as the previous one where we have a selector on image, but we're not going to change just a single attribute, jQuery supports doing what's called a map, and a map is basically just a JSON object, it's an object with properties. Now those properties need to map up, or match up, either way you want to look at it, to the images properties, or attributes in this case. So we already talked about title. Now in this case, title:, this is a separator between the value of the property and, well I'm sorry, the name of the property and the value of the property. Then notice we put comma, style:, and then the value of the style that we want to set here. Now, that is all wrapped with these brackets, so the start of this little object, this JSON object, is here, and it ends down here with this bracket, and it has two properties. So what this'll do is jQuery will now find all the images, it'll go in, iterate through those for you, and then set the title and the style to these appropriate values. Now style can be said, as I showed in the selector, through the CSS property, and that's normally how I would do it, but this is just demonstrating that you can assign multiple attributes in one shot, very good for performance because I don't have to iterate through all of the images twice just because I want to set two, three, four, whatever it is attributes I want to set, very useful. Alright, let's talk about JSON just a little bit more and show you another example. This is going to come up again in AJAX, so if you've never done this before and never seen JSON, you might say hey, wait a sec, what's this JSON thing you're talking about. Well, JSON is just a way to delimit objects and to find properties and values. It's been around actually since the late 90s in almost all the major browsers. It's a feature of the JavaScript language, it really wasn't leveraged until several years ago, but it wasn't leveraged until fairly recently. And, this is something that you just saw an example of that with the title and the style attribute, but here's just more of a data JSON object. The start of the object is here and it ends here. Now a JSON object doesn't have a name typically, it's an anonymous object. Now you'll see in a moment that depends, but FirstName is the property name, John in the value, comma, LastName Doe. And you'll notice, so properties and values again are separated by a colon, a comma separates the different properties from each other. So if you've done XML before, you would normally wrap a tag around John, you'd have a start tag and an end tag around John. Well JSON is just a more compact way of kind of describing data, it's very, very easy to work with actually, it's something I use a lot, especially when you get into AJAX functionality. So, you'll notice we have FirstName and LastName. Well we also have a nested JSON object, and this case the property is named Address, but Address isn't just a simple value, it's actually a nested JSON object. So you'll notice that it starts with the red bracket here and then we have the closing down here. And then that object has its own set of properties. So the object officially doesn't have a name, but that object is assigned to a property of this parent object, and that property is called address. So that's kind of how it works, you can nest them, and you can even do a raise with JSON. You do that by doing the square brackets, which isn't shown here, but you can definitely do it when you do AJAX to pass back multiple customers as in this example. So that's what JSON is in a nutshell, and going back, that's exactly what's going on right here with the attr function, we're simply passing a JSON object, it has two properties, and then once we select that image, it'll update both of those attributes on that object for us. It makes it very, very easy to work with. So, let's take a look at using raw property modifications versus JSON objects and jQuery objects to use the attr function.

Modifying Properties and Attributes Demo

Now that you've seen the direct properties that can be set and the attr function that jQuery provides, let's see it for real in an example here. So if we come on back down to the divs I was working with in the previous demonstration with the each, we have two divs, and let's say that we'd like to first go in and set the title of these divs. Well, if we're just looping through those in general and doing something like the following, then we're going to get the raw object back, and when I say raw, I mean the raw DOM object. So if we do this.title equals Some title, this is what's passed when the function is called as it loops through, in this case we're going to get back two items that are going to be passed in. And so we can just take that raw DOM object and we can update it directly. So if we run that and then mouse over the divs, we should see title Some title applied. So let's go ahead and run that first, and then we'll come back and show how to use the attr function. So let's mouse over it and you'll see some title shows up as you would expect. So you see we're able to actually update the title directly. Now, we could also go in as we loop through and we can get to that title as well of course, and obviously it's going to return the same value because we just set it, but there we go, Some title and Some title. And that would be the raw DOM object way to do this. And you can change all kinds of properties using this if you'd like. Now you can also though come in and wrap this into a jQuery object. So we could say this., and now notice in IntelliSense here in Visual Studio, we have full access to the jQuery API, and so therefore we can come in and do some of the cool stuff, there's our attr function, the attr function. So we can come in and we can override, let's go ahead and try to override that with Some title 2, just so we can distinguish, but we need to tell it which property of course, we can't just give it the value. So we're going to say title, Some title 2, and then we'll end it, and then let's run it and see what we get here. So we'll mouse over those and you'll notice it now overrode it. So what I wanted to demonstrate here, and the reason I kind of did the same thing twice, is ultimately this is updating the same property that this is updating, it's just that because this is the raw DOM object, we can go directly to that property versus if we come into the jQuery object, now we have to go and get into the jQuery API to do it. Now a lot of the time I just prefer to use the attr function, but to each their own. It depends on who you talk to there, but that's kind of how that works. So we'll go ahead and leave those there. Now moving on down, we can also go in and let's say that we want to change, right now there's a background-color and there's no font though, you can see the styles of each of those divs are defined here, but let's say we want to go in and change those to something else, but I don't want to use an each. Say we want to change the font on them. Well, we can go in and the best way to do this would be to use CSS to find it, then we could come in and say font-size, let's go something kind of big here, we'll just do a 20pt real quick. And the CSS property will update the style and you can see that worked, we now have 20 point fonts. Well, you can change anything you want on these though. Instead of doing it this way, I'm going to assume we want to change several properties, or attributes in this case. So I'm going to use the attr, we're going to wrap that down to there, and then inside of it I'm going to pass what jQuery calls a map. And this is my JSON object. In fact, let me go ahead and just wrap this so you can see it down in here a little bit easier. And now we can come in and we can simply say the name of the property we want to change, so we'll go ahead and put a colon to separate it, and then we'll put the value. So I'm going to put Some title 3 since up here we'd have Some title, but it'll be overridden by Some title 2, now we have Some title 3, and now I can't use CSS:, that's a jQuery function, but I can do style and we could do font-size, let me do it all out here, and we'll do, let's do like 14pt. And then, we could even chang the background-color if we wanted, background-color is yellow to distinguish easily. And we could just keep putting commas and we can change all kinds of things on these divs if we wanted. So that's an example of we're able to actually find the target divs in this case and automatically apply two attribute values, or two properties really, to those, and then pass those into this attr function and work with it. So let's go ahead and run it and we should see a 14pt font, we should see yellow, and we should see Some title 3 passed into that particular value. And there we go, not real easy to read, but we'll mouse over it, there's Some title 3, and we could even come in and on that style say color Black, run it again, and we could do this. Now, I mentioned earlier that you could do the style this way, and absolutely that's fine, but let me show you something called chaining that ties into this. Now that we've talked about some of the APIs a little bit more, jQuery has a very cool feature called chaining that could be used. And so to be honest, what I just did is going to have less code involved, but let's say we had multiple properties we want to add, and then as I'm looping through each of these, I don't want to just change the attributes, I also want to change their style, but I would prefer to do it, we're going to say, through the CSS function. So what I typically like to do here is I'll wrap these and I like to wrap each one on its own line if it gets too long, because otherwise if it gets strung out all the way over to here, it gets a little bit hard to read and so I'll come in and do whatever I need to do to make it look nicer, but this is chaining. Now what attr does is it's going to still return the modified set of objects that were selected with this tag class selector. So literally I'm going to tag on, in essence I'm putting this function after this function, right there, then we're going to call another function. And we can do CSS and we could say background-color, yellow. And then, if I want to do CSS again, and again I'm going to wrap it to keep it a little bit easier to read, then we could just keep chaining these. So we could say color, black, .css, and we could say font-size, 20pt, and we could just keep going to town. Now I'm done. So we just chained one, two, three, four functions together. Now you may look at this and say I prefer setting the style up here, and that's fine, whichever way you prefer is good with me, but the bottom line here is that jQuery allows you to chain different functions together to, instead of having to iterate through these and just change the attributes and then iterate through them again and change this, and then iterate through them again and change this, that would get expensive. Well this allows me to do it all in one shot. So it's only selecting these two divs from the DOM once, then it's allowing me to act upon those particular nodes, so if we run it, we won't see much of a change here, but you can see we get to Some title 3, the font's been applied, the background has been applied, yellow, and the color of the font has been applied to black here in this example. So that's object chaining combined with using the attr function, and like I said, you can certainly, if you prefer to do style, that to me is actually a little bit easier than doing all this code, but it depends on what you're doing. Some of these values might dynamically come in and who knows what you're doing in your app. So that's an example of setting raw properties using the attr function, using the attr function with a map, and doing jQuery chaining in this particular demo, so some really, really useful stuff you can do in your jQuery applications.

Adding and Removing Nodes

Changing object properties and attributes is important, but there's going to be times when you want to add or remove specific nodes from the document object module. So in this section of the module, we're going to talk about how you can do that with jQuery, and we're going to talk about four main methods that you see here. So there's some other things you can do and different techniques, in fact I even showed in the previous example of how you could use the HTML function to actually add in stuff on the fly, but the kind of official way to work with the DOM is the append, appendTo, prepend, and prependTo. Now I'm only a big fan of two of these, which is append and prepend, but I'm going to show you how all four can be used, although I don't really use two of those a whole lot. Now you can also remove nodes from a given element by using the remove function as well. So very simple to do, you can either go to that object directly or you can remove groups of objects doing a more generic selector to find those different nodes in the DOM. So append will append a child to the bottom of the existing children versus prepend will put it at the beginning of the children. And like I said, we're going to look at all four of those and see how you can use them. So let's talk about appending two nodes first. There's oftentimes the scenario where you might have a container like a div, and inside of that div you might have some different children you want to add as the data changes, and this is very useful as you get more into more robust applications with AJAX integration, where again you go out to the server, you get back some data, and then you want to take that data and update it up into the document object model, the DOM. Well there's two main ways you can append nodes to a particular item. Now like I said, one of these I don't really use a whole lot, the other I do. The first one is the appendTo, and it works great. I don't find myself using this one as much, but I know other people that love it, so it just depends. But what you do is you simply give it the string of HTML that you'd like to add up into the DOM. Now you'll notice I'm not having to call document.createElement anywhere in here. It really saves you a lot of time compared to the normal JavaScript way of doing this where you have to go through the document object. Now obviously behind the scenes jQuery is interacting with the document object, but it just saves us that time. So in this first one, we have a span with office in parentheses and the closing span tag, and we want to append that to all matches, all nodes, that have the officePhone class on them. So this would go find everything that has officePhone on it as a class, and it would append this span at the bottom of the child collection. Now I actually prefer to do the selector first, and again it just depends on the scenario, it's not that one's better than the other, but I find myself using the bottom one, the append a little bit more. So I'll come in and say let's go find all the nodes that have the officePhone class and I would like to append this particular span, which is obviously the same thing, at the bottom of the child collection. Now, that's just awesome. If you've been doing this the old fashioned way, you could see how much time this can save you because you can add in whatever HTML you want in here, you can build up that string, and as I showed in the each demo earlier in this module, you would want to build up your string first before you call the append method because you want to interact with the DOM the least amount possible, keep your interaction down to a minimum. So that's one way that you can append nodes, or two ways I should say, you can append nodes. Now you might not want to put the nodes down at the bottom though, you might on the append want to put them up top of the child collection. And that's where the prepend comes into play. So you can prepend using prependTo, or you can use prepend, and it's the exact same functionality as the append that we just covered. So you take that string, and then you can do prependTo and you can find what it is you want to grab, or you could first do my preference, which is let's find what we want to append, or prepend in this case, call the prepend method, and then give it that particular string. So regardless of which way you choose on the append or the prepend, very, very easy, super, super productive when it comes to really dynamic apps where you have to get that data up into your page, the DOM. Now you can also go in and you can do a really cool feature called wrap. And this is useful if you have maybe a set of objects or just one object maybe, but you want to wrap all those with a parent. Where this comes into play oftentimes is you might want to highlight a set of items, and maybe you want to change the CSS class or styles that are being used, but you don't want to individually do it on the items, you want the parent for instance to drive say the background-color, or something like that, or the border. So in this example, if we had a div with class state, and then we have this text inside of it, if we want to wrap that with another div that has a class called US\_State. So notice that we have a div here, class is US\_State, but we just close it, shortcut close. And you could do it either way, the long tag way or the shortcut. Well, what this does is this will actually wrap this div around this div. So what makes it really easy to apply this US\_State class as maybe the background or whatever you're doing as far as the CSS goes to change this, so this is what you'd end up with down here, down in the bottom. So we'd go find all the states, we would then wrap all those with this particular div, and that'll just add it as a wrapper just like you see. So again, very, very performant, it's a super productive feature if you're working with a lot of DOM manipulation and you need that particular functionality. So in addition to appending, prepending, and wrapping nodes, you can also go in and remove nodes as well. So we can go in and use a remove method, very simple to do. So the remove method is actually very functional. You can remove specific items or you can even just remove more, I guess narrow-focused type items. So what this will do is the remove will come in and remove all matched elements. So what I meant by more specific or least specific is in this example we're being kind of general, we have a phone class and a location class and we have that comma so that we can define both those sets of nodes. So who knows, that might return 50 nodes potentially. Well, if you want to remove every node that has those classes associated with them, one or the other, then you just call remove, and that's it. It's very simple versus with the document object model, you first have to find the node, then you have to find the parent, then you have to remove it off of that parent, and it's actually quite a bit more tricky. This makes it super, super easy, and of course, if you only have one specific node you want to remove, then you could just use an ID selector with the hash or the pound, and remove that node very, very easily. So it's again, a very productive thing. And I think of anything we're going to talk about in jQuery in this course other than selectors, which will definitely up your productivity, I think the append, the prepend, wrap, and remove features are extremely powerful for situations where you have some really dynamic pages, especially in AJAX scenarios where you're getting data from the server. So now that we've talked about some of those, let's look at an example of some of those in action and see how it works.

Adding and Removing Nodes Demo

The jQuery DOM manipulation APIs are really easy to use, so let's take a look at how we can use append, prepend, and some of the others that we just discussed. So first up, I have a div, and this particular div is TableContainerDiv, and right now that's all it has is a table, but let's say that above the table, and maybe even below the table, we would like to put in some extra nodes. Well, if we did this the traditional JavaScript way, I'd have to come in and say var child1 equals document.createElement, and then we'd have to create an element. Let's say that I want to add some divs, so we could do div. Then we'd have to say child1., and we could get to the attributes or we could set attribute actually, and we could set those or we could go through the attributes collection, either way. And it'd be quite a few lines of code in order to build that up. Now you might say well, you could also go in and do document.getElementById, and we could find TableContainerDiv, and then we could get into the innerHTML and we could manipulate that maybe. But if we want to add stuff above and below, it'd be a lot of string manipulation going on, plus it just gets long. Well, with jQuery it's really, really easy to append and prepend different nodes. So the first thing we'd have to do of course is find the node. So we'll do TableContainerDiv and that'll find it with an ID selector. The next thing we'll do is we're going to call append. So I'm going to put something at the bottom, and let's just put in a span, and I'm going to give it a style so that we can pick it out easily. We'll say background-color is green, and we'll just say Appended Child 1. Now I'm going to kind of borrow that line. If we want to prepend, really the same thing. I'll change this to Prepended Child 1, we'll change this to prepend. And we're off and running, and that's it, you don't have to build up this document.createElement and set the attribute, it's quite a bit of code to do that, or even really touch the innerHTML property. What jQuery will do is it'll actually convert this string into a node, handle all the attributes, all that stuff for you, and then append or prepend as appropriate to your target parent. So if we run this now, we should see above a span and below a span that's a green background, and there we go. So we have our prepended child and we have our appended child below it. So moving on, we could do this the opposite way as well. Now I don't prefer this way nearly as much, but we could come in and we could give it the string first, so let me cut and paste that, then we could say appendTo and we could find our TableContainer. Now right off the bat, this is not the most efficient way to do this. So let me go ahead and fix things up a little bit. What I'm going to do is cache this object because this is going to select it each time. So we're going to cache that and just say tcd.append, and same thing here to keep that a little more efficient and it'll be a little more compact as far as coding goes. And so we'll say append this particular child 2 to that node, and since it's going to be appended, let's add a br so it looks good. Likewise, we could do the prepend much the same way on this prepend that we just did, I'm going to add a br. Well actually we'd need to be right there because this will be the second child now. So we'll go ahead and take that one out. This will now be the first child because we're going to prepend it to the same parent, this TableContainerDiv down here, and let me change this to Prepended Child 2. So if we run that, now we should have four green areas, four spans. There's our Child2, Child1, Child 1 and Child 2, and you can see how that works. When you append, it always adds it at the bottom, when you prepend, it always adds it at the top. Very, very easy to work with. Like I said, I prefer this way, but this way is certainly valid as well, so whichever way you like. Okay, so that's an example of working with those types of things. Now, some other things we can do with manipulating the DOM is there may be times when you have some children and you want to wrap those children in a specific parent, because maybe that parent has a different class you want to apply to change the border or the background, whatever it may be. So we can come in and use wrap to do that, and the wrap method is really, really easy to work with. So if we scroll on down, I have three spans right here that just gets written out one after another with a space in between them, and if we run it, you'll see these spans right there. So Span 1, Span 2, Span 3. Well, what if I'd like to wrap a div around the spans? Let's go ahead and do that. So first off, we'd have to know what the spans are. Now I just have kind of a bogus class of Foo here, so I'm going to grab that. We'll come back up and we need to go in and find that span with Foo. So I could of course just put that, but we want to make it as efficient as possible, so we'll do, since it's on a span tag, we'll do span.Foo. And from here, now I can say hey, I want to wrap that span with a particular parent. So I can come in and say div, let's go ahead and give it a class of, let's use one I already have up here. So I have BlueDiv and RedDiv. Let's go ahead and do RedDiv. So the class of that will be RedDiv, and then I'm going to also add another class, which is just kind of a marker class called ParentWrapper. And then I would simply end my div that I want to wrap. Now what jQuery will do is it'll automatically go find the span and then wrap this div around that particular span. Now let's run it and see what happens, and you can see it works, they're all going down now vertically and it almost looks like there's one single div that's wrapped around all the spans. Well, that's actually not the case, and to prove that, what we can do is I can go in, grab this parent wrapper, and let's go find that div. So we'll say div with a ParentWrapper on it, a CSS class, each. And now what I can do is grab the HTML. So we're going to wrap that this keyword and we'll call HTML and I'm going to put that up in an alert actually. So we'll say alert the HTML. Now you're going to see that when it finds these, each one's going to have its own span inside of it, so it's not wrapping one div around all of the spans, it's wrapping, really there's three divs since we have three spans. So we'll go ahead and run it, and there we go. So there's the first div, there's the second div for Span 2 over here, there's the third div that had the class, or has an ID it looks like on it, and we're kind of off and running. So very easy to work with this and wrap around particular areas. And it may be not quite as useful for a span because maybe like for instance around this form I want to highlight based on something, a background-color that they need to fill that out in case there was multiple forms on a given screen. All kinds of things, the sky is the limit as to what you can do. Now the other thing you can do that's really nice for DOM manipulation is you can remove nodes. So we can call the remove API. Let's see here. Well, we could go in and if we wanted to remove all those divs we just added that have ParentWrapper on them, then we could easily do that. Now that's going to remove the div, but watch what happens here. We'll get past these guys, it's still there, and there we go, they're all gone. So you can see not only does it remove the div, but it also removes all the children of course because now that child has nowhere to go. So that takes care of doing that really easy, especially compared to the normal way we do this with the DOM, which is you have to find the node first, then once you find the node you can remove it from the parent, which means you have to find the parent. It just gets a little bit more painful using the kind of custom JavaScript way. Everything I've shown you here works cross-browser, it's very, very easy to work with and as a result, you're more productive, at least I hope you are. So, that's an example of append, prepend, wrap, and remove. And there's some others you can do. Again, I'd highly recommend, check out the jQuery documentation, let's run there real quick because I'm going to show you how to find these. If you go to jquery.com/docs or docs.jquery.com, either way, you can come down to Manipulation it's called. Well this is all the DOM manipulation APIs. So there's the appends, we've already talked about attribute, you can do before, you can clone things, we've talked about CSS, and there's a whole bunch, we won't have time to go into everything, but there's some other remove type things. You can remove attributes, remove classes, we're going to talk about that next actually. Replace, so I could actually go find a node and replace it with a different node if I wanted. So there's all kinds of cool stuff you can do here, including even unwrap, wrap and unwrap. So I'd highly recommend, get to know all these different APIs because there's even more than we have time to cover in the video series. Alright, so no pun intended, but that's a wrap.

Modifying Styles

There's going to be times when you want to modify the styles of elements in the DOM, so that's very easy to do and it's actually something we've briefly talked about, but let's officially jump into working with styles. So, you can jump in and work with the CSS function and it makes it really easy to get the value of a particular item or set the value of a particular item. So in this case you'll notice that we have a div, and on all the divs, since it's a tag selector, we're going to go in and say hey, let's change the font color to red so that any text that's in those divs would automatically have that particular color. Very, very easy to work with. Now what's nice about the CSS function is it's kind of like the attr function. You can also go in and you can set multiple values. So we can go in, and in addition to the color, which in this case would set kind of a light gray, ccc, we could also go in and say let's also change the font-weight to bold in this case. Well, this is the JSON object that we talked about earlier. So, we can come in and we can actually pass CSS a JSON object, just like we can the attr function, and this JSON object starts here, it ends here, and of course it has two properties. And so you'll notice these properties are quoted, in this particular case you can get away with either non-quoted or quoted for the properties, but it's a little bit different than we saw before. So that's really nice though when you'd like to set multiple CSS values. Now earlier, I ran a demo where I chained multiple CSS, because mainly I wanted to show how you could chain jQuery functions together. Well, you really don't need to do that, you can just pass the different values you want as a JSON object and you can be done with it, it makes it much easier to work with. So let's take a look at how we can work with that in our application.

Modifying Styles Demo

I've already demoed the CSS function earlier, but I'm going to go ahead and come back up to what we did and work with that a little bit. So I showed you how we can call .css and pass in the background of yellow, the color black, but I chained multiple CSS functions together with the attr function. Now, that certainly works and is fine, but in this case I really wouldn't chain them, I prefer to use a JSON object. So what I'm going to do is we're going to convert this guy into a JSON object, and then we'll come in and actually assign these values and we'll use the colon to separate the property name from the property value. So we'll take out the comma there and we'll do one more right down here, which is going to be our font-size. Okay, now let me change this up just a little bit and we'll go ahead and have, let's say, purple and white, not maybe my favorite colors, but we'll go with it. So I'm still going to use chaining, you'll notice that I have the attr function to set the title property, but then I'm going to chain the result object that's returned and assign CSS to it, but I'm not just going to do multiple CSS calls, that's kind of wasteful on syntax. Instead, we're going to come in, and let me just wrap this down to make it really easy to read, we're going to define a JSON object start and end, and then we're going to define three properties, separate those with commas. So if we run this, we'll now see that we get the same effect, but we have purple here, and there's our div and the names don't really match anymore, but you can see the idea there. In fact, just to demonstrate chaining, we could come in and we could chain this and say text equals Changed Color. So we're going to chain three methods together now, three functions, and we'll get through the alert from a previous demo and there we go, we have Changed Color. So chaining is really nice, but the point of this demo is that although you could chain multiple CSS functions together, it's a lot easier generally, and I think it's easier to read because they're all in one place and it's easier to maintain as a result, but you can just do that directly using the CSS function and a JSON object. It makes it really, really nice to work with.

Modifying Classes

The next topic we'll cover is how we can modify classes. It's one thing to change the styles, but that gets a little bit messy since we'd have to duplicate those styles maybe across multiple elements. Oftentimes to follow best practices, we'll want to use CSS classes. So let's talk about how we can modify classes and do some different things there. So to jump on in, there's four main methods you can work with classes, and there's some others as well, but addClass hasClass, removeClass, and toggleClass are ones that you use fairly frequently if you're working a lot with CSS classes. It makes it super easy to change the DOM, how it looks and feels to the user in the browser. So addClass pretty much does as it says. You can add a class. If there's no class at all, attribute on that DOM element, it'll add it. If there already is a class, it's like adding a space between two classes or more. HasClass simply checks. You can pass in a string with the class name, it'll return true or false if it has that class. RemoveClass can either remove all classes off of a node or it can go in and remove a single class. And then toggleClass turns a class on and off based on is it there or isn't it there. That one's actually really, really useful in a variety of scenarios. So let's take a look at how we can use some of these. So the first one up is addClass. Now very, very simple to use. All you have to do is go in and find your particular DOM elements or element, depending on what your selector has. And in this case we're going to select all the paragraph tags and we're going to call addClass to each paragraph and we're going to add classOne. Well, you could also add more than one if you want at the same time, just do it like you would in HTML. You can go in and add the space in between the classes and it'll do the same exact thing. So we'd go find all the paragraph tags in the document, we'd call addClass, and then this will add classOne and classTwo onto those. So really, really easy to work with. Now sometimes before you add a class, maybe you want to check if it has another class or something associated with it for whatever reason. Well, what we can do is we can use the hasClass method, or function. And so this shows in the example of hasClass, and all you do is pass in the class name that you want to check if it has that class, and in this case we're going to go and say hey, is there any paragraph tag that has the class styleSpecific, whatever your style is of course would go there, your class. And then based on if that's true or false, we can then come in and perform some work inside of this little JavaScript if statement. So it makes it really, really easy to work with. So, this is useful anytime we want to check if a DOM element loaded up into memory has that class or not, and then based upon that criteria, you can perform some work. The next one we can talk about is removing CSS classes. Now, this is very useful actually when maybe the user is filling out a form, and as they do that, you change the div that wraps it with a background-color. Then they move out of that, they lose focus, and move to some other part of the page and you want to change that background color so it's not highlighted anymore, you want to totally remove the class. Well, that's super easy to do with removeClass and addClass. So the paragraph tag in this example is going to be found, all of them, and then we're going to call removeClass, and because we're giving it specific classes to move classOne, classTwo, classThree, whatever it is, or just one class, it'll automatically remove just those classes that are available. Now if you just want to remove all the classes, you want to strip everything off, it's like stripping down the walls when you're painting, you want to take everything off and get it back to the base, then you could just call removeClass and pass nothing. JQuery will automatically go in, find all the paragraph tags in the documents, and strip off all the CSS classes from those. And you can see how productive that is. It's pretty amazing, especially if you've been doing it the old way, which is you'd have to go do document.getElementsByTagName, then loop through all the p tags and clear out the class attribute. We don't even have to access the attribute, it just does it for us. Very, very easy to work with. So these three up to this point, addClass, hasClass, and removeClass are really nice, but there's one more that's also extremely useful with things like hovering over items, and that's called toggleClass. So there may be times when you want to add a class, but when it's not there and you call this code, you want it to remove the class, you want to toggle it on and off like a light switch, it's on, it's off. What toggleClass does is in this example, if we have a CSS class called highlight, and it sets the background-color to yellow, then if we want to apply that class selectively based on if it's there, take it off. If it's not there, add it, then all we have to do is call the toggleClass API. So in this example we're going to do an ID selector, go find some ID of PhoneDetails, and then we're going to say toggleClass highlight. So if we run this code and highlight is currently on the PhoneDetails object, then it's going to remove it. If we run this code and there's nothing on that particular item, then it's going to add it, so that's why it's called toggleClass. So whereas you could use addClass with removeClass and hasClass, you could write some if statements and do this code yourself, toggleClass is a heck of a lot easier because if it's there it removes it, if it's not there it adds it and you don't have to write the if statements to take care of that. So let's take a look at a demo of how you can use these four different methods within your jQuery apps.

Modifying Classes Demo

So let's take a look at an example of how easy it is to work with the four class-specific methods for CSS, so addClass, removeClass, hasClass, toggleClass. So I have some textboxes, input, type equals text, and let's say first off that we'd like to add a highlight class onto those when the page first loads. So we can come back up top, and you'll notice that I have a highlight class already created for us that just sets the background-color to yellow. Well, what I'd like to do is let's go find all the inputs that are type text, so we can do that with a selector, so let's go ahead and we'll say Class functions. And we'll go in and say input type equals text, and that'll find all the textboxes for us with our selector, and then we can say addClass and we can give it the class Highlight. And that's all we have to do to take care of that. So we can go in, we can run this, and once it gets to that code you'll see that it automatically highlights both of those and takes care of that for us. I'm going to take out that alert real quick. And likewise maybe we want to remove. Well, let's go down and let's see what we have here for an ID. We have LastNameTextBox for the ID of this guy. So, likewise we can come in, we can do, in this case an ID selector, removeClass, and then we can give it Highlight. Now because there's, I believe only one class on here at this point, we could just leave highlight out and it would still work, but let's go ahead and run it as is and you can see now we just have this guy highlighted. Very nice. Now just to prove the point, let's go ahead and just to remove class and you'll see the same result, but it'll strip off all classes. So if we had RedDiv or BlueDiv plus the Highlight applied, it would strip off everything together. I'll go ahead and put that back for now. Now you obviously probably wouldn't even want to do this, why highlight it in the first place if you don't need it. So, what we could do is we could do the hasClass. So we could do LastNameTextBox hasClass Highlight, and we can wrap an if statement around that, and then we could change it based on some condition. Now I'm not going to do that, I'm going to do a little different scenario. So I want to make it so that first off we don't highlight it right off the bat. So I'm going to leave those as comments, but we don't want to highlight that. Instead what we want to do is when that textbox gets focus, we want to go ahead and set it to yellow with jQuery or unset it, remove that class. Now again, I could combine addClass, removeClass, and hasClass with all this, but toggleClass is really what this is geared towards. So we haven't gone into how to handle jQuery events yet, so I'm going to handle it the traditional way and put it right into the HTML, but keep in mind that we can certainly do this a different way and that'll be coming up in a different module. So I'm going to say onfocus, let's call the Focus function and we'll just pass this particular item, and then we'll also call onblur and we'll call Blur and we'll pass in this and let's put that on both of these textboxes. So we have our onfocus and our onblur, and they're going to call the Focus and the Blur and pass themselves into that. Now again, this is the traditional way to do it. There's some other ways you can do this, I'm going to show you in a later module. So let's go ahead and make those functions inside of our script block. So we'll make a Focus, we'll just call textbox, and let's do a Blur. Now this is going to, when we call this, it's actually going to pass in the DOM object, so we're going to go ahead and wrap that. And we're going to do toggleClass, Highlight, and then when Blur is called, we're going to toggleClass Highlight. Now what that's going to do, and we could, I'm going to change this up to wrap up, but we could just call the same one, we could call, I don't know, FocusBlur or something, but what we'll do is now when they focus, if Highlight is not on that textbox that's passed in, we're going to set it, or vice versa, if they Blur and we call toggleClass Highlight now should be on it, because this would have been called first, and we're going to remove it. So let's go ahead and try that out. So we'll focus, you can see it sets it, we'll change, and you can see it removes it. Really, really nice, and makes it super, super easy to work with. Now obviously there's some other CSS tricks you could do to work with this type of same scenario, but there's a lot more you can do because now we have full JavaScript. Now let's just go ahead and change this to FocusBlur, and we'll just use the same exact item here, same function, and so we'll set this to FocusBlur, FocusBlur, because now, we really just need the same exact function called, because if it's there, it's going to remove it, if it's not there, it's going to add it. So we're going to be able to reuse our code a little bit better, refactor it. So we'll go ahead and run it, we'll focus, and you can see we get the same result. Really easy to work with. So obviously you can make yours look a heck of a lot better than what I have going here, the goal of what I'm showing in this lovely screen is mainly just to focus on the concepts, but you could certainly take it up to the next level and make it look really nice for your particular forms. But that's an example of using addClass, removeClass, we could do hasClass, and then toggleClass. It really simplifies what we'd have to do otherwise and I like the fact that we can still leverage traditional JavaScript where we pass in the DOM object, then we put that in a jQuery wrapper so that we can access the jQuery APIs. It makes it super easy to work with. So that's an example of how you can work with classes using jQuery syntax.

Summary

In this module, we've seen how we can iterate through nodes and modify the DOM in a lot of different ways. So we started off by talking about the each function and I talked about how you can, as you're iterating through the nodes with each, get to the index value and how we can take the raw DOM objects that were fed and wrap those with a jQuery wrapper to do all kinds of other things. I also threw in a couple tips and tricks there for performance that I would definitely recommend you take a look at and consider as you use that each method. We also talked about how you can modify properties and attributes. We talked about how if you have a raw DOM object you can get to the properties of that object directly versus if you have a jQuery object you can use the attr function to modify the attributes of that particular element from the DOM. From there we moved on to modifying the DOM using append or prepend functions. Very easy way to do it, definitely simplifies things as compared to using document.createElement, but keep in mind, you can still use the traditional ways of interacting with the DOM, it's just in most cases jQuery is going to make it easier for you. From there we talked about the wrap function and how we can wrap different children with a parent. It makes it really nice if you want to change for instance the styles or classes associated with that group of items. And we talked about how we can change styles and work with classes and even toggle classes on and off, it makes it really easy to do that as well. So a lot of good stuff in this chapter, and I hope that'll help jumpstart your jQuery development and your projects.

Handling Events

Introduction

If you're working a lot with JavaScript in the browser then odds are you probably have to handle different types of events. After all, you need to know when the user clicks on a button or when they select a different item from a drop-down, or maybe you even want to know when they mouseover a div or something along those lines. Well if you've done it before by hand and had to deal with cross-browser issues you know it can be a little more challenging than you would think, at least at first glance. And that's because depending on how far back you need to go in the browsers as far as the support goes, it can get a little challenging because the older browsers, especially Internet Explorer, didn't support the standard at the time. And the newer browsers do so now you'd have to write some code to figure out which one to do and we'll be talking about that more coming up. So what we're going to talk about in this module is how jQuery really simplifies handling events, and not only does it make it easy but it provides a very robust API; and you can handle different types of events in different ways and we'll talk about several of those ways as we move along here. So the agenda is we're going to start off by talking about some of the cross browser issues out here and the challenges that provides us as developers, and then from there we're going to jump right into jQuery and talk about so what are some of the shortcut functions it has that allow you to handle events. We'll talk about some of the new on and off type of functions that can be used for binding two events in a really dynamic way. And then we're going to talk about how events can bubble up and how we can handle those and provide some efficiencies, especially when it comes to working with a lot of children in a table or maybe a bunch of children in a div or things like that. So we're going to talk about the live, delegate, and on functions, and I'll introduce live and delegate—they're a little bit older—mainly so you know about them in case you come across them on a blog post or something like that, but we'll mainly focus on the new and preferred API, and that is the on and the off. And then we'll wrap up by talking about different techniques you can use in jQuery to do hover effects. So if you've ever handled it by hand with an event you know that you can do things like mouseenter and mouseleave, and you can do that in jQuery as well and I'll show that. But there's some built-in things that make it really nice to not only detect the events but also to swap out styles, swap out CSS classes and things like that. So let's go ahead and jump in and get started by talking about how jQuery simplifies the process of working with events.

jQuery Event Model Benefits

Let's take a look at what jQuery brings to the table when it comes to building JavaScript enabled applications that have to deal with events. So first off a question for you, what type of JavaScript code do you write to handle a button click event? Think that through real quick. Now I'm not talking about doing it the HTML way 100% JavaScript and if you've been doing this very long you probably said well Dan, that's a trick question, and it is. It depends on the browser because each browser does it a little bit differently depending on the category, and let's break those categories down, it's pretty simple. You have IE8 and earlier, and in Explorer8 and earlier, and pretty much everything else; IE9 included in the everything else. They all do it differently between those two categories so most browsers will simply take the DOM object and allow you to say addEventListener and then give it the name of the event and a callback. That's the standard way to do it, that's what you do in Chrome, and Safari, and Firefox, and all the major browsers out there. Well in IE8 or earlier it's a totally different way to do it. Something Microsoft came up with that's now fortunately remedied in IE9, but there's still a lot of IE8 and earlier out there and will be. So what you do is you take the DOM object and you say attachEvent and you'll notice that the name of the event you want to attach to is even different than the one above, which is click, so you say onclick instead of click. So that right there can make it confusing because you have got to get it right; it's a string and you have to, you know, make sure there's no problems there. So what jQuery brings to the table is it wraps both of these pieces of functionality and it abstracts that out from you as the developer. You don't have to worry about it; it just automatically takes care of knowing which one of these to call based upon the browser. So to wrap up this particular section of the module let's just do a real quick review if you're newer to this whole concept of events before we go on to talk about some of the jQuery shortcuts for handling events. So events are just a way to notify that something has happened. So an example would be a user clicks a button, you as the JavaScript developer need to know that they clicked the button. Now, if you're coming from a Java or a .NET background—some other language that handles events—then you know, this is old news to you, but that's all we're talking about is you focus on a textbox, we want to know; you mouseover something, we as the application want to know. So jQuery provides a very rich API for cross browser event model support across all the major browsers and some very compact syntax—a lot of different options and some cool stuff that we're going to talk about that to do it with raw JavaScript would be pretty difficult. So in the next section we're going to be talking about some of these shortcuts and talk about how we can hook up to events in a cross browser way.

Using JavaScript to Handle Events Demo

Before actually jumping into how to use some of the JQuery functions, I wanted to show the traditional way that we write JavaScript to be able to handle events. So I have a very simple form here and you'll notice at the bottom we have a SubmitButton and what I'd like to do is add a little bit of code to programmatically hook up; now we could certainly come in and we could do it the JavaScript or the HTML way, however, I want to do it programmatically. Now the first thing that I have to worry about is I'd have to know so when is the page loaded? So I'd have to deal with the on load and handle that with IE or with all the other browsers. Now what I'm going to do to get around that is one of the easy ways to know when it's loaded—it's just down here at the bottom. We can come and put a script tag so I'm going to put it right outside of my form actually. And then inside of here we can add some code to handle the events, I'm just going to put Handle events the old-fashioned way. So this is what the code would look like. First off we'd probably want to go get our button, so you'll notice it's called SubmitButton. So we'd say document.getElementById. And then from there we need to know which browser are we in; are we in IE8 or earlier or are we in anything else? Well the easiest way to do that is you could just come in and say hey, does the document support the addEventListener? If it does support that API then we can go ahead and do it with what most browsers expect, and if it doesn't we'll have to do it the IE way. So now that we have the button we can say hey, let's go ahead and add an EventListener for the click event and then I'll just put an in line function; we'll just alert Clicked Button. Okay, so that's kind of the standard way to do it, but as we know IE handles it differently, so we'd have to do attachEvent and onclick and then we can give it our function. And I'm going to go ahead and this time say Clicked IE Button so that we know the difference. Okay, so that's kind of your two options for attaching EventListeners, and of course depending on the event you would substitute in a different name here. So we go find the button, we then check does this browser support this API, if it does we're going to use it, otherwise we're going to fall back and assume that Internet Explorer. Okay, so if we go ahead and run this as is, right now I'm in Chrome and you'll see we got Clicked Button because it supports the official addEventListener. Well let's go ahead and grab that URL, and we'll open the same thing up in Internet Explorer and run it and you'll see we get Clicked IE Button. So you can see it's not that hard to work around, however, it still is quite a bit of code and it'll really make you appreciate what I'm going to show in the next section of this module, which is some of the jQuery shortcuts.

Handling Events

Let's take a look at what jQuery offers us out of the box when it comes to handling events in a cross browser way so that you're not pulling your hair out worrying about which browser works and which doesn't for the different attachments. So there are a lot of different shortcuts that are built in to jQuery that automatically hide from you the cross browser issues that were mentioned in the previous section. So here's just a few of them: click, blur, focus, you'll notice there's even a dblclick, which is a little trickier to do in JavaScript without—you'd have to set up interval times and things. But different mouse events and key events and all that type of stuff, and what you can do with these is find your object using the selector, simply hit .click, .blur, whatever it is based on the object and then have a callback function; very, very simple. So you can get a lot of details about this at the URL below: api.jquery.com, so let me go ahead and pull up the browser and show you what's at that site. So the event documentation for jQuery is actually really good. A lot of details, all the API functions are called—or covered I should say—and there's even demos with most of them. So if I scroll on down you'll notice that we have this bind, we're going to talk about that a little bit later—that's not a shortcut function, that's actually what the shortcut function's called behind the scenes. But here's blur, change, click, dblclick; we'll talk about some of these others later—delegate, and die, and live, but in addition to the shortcut functions there's also this event object so you'll notice that you can get to different properties of the event, such as if you want to know the pageX and pageY—the mouse position relative to the document. Then what they've done is the event object that's passed when you use one of the shortcut functions—such as mousedown or mouseup—will automatically pass you an event object that's wrapped with jQuery wrapper and automatically gets you into the appropriate property based upon the target browser. Now, the event object itself is even different across the different browsers, or can be. So that's a nice thing. So if you want to get to the target for instance, there's only one way to do that if you go through jQuery so you don't have to worry about the hassle of doing it on your own. So here's where you can get to the different API's; now I'm not going to have time to cover every single one of these but let's do a real quick demo of say dblclick. So if you go in to dblclick you'll notice that you have to give it a handler, and the handler, which is the callback will allow an event object to be passed in. And then scrolling on down here's how we use it, it's very simple. So we grab the target, in this case in other words the DOM object wrapped with a jQuery wrapper. We call dblclick and here's our callback. Now we could pass an event object right into here if we wanted, but in this case we're just going to pop up an alert. So at the bottom of this we have a little better example where we have a particular DOM element we're going to find and then we're going to attach the dblclick function to it and have it our callback. Now you'll notice when it's dblclicked on the target we're going to go ahead and toggle the class called dbls. We're going to switch it back and forth between different items so here's dbl actually, which is a yellow background. So I'm going to come down, we'll dblclick it, you'll see it change to yellow, I'll dblclick again, that'll take that class off into the toggle and we're back to where we were. So that's an example of the documentation, and with the events it's really nice because there's just a lot of these shortcuts that you probably won't have memorized of course, that's where you can go to look. So now that we've seen a few of the shortcuts and how you can get more documentation about them let's look at a quick example here. So the click is one of the more common ones, and as shown even with the dblclick its are very similar. You have a handler and that handler can even have an event object passed if you'd like. Well, in this example we're going to go find an ID selector so we're going to go find myID, whatever that is, attach our click to it, identify our callback function, and then in this case we're just going to pop up an alert. So nothing real fancy on this particular example. Another example that is shown here, and so you'll notice that we call click, but in this case we target some other objects click. So you can actually trigger events, there is a trigger function actually, or in this case if we just want to trigger in essence the click event, then we're going to say hey, when this one is clicked we're also going to cause this one to be clicked. So click in other words is not just for receiving the data, but you can actually trigger these events on other objects if that fits your particular application. So it makes it very easy to work with that way. So let's take a look at using some of these shortcut functions in an application web page.

Click Event Demo

Let's take a look at how we can use the different jQuery shortcut functions to easily attach to an object and hook that up to an event handler for events like click and some others. So I have a form here that's just laid out using a really simple table structure; if you like divs, by all means, you can do that too. Going back to the source, the traditional way we kind of wire up this button here is we come in to the HTML and we do onclick = and we give it some functions, so we'll just call this SubmitButton\_Click. And then we come up and add our script block. And then finally add our function. And let's just put an alert to get it going to start. Okay, so that's kind of the traditional way and there's nothing wrong with that, it still works fine, that works cross browser very nicely, and if we go ahead and run this guy you'll see that when we click it we get Clicked Button. So if that's all we need to do then you could certainly go that route. Now the challenge with this, and I'm kind of anti hooking up events through code now days this way because I like to have one place I can look, so generally I prefer actually to do it through code because then if I want to change things I have one place instead of having to scan a big HTML document. But there's absolutely nothing wrong with what we just did, it certainly works. So let me go ahead and take that out and let's do it the jQuery way. Now we'd have two options here if we want to hook to this button and do an event handler for the click, then I could come down into the script block that we used earlier in this module for a demo where I showed kind of the old way of doing it, but let's go ahead, this is a good reason to use the ready function. So let's go ahead and go back to that. We'll wrap the document object and say .ready and we'll have our function. Okay, because that way I know everything's loaded, but again you can always do it different ways, but that's a good example where the ready function's available. Now what I like to do is I'll often times have my own custom function in here, I'm just going to call it WireEvents to keep it real simple. And then when I know it's ready I'll simply call WireEvents, and then in WireEvents we can come down and find our SubmitButton and we can go ahead and do an ID selector to get to it. Now we can get into all the different event shortcut functions that are available with jQuery. So obviously for a button I'd probably want to go ahead and do click, see you notice this takes our handler. And we could do the same thing as we did before. Okay, so that would kind of be the jQuery way, and you know, if you don't like putting the WireEvents like I did you can certainly put this up in the ready as well, but in a real live app what will typically happen is there might be quite a bit of things you wire up and you might sense some CSS dynamically based on the user, so I generally like to break that out into functions to make it easier to maintain. So let's go ahead and run that. And we'll go ahead and grab Clicked Button. Now that's not real impressive, let's go ahead and take to the next level instead of saying Clicked Button. Let's go ahead and grab the value from let's say the first name and last name. So we have FirstNameTextBox and LastName. And let's say that we'd like to just write those right below the button here. In fact I'll do it right to the right of the button in the form. So we'll just put a simple div and we'll give it an ID of DivOutput. Okay, so we'll come in here and let's go find DivOutput and we can simply do the HTML or text, and I'm just going to go ahead and update the text and keep it simple in this example. And now we need to grab our text boxes, so what I'm going to do to kind of make it easier to read is we'll just say var value = and we'll grab our FirstNameTextBox and we're going to grab val. We talked about that briefly in one of the previous modules in a demo, but that's the easy way to with jQuery get the value of that when an event has happened and you're dealing with form input controls. And we can come in and do LastNameTextBox; do the same thing. And let's call this—actually I meant to call that first name—and lastNameVal; keep it simple. Okay, so pretty standard code, all jQuery though. We have the ready function that fires that calls WireEvents; WireEvents is now going to wire up programmatically to our click button so we're going to keep it out of the markup in this case and do it programmatically. When that gets called we'll go ahead and go into here, grab that FirstName and LastName value and then concatenate them together. It should update that div down there at the bottom. So let's go ahead and run this. And let's just do John Doe. And you can see it outputs it like you would expect. Now that's just the first example of using the click.

Change Event Demo

Now that we've seen the click let's go ahead and add a little bit more. Maybe when they select a different value from our select here, our dropdown, we might want to do something specific when that particular event fires. Now again, doing it the kind of normal way we could come in and say onchange, but we're going to go ahead and do it the jQuery way. So in the WireEvents I'm going to come in and we'll say Handle select and we'll go ahead and go and find that for our states. So let me go ahead and I'm going to change this ID to StatesSelect—make it a little more obvious. Alright, so when we come in, you know, of course we don't want to do a click on this—wouldn't make much sense—but we can come in and we can do a change. Now the change does the same type of thing. We can have event data that's passed in or if we don't need it we can just go ahead and handle it directly. Now I'm just going to go ahead and handle it directly with no event object. And there we have our change. Now when the change fires what do we want to do with it? So what we want to do is obviously grab the value of our select and work with that. Now, the easiest way to get that is when the change is fired then I'll be passed in also the object that changed. And so we can actually come in and say this and we can grab a val. So let's go ahead and we'll just pop that up in an alert just to see it in action. And let's run it. And you can see we get the value that was clicked, so very nice and very easy. I didn't have to worry about getting the selected index or anything fancy; you can see I could just get directly to that object without having to, you know, retype that ID and look it up again in the DOM. Now here's an interesting thing though with change; it actually works not just on selects in jQuery, it actually works with input controls or text blocks. And so we can come in—or text areas I should say—we could come in and let's come on to here and let's just add a kind of a bogus class here. I'm just going to call this MyInput. And I'm going to go ahead and put that on two of these. I'm also going to put it on the select. I'm going to demonstrate the concept here. So we now have MyInput on a text box, a text area, and a select. And so coming back up what we can do now is instead of saying only find this ID we could do a class selector on MyInput. Now whatever changes will get the val. Now watch what happens here, this is pretty cool, and something that you just out of the box can't really do with JavaScript. So let's go ahead and take that off and I'm just going to mouseout and you can see the value—there is no value. We're going to go to John—I'm now going to mouseout, now I didn't put it on that one so it shouldn't do anything; see it doesn't. Doe, now this one should trigger it; it did. And then This is my comment, mouseout. So you can see that the change actually can be used much like we would do with a key press down or key press up, or focus or loss focus; it makes it really easy to work with this. Plus, it also works with the select. Now I don't know about you but I find that pretty cool because it makes it really easy if there is a scenario where I need to know multiple things are changing because I might change a style or whatever it may be. As an example let's say that we do have a style up here; well actually let's just reuse our highlight— that's what I was just going to do, forgot that was up there. So let's come in; instead of just alerting the value—we can do that too—but let's go ahead to that element and we can do addClass, Highlight, which will set the background color. So now once they've filled out particular aspects of the form—the first one doesn't have it so it won't' do anything—Doe; now once it shows the alert it's going to highlight what they've changed in the document. And you can see it's now highlighting everything; now you may or may not want that behavior or that particular color, but you can see what can be done very, very easily using jQuery and the change function.

Mouse Events Demo

The next thing we'll look at now that we've seen the click and the change is some of the mouse shortcuts that are available within jQuery to handle events; makes it really, really easy. So I have just a div down here and let's say that when we interact with that div we want to maybe do something. We want to update that HTML; we might change the background, something to that nature. So two things I want to show: number one I'm going to show the shortcut functions, but I also want to show the event object; we haven't really talked about that yet, I'm going to show kind of how that works. So the first thing I'm going to do is let's go find MyDiv, and then once we've found MyDiv let's go to the different mouse options. So you see we have mousedown, mouseenter, mouseleave, all the way on down to mouseup there. So quite a bit of functionality built in out of the box. So let's say that when we have mouseenter we'd like to execute a function. And I'm going to chain onto that; we want to do mouseleave, another function. And let's go ahead and we'll leverage our highlight class to highlight the background of that particular div. So the easiest way to do that of course is we can do the toggle class. Now we can certainly do addClass as well but since we want to only add it when they mouse in and out it's a lot easier to go with toggle class; otherwise they'd have to do an addClass here and removeClass here. Now there's going to be some little code duplication to start but we'll talk about that. So we're going to do this. and we'll do toggleClass, and then just to get the demo working let's go ahead and cut and past that, and definitely we would want to re-factor this—right off the bat that should raise an alarm; we have two kinds of code that are the same. I'm actually going to show you an even better way to do this with less code when we get to the little feature section in this module covering the bind and unbind—really useful. Now we certainly could re-factor this out, let's get it going and then we can do that as well. So we'll mouseenter, should show it yellow; mouseout show take that yellow off and that's going to highlight this particular div here. Alright, and there we go; you can see it in action. Now we might also want to go in; let's say when we mouseover we want to change the cursor, well that will be easy to do, we have the object. We can go to the CSS property or function I should say, and we can say cursor and let's change it to a hand, which is the pointer. Now that'll automatically change it; you'll see as we mouse in and out we change to a hand to allow the user to click on it whenever they'd like. Now let's go ahead and we may also want to redo some of this. So let's go ahead just for the sake of showing it so I feel better, let's make a new one called Toggle. And this is going to take the object that we're working with, so this—we'll just call it div. And what we can do is simply take div, whatever it is, and notice if we do div right here .toggleClass we're not going to get any IntelliSense. Let's run that as is; we'll see what happens here. So we'll pass Toggle and we'll pass this. So at least we have the functionality in one place. Okay, so let's go ahead and run that as is. And you can see it still works. So the downside of this is you're not going to get IntelliSense because at this point it doesn't really know it's a jQuery object; now you could certainly come in and what I would do actually in this case is instead of wrapping it here it's probably going to be easier to simply do a wrapper here so that in this case Visual Studio knows about it—now we're back to our IntelliSese and that way we don't double wrap it or something like that. Let's run it one more time and we're still back in business here. Okay, so that's an example of using the mouseenter and the mouseleave and it's just as simple for the others. Now, a little bit more though I want to wrap up with, and that is we haven't talked about passing the actual event object into any of this stuff. And so as an example when this function is called; there is an event object that's available. In fact I showed that a little bit earlier. We went out to the jQuery site and I showed you we can get to things like the current target of the event, the pageX, the pageY, there's a source target—there we go, right there. And so you can get to all that stuff directly. So let's go ahead and say that also on this div—we're going to chain one more here—we'll do a mouseup, we'll pass in a function, but this time I want to just pass an e, you can name an event and call it whatever you want. But now we can get to event data about this. And so what's nice is we can go in and let's just change maybe the HTML of the div; so we already know the div, we've selected it so we could say this, and actually it's not going to—we could just do the text—and we could say e, well let's just do this. Let's do X is e.pageX. Now you'll notice you don't get IntelliSense here for the event object so I often just will run out to the jQuery docs to grab that. And likewise if we wanted, we're really feeling like showing everything we can go ahead and do the Y here. Alright, let's run that now. And now I can mouse in and out but when I click it actually updates the coordinates of where I clicked; you can see as I click around it just continues to update that. Move up towards the top it gets pretty small on the Y. Very, very easy to do. Now the other thing you can get to here that's nice is the target. Now if we just do e.target and run you see that it just gives us the actual object of this particular item, but remember we talked about the adder function so we could say target, and let's try to grab the ID of this and let's see what happens here. Okay, now why do we get an error? Hopefully by this point in the course you know. Okay, the answer is yes, this is giving us the target, but you notice when—let me run this one more time— that it was the raw object—it's the raw DOM object, and the HTML div element. So if we want to get to the jQuery features we always need to wrap that with a jQuery wrapper. Now we can get to whatever we want; I'm just going to say go to ID in this case using the adder function. Now let's run it. You'll see we get to MyDiv. So I've pointed that out a couple times in the different demos, but it's just one of those things to be aware of that when jQuery passes you things such as this you're getting the raw object. When it passes you the event and you get the target or the current target property then you're getting the raw object, so you always need to put it in a jQuery wrapper if you want to get to the jQuery functionality there. So that's an example of what we can do with the mouseenter, the mouseleave, the mouseup, all that type of stuff; very, very easy to do, and you know, what's nice about this is let's say that in addition to this guy that as a user mouses over each row we want to go ahead and highlight that as well. Well remember we can always put a comma and now we're going to say on this selector, match my div, and match the all tr tags. So if we run it, it may not be great colors here but you'll see that it highlights all the rows plus our div and that's just amazingly simple compared to the normal code you typically write if you do this with just custom JavaScript; very nice, very easy to work with. So that is a summary on some of the different functions that are available; the shortcut event functions and how we can use the mouse events and actually handle the event object to get to to some of the different data that's available.

Binding to Events

Up to this point you've seen how we can use jQuery shortcut functions to handle different types of events very easily in a cross browser manner, so it really saves you a lot of hassle compared to if we had to write this code from scratch and deal with IE6 through 8 and modern browsers and mobile browsers and all that fun. Well behind the scenes with the shortcut functions is actually a core jQuery function called on, and that's what we're going to talk about in this section is what the on function is and how you can use it, and I'll also introduce kind of the inverse of it, which is called off. Now the on function's very simple, it allows you to define the event or events you'd like to attach to and you'll see you can do that right here as a string such as the click. And then you define the handler. So in this example down here we're going to say when the ID of my div is clicked, so on the click of that div let's call this function. Now the first time I ever saw this on I hadn't really seen it before in a lot of other areas and it didn't make a lot of sense to me—I'm like on, what does that have to do with events? Well you have to read it kind of like a sense. MyDiv on click called this function and that's how you can think of it and then it made sense. Now this was added in jQuery 1.7 so I've mentioned the bind function a few times earlier, and bind is something that you could also use in jQuery but the recommended approach with jQuery 1.7 or higher is on. In fact, it is the core event function if you will that's built into jQuery. But that's all we have to do is simply give it our selector, give it the event or events, and I'll show you how we can handle multiple events in a moment and then give it the callback. Now if I wanted to pass the event object to get some data such as the target element and things like that, I could do that as well, but in this case we don't really need it we're going to assume. Now the opposite of on is off. What if you want some buttons that have click event handlers to be disabled for whatever reason because maybe the forms not valid and they can't submit it. Well, there's an off and it takes the event that you want to turn off. So as an example if we had a ID of tests when it's clicked we want to call some handler, and let's say that for whatever reason I want to turn off that click event, then I could just say test off. And now what that'll do is turn off not only the click but any other events that are attached to that particular DOM element. Now if you want to remove specific events such as just the click because maybe I had a mouseenter and a mouseleave attached as well, then we could do off and you pass in the event or events. Now you'll see that you just separate if you have multiples here with a space and we're going to see that in just a moment when we handle multiple events with on. But that's what off does. Now this only something you'll need when you want to sort of detach events of course; you may not use this one nearly as much as on but it's nice to know about. Now if for some reason you're working with jQuery 1.6 or something less than the 1.7 that I mentioned earlier for on, then there's a bind I mentioned, there's also an unbind, so off is kind of the newer version of unbind. But if you're on 1.7 or higher, which you probably are, then I'd recommend using on and off. Now you can also come in and bind to multiple events and this is actually kind of handy in some cases, so let's say that when they mouseenter or mouseleave on a particular div I want to call a function to toggle the entered class on and off, so when mouseenter is first triggered then if entered is not there toggleClass will add it, then when mouseleave is triggered entered is already on it so we'll remove it. So it's a very simple way to handle two events in kind of one go here. So that's the basics of how on can be used and we're going to jump into a demo now of using it. But there's some other cool ways it can be used as well, which I'll be showing just a little bit later in this module but until then let's jump into a demonstration and see how we can actually use on and off.

Binding to Events Demo

Up to this point you've seen how we can handle events using the shortcut functions like mouseenter, mouseleave, mouseup, and I chain those together and it works out really well. Well behind the scenes these are actually delegating to on; in fact, I'm going to show a nice source code viewer here that'll help you see exactly what the jQuery functions do. So if I go type in jquery source viewer this will take me off to the jQuery source viewer home here. And now I could switch to the latest version of jQuery. And as I type click this will let me see the actual source code that jQuery runs behind the scenes. You'll notice it's actually delegating to on. Now we could even navigate if you want to see all the details for on. Now likewise if I wanted to look at the focus, it's also going to on; in fact, all of these are. We can do the mouseenter and notice it's also going to on. So really on is the core event handler function that's behind the scenes. So what I'm going to do to demonstrate it is let's come on in and tweak this a little bit, so I'm going to comment this code out that we had before. And let's look at using bind and on. Now bind is the original version that we had prior to jQuery 1.7. So if you are in an earlier version because you're kind of stuck there for now then you're going to have to go with bind if it's prior 1.7. But 1.7 or greater you can actually go with our on. And so now I can come in and do the following: we can go find the same thing as we did up here and so we'll find MyDiv, and then I could do bind if I want and then give it the events. So we could say mouseenter, mouseleave, and the last one up here is mouseup. Now we're going to be able to handle all three of those and that's all you would do with bind. Now to change this to the modern way to do it you simply change that to on, so not a big surprise there. Now when these events are fired what are we going to do? Well we need a function so we're going to handle the function and pass an event arc here. And that'll represent our event object that gets raised. So when mouseenter, mouseleave, or mouseup are fired we're going to come in and do something with those. So I'm going to get to the div that was actually triggered here and we'll say toggleClass and I'm going to toggle a class we had up earlier called highlights, so just as a quick review it just changes the background to yellow. And so we're going to use the built in jQuery toggleClass to either add the class or remove the class based on the presence or lack of presence. So we'll go ahead and do highlight there. Now I'm also going to go in and change—we're just going to use CSS for this one but I could certainly do it through a class. And I'm going to change the cursor—let's put that in quotes actually. Now I could certainly go in to the highlight and change that, but in this case I'm saying I always want it to be there. I generally prefer to use classes but this will get the job done. Now I want to do something really special in this case—if its mouseenter or mouseleave this is all I want done, but if it's a mouseup I want to do what we did earlier and grab this text right here and write out pageX and the pageY location. So we'll come in here and put that but I only want to do it if it's a mouseup. So we could say if (e.type == ‘mouseup') and that'll let us know that it's just this one event, then we'll come in and we could handle the mouseup and we're going to write the text of the div and update it to the pageX and the pageY; in other words, where'd you click with the mouse? And that's all we have to do to actually go in and use the on. Now if at any point I wanted to come in and turn things off then we would do the same thing; we could say MyDiv or we could even cash that in a variable since we've already selected it and say off and then if we wanted to turn, for instance, mouseup off we would just do that and mouseup would be off. Now If I just put off it would turn off everything in this case. So I'm going to go ahead and just leave that commented out here. So now if we run this what'll happen is I can now mouseover the MyDiv and it should automatically change it, and then if we click in it when the up actually fires for it then we'll be able to see that and write out the different pageX and pageY. Alright, so there we go; so there's the hover affect that we had earlier, so nothing really impressive, we're just using the on right now and then if I click it notice that it now writes out the X:73 Y:22 and wherever I click it'll update it of course. And now we're able to handle all of that in one simple location using the on. Now you can certainly use the shortcut functions if you'd like; it's an alternative—it's really up to you on what you'd prefer. I do think the shortcut functions are nice in the sense that I don't have to type any quotes, but in this case I prefer the on because now I can actually go in and handle multiple events as you see. So that's an example of the role of on and as I mentioned, you could change this to bind as well and it would still work but that would only be prior to jQuery 1.7. The modern way to do it is to use the on.

live(), delegate() and on()

You've seen how we can attach events and event handlers directly to DOM elements throughout this module. Well jQuery provides some other really nice options that allow us to attach an event handler without really attaching an event handler. Here would be the scenario. Imagine you have a table that has a lot of rows in it; let's say it has a couple hundred rows. Well how do you know if the user clicks maybe on the tr or mouses over it? Well the normal reaction there is we'll write a selector to go find all the trs and attach an event handler—maybe a mouseenter, a mouseleave, and a click to every tr. Well if you had 300 rows then of course that means we're going to be adding 300 different event handlers, and that starts to build up in memory a little bit. So fortunately there are three different options that you can use in jQuery to handle this a little more efficiently and those three functions are called live, delegate, and on and we're going to be talking about these in this section. Now live, delegate, and on as mentioned provide a way to attach event handlers at a higher level and you probably know that JavaScript, when you click on things or mouseover, if nothing handles it at that DOM element then it bubbles up by default and a parent will then try to handle it, and then if they don't have one it bubbles up to its parent. And this is the standard way that events work with JavaScript. Well live, delegate, and on provide a really unique way to attach an event at a higher level in the DOM hierarchy so that when a child is clicked or mousedover, or whatever it may be it can actually bubble up. And so you'll see this provides a really efficient way to handle events in a variety of scenarios. So in a nutshell what it does is it allows children to be added into a container, such as a div or a table, and you don't explicitly have to attach an event handler to each child. Well let's go back to the row example; if you had 300 rows that means that depending on which of these functions you use you could attach an event handler at a higher level in the DOM hierarchy and as a child is clicked there is no event handler there, but it'll bubble up and then the parent will detect it. So let's dive in here and I'll give you some examples of how this works with each of these. So first up if live; now it's important to know that this is kind of an old API and it's been removed completely in jQuery 1.9. So if you're using an older version of jQuery then you could certainly use this, but jQuery 1.9 doesn't even have it at all. So why am I covering it?? Well because you may see this in blog posts or other areas, and although you may not use it after I show these other options, it's important to know what it's for and how it works and why some of the other options are better, so we'll go over it just real quick here. So what you can do is do a selector, and in this case we're going to target everything that has some class on it and we're going to attach this live binding to it. And what's going to happen is anything that has some class on it—the click event won't actually be attached at the DOM element that has some class. Instead what'll happen is it'll bubble all the way up to the document object and you can see up here—the document object handles events by default. Now what'll happen is we'll have one function in memory that acts as the event handler, it'll be attached to the document object and now when any child that has some class is clicked in this case it'll bubble up. Now you can turn that behavior off by using the die, so a lot of people call this live because it kind of looks like that; I guess you could officially say it's live because there's a corresponding die but it really doesn't matter—you say tomato I say tomato. But we say die and click and give it the same function that was attached and what that'll do is basically detach the click and the event handler from the document object. So let's look at a more visual example of this. So you can see that I have some class as my selector and I'm going to attach this live meaning for the click and you're going to see that ultimately that's going to be attached up on the document object level. Now imagine that I had a span, and a p tag, and a div, and maybe some others that all have some class. Well normally if you did .some class for the class selector .click then we would have three different event handlers attached to a specific function for each of those that eats up a little more memory. Well because we're doing the live binding here the click event will actually cause a bubble up effect and the event will actually bubble up to the document object. Now the document object will actually look for all kinds of events, but in this case it's specifically going to monitor children that were clicked that have some class on them as a class. And so if it does the click event will be fired and then the event handler will be called. And so you can see now we only have one event handler and even if we had 1000 elements with some class on it there would still only be one click event handler, so it's very efficient that way. Now the downside of this is it is bubbling all the way up to the document so let's look at another option. So a newer version of the kind of live concept is delegate. Now this was added in jQuery 1.4 as you can see and it's still available in jQuery 1.9 as well. And what it allows you to do is specify a context object— you're going to see that right in here—that actually is the parent that handles the bubble up events and then processes those events. So it works as new children are added into its container and will ‘ automatically detect what child was clicked and if it's appropriate, will handle it. So here's an example of using delegate. So our context is going to be this # or hash Divs, so the ID with divs on it. We'll say that's just a div that's a container. We're going to delegate that when any child that's a div is clicked, call this function here. And then you can undo this by calling un-delegate, it's kind of like the die I just showed you. Now this works the same way visually; the difference is it's not going to bubble all the way up to the document so we have this #Divs for our ID selector, delegate, and in this case someClass is going to be our selector and what's going to happen is any child of this that has this class on it, when it's clicked it'll bubble up. So the same premise here. We could have a span, and a p tag, and a div and they might be in this wrapper div here, and when any of those are clicked they don't have an event handler on them so what'll happen is it'll bubble up, but instead of bubbling all the way up to the document it's actually going to go up to our context that we set, which is our divs here. Then if one of those was clicked the click event handler gets called and we're off and running, so very similar to the whole live binding concept but more specific and you don't have to bubble up as far. Now the next one I've already shown earlier in this module and that's the on function. Now as jQuery grew we had all these different ways we could handle events: we have bind I showed briefly, delegate we just looked at, live we just looked at, and the problem is it got a little bit confusing on which should I use in, you know, which scenario. And so I've already shown one example of using on, very similar to this earlier in the module. Any div on a click, call this function, and that's kind of the standard way that you'll see on. But it truly is a replacement for bind, delegate, and the whole live binding concept. And so now we have one API to rule them all and it makes it much easier to just go with one thing and be able to use it. Now on was actually added in jQuery 1.7 so it's been out for quite a while, and it can be used for simple things but it can also replace the whole live and delegate concept. So here's an example of we're going to set a context note and it's almost analogous to what I just showed with delegate, but things are reversed as far as the parameters. So in this example we're going to say go find this MyTable ID and find the tbody under it. On a click of a tr, bubble up to the tbody and this function will now be attached to the tbody. So just like delegate we can set a context, but this one allows us to I think find a more readable solution. So on a tbody on click for a tr, invoke this function. Now we could also in the tr we could do that someClass I just talked about earlier. So this is the preferred way now to do this and so you really don't need to use bind, live, delegate, or those at all, because you can do everything with on. So as long as you're on jQuery 1.7 or higher, which odds are you are, then I would recommend you go with on here. Now you can also map on to multiple types of events and callbacks. In fact, you can even say in addition to having mouseenter and mouseleave call the same event handler function, which I showed earlier in this module, you can even map them to different callback functions. So here's an example of doing that with what we call a map. You'll notice that we say for a tr on and then we have an object literal here, so here's the start of it, there's the end of it. We're going to say when the mouseenter fires call this function, when the mouseleave fires call this function, then if I had more I could keep putting commas. So you could just put all these as one big stint on as the first parameter and have a single callback or you can do this map and actually map an event to a specific callback function, so very, very nice to know about and very handy and I think good for maintenance and productivity over all. So on can be used for simple things like clicks; it can be used for the bubble up concept, which I'm going to demonstrate here in a moment, and you can even map events to their own individual callback functions, which is really, really nice. So now that we've seen the different options let's jump into a demonstration and I'm going to focus on on because that's what we should be using, but keep in mind that you can do similar things as what I'm going to demo with the live or the delegate. So let's go ahead and get started here.

Using on() Demo

In this demonstration we're going to take a look at how we can use the on function to handle events without actually attaching an event handler to every single DOM element in our target. So I have a simple table here and you can see it has several rows. And ill switch over to design view here and you'll see that. So nothing real fancy and the standard way if we wanted to know when they clicked to handle this would be once we know the document's loaded we could go find our trs and then we could do click or if we want to use on we could say on click and then we can give it our function. And then inside of here we could go dig down into for instance, the different td's or something, but I'll just to show it works—we'll go ahead and say alert and we'll do this, which will be the tr and we'll grab the HTML. And so if we run this as we click on say, Tina Smith here, you can see Tina Smith is found and we click on Jim Leaner and it's found, and you get the idea there. Well what this is doing is actually attaching an event handler to every single tr in our table. Now I only have a handful here so it's not a huge deal, but imagine that we had quite a few. Well that starts to eat up into memory, and especially if you had a lot of these—hundreds or maybe even thousands—and so there's a more efficient way we can do this. Now the problem kind of worsens if I said I also want to handle on mouseenter and mouseleave, let's say. Because now we're going to have three different events and we're going to be attaching this function here for those. HTR now has kind of three things going on. And although it should work it's definitely not what we'd want to be after here; you can see it fired both those as I enter and leave and then I'd have to find a way to click them. Not very efficient and it works, but we have a better way we can handle this. So what we can do is instead of actually attaching into every single row we can go select a parent. Now let's say that our table had an ID of customers. Then we could come back in and we could say find the ID, so we'll go find customers and then I want to attach either to that, which would be fine, or I could go down even a little lower and attach to the tbody and I'm going to say on a click for what? And now I can say in this case a descendent that's a tr or I could even do a td. Now tads even complicate it more because obviously every row has two tds and if we did it the way we've learned up to this point and just attach something to every td, then now we're really talking a lot of different event handlers, especially if you had more than two columns. So now we're going to say for the customer's tbody on a click of a td and you can see this is a descendent of the tbody; call this function and we'll grab the HTML inside. In fact let's just grab the text because now a td would have John Doe and Jim Leaner and things like that. So we'll go ahead and run this now. And now as I click on Jim Leaner you'll see we get Leaner because that's what I clicked on there. And we can get Jim, Patoria, Sapna, and you get the idea. So a much more efficient approach now, because the tbody, which there is no tbody but the browser will add it for us—in fact just to kind of demonstrate that concept if we go in and get to the chrome developer tools—you can hit F12 to do that—and now we can come down to our table and notice there is a tbody there, and so it's just added dynamically by the browser and that's why this all works. And so now there is no event actually attached to the tds, it's to the actual tbody here, so very, very efficient. Now this is something that I personally like to do a lot. Any time I have, whether it's divs that're going to have children or some other type of item, this works great. Now, let me show you another example here. Let's say that we had a button and for this button let's say that when it's clicked we want to do something like add a new row, so I'll come in and let's give this an ID of we'll just call it addRow to be really obvious and we'll say Add Row here. So let's come on in and right up above here we'll go find addRow, and we could say on a click we'll add a new row into our table. So now I can come in and we can find our table; now I'm going to be doing the table twice here, so normally what I'll like to do is cache it so let's go ahead and say table = and we'll do customers. And the reason is I don't want to type customers twice in here and here. So what we could do now is we could go ahead and from our table we could find the tbody, and that'll go look in the descendents for the tbody, and then on a click of a td do something. But what I want to do here is say table., and we can append a new row in here. Now we know there's a tbody so we could certainly go find again the tbody. And in fact, we could even cache the tbody up here if we'd like. And so to make this even easier go ahead and just find the tbody up here and we'll just reuse it. So now we can just replace that with tbody on click and then here we can say tbody., and we're going to append in a new row so let's go ahead and just real quick make a new row here. And we'll do let's say Jane Doe. Okay, so every time we add a row it's going to actually add Jane Doe every time, but what's cool about this is I don't have to officially attach anything and I'll still be able to click on these newly added rows. So let's go ahead and run it and see what we get here. Okay, so we'll Add Row and notice we get Jane Doe and I'm going to add a couple of these. Now we're going to click and notice it still works, yet I didn't attach an official event handler to any of these; it's simply bubbling up to the tbody and then the tbody's handling it. So that's where this gets really nice because if it was a div and it had children in can now programmatically add children to the collection or if it's a table with rows I can add rows to the collection and I don't have to worry about adding the event handler, plus it's just better on memory all the way around. So that's an example of using on in a few different scenarios and it's definitely a great thing to know about if you work with a lot of children or descendents in a container.

Handling Hover Events

The final thing that we're going to talk about in this module on handling events with jQuery is hover events. Now I've actually shown a couple different ways to do this with shortcut functions and the bind, but there's actually a built in hover that we can use and it makes it really, really easy to handle the mouseenter and mouseout with a minimal amount of code. So let's take a look here at what we can do. So there's a shortcut hover function. So it's kind of like combining the mouseenter mouseleave functions; you can actually see that here in the second bullet. So you make your selector and get that all wired up so that it finds the proper DOM element, you then go in and call .hover and then you can give it in this case two functions: one to handle when you hover in or the mouseenter and one for the handler out, which would be equivalent to the mouseleave. So really simple to do; now there's different versions of this, defend overloads of this functions, so you'll see in a moment there's different functions we can pass for these different parameters. So the easiest way to use it is to simply call .hover and then pass in those two functions an you'll know that the first function is for when you mouseenter and then mouseleave on that second function. So in this example you can see that we're simply going in, we're doing a jQuery wrapper in that first function there around this ID selector called target. So we're going to find that DOM object, put it in a jQuery wrapper with this and we're going to simply change the CSS property of background color. Now when they mouseout or when the mouseleave is called officially then we're going to go in, call that second function, and we'll do the same jQuery wrapper and we're going to change the background color it looks like back to white in this example; so very, very simple to do. Now if all you wanted to do was change one particular style this would be a pretty good way to go, but let's say that you want to swap a class on or off of the DOM element. Well we already know how to do toggle class, I've shown you that actually several times throughout the course, and toggle class combined with hover provides the ultimate in ease of use and flexibility so here's an example of doing that. You can actually just pass in one handler, one function, which is the in and the out. Now this actually fires when they mouseenter and went they mouseleave and you could certainly detect that with the event object, but the wrapper here around this, you'll see we simply leverage toggle class. So it really doesn't get any easier than that, you know, you could do this with bind and if you recall with the bind you can pass in a string the function—or sorry, the event names—and so you could pass mouseenter space mouseleave. Well that certainly works and you can go that approach, but you know, with the shortcut here it's like it doesn't get much easier than this. So we'll call .hover; in this case we're going to do it for paragraph text and we'll simply toggle the over class as they hover in and out of any paragraph class within the page. So let me show you a quick demo of what that looks like in action and then we'll kind have go from there and that'll be the end of this particular module.

Hover Events Demo

In a previous demo in this module I showed how you can use the bind function to wire up multiple events with a single statement to a particular callback function. So we did that with the bind, you can see that example right here; so we bound in this case the mouseenter, the mouseleave, and the mouseup to this particular function. So you could certainly do that to handle the equivalent of a hover and it's really the same thing, or I could even use the shortcut functions mouseenter mouseleave and they would work as well. But if we really want to make it super easy to work with then we can handle the hover shortcut and use that. So let's go ahead and what I'm going to do is we're going to add a selector ultimately and then do .hover, but let me show you what we want to select. So we have this simple form we've been working with, which just has two columns in it to represent our labels and our controls. And so let's say that when they hover over these particular rows we want to put a kind of a light gray background on it; alright. So we can do that by simply doing the hover and making two functions. So coming on back up to the JavaScript let's go ahead and first let me show you the tables. So you can see the table is MyTable and then we have trs. Now that's the only trs in this particular page so we can just say tr if we want, but of course it's always faster from a performance standpoint with a selector to give it a starting point so we're going to go ahead and say go find MyTable and then go get the trs within that. And now we'll say hover. Now from here we simply pass in either one function, which I'm going to do in a moment or the two representing each type of function. So let me wrap these down so we can see them clearly; so this first one would represent the mouseenter equivalent. And we'll do one more. And this will represent our mouseleave. So let's say that we just want to put that kind of light gray background on it, so we can just simply take the object that was passed to us at this point, which would be this, go to our CSS and say let's change the background color and we'd like to change that to kind of a light gray here. And when they mouseout or call the equivalent of that, which is mouseleave, we'll go back to a white background in this particular example. Alright, so let's go ahead and try that out and you should see as we hover over the different rows it'll automatically highlight them and it's a really subtle highlight but its there. So you can see that now when they're in this it makes it really easy to kind of keep track of where you're at. Now in this example, you know, this is not so much a data type table, but you could certainly use this for when you have the equivalent of an Excel spreadsheet as an example and it makes it really nice to highlight that row that they're working on. Alright; now while that particular code works fine it's quite a bit more code than I might want to do unless you know, maybe I want to change just some individual styles—the cursor or something like that, then this would be fine. But if we move up top to our CSS area I have this light highlight and this is a little bit darker gray to make it a little easier to see, and what I'd like to do is show you the ease ire way to do this. So we can come back down and instead of doing this way let's go ahead and comment that out. Let's add a comment here: Using hover. And then we'll come on into here and we're basically going to do the same thing, so we're going to come in, we'll find MyTable tr and we'll hover, but we're going to do a single function this time. Much easier, because now all I have to do as shown earlier, is we can come in and wrap this and toggleClass and we can pass in that LightHighlight class. Now this is going to do the exact same thing—it'll be a little bit darker than this gray. Let's go ahead and try it out. So you can see now that as we highlight that it highlights those different rows really easily here. So super, super easy to implement. And you know, you have the flexibility of handling the unique mouseenter and the unique mouseout, or if you just want to kind of treat it, let it do its thing, and use toggleClass this is probably the easiest way to do it, so at this point I've shown you how you can use the shortcut functions; mouseenter, mouseleave; I've shown you how you could use bind—that was shown up here, and then finally we just looked at how you can use hover. So there's three and there's even more than that I'm sure I can think of another way, but different ways to work with this. I think the flexibility there is one of the strengths of jQuery—yeah, sometimes it's a little difficult to know, so which is the better way? Well in my opinion this is the best way if you want to do a simple hover, because I prefer to use CSS classes you know, whenever I can; it makes it nice and easy. Alright, now to end this video let me go and and show you one more little shortcut function that's kind of related. I'm going to come and comment this guy out. We're going to grab this code, and I'm going to do something that's similar here. Alright, so let me comment this back in. Scoot that over a little bit. Hover is obviously the mouseenter, mouseleave, but there's also another one called toggle. And that'll call other functions as well; in fact, you can have more than one function with a toggle. So every time you click it, it could do something different—click, click, click, click, click. So if I run this, now notice as I mouseover nothing happens here, but now I'm going to click on the row and you can see that it highlights it, when I click it again it turns it off. Now I don't know that that's real applicable to rows per say, but while we were at it I wanted to show how it really is using the same exact code, it's just now it toggles it based on when we click so first time we click this gets applied, second time we click that gets applied, and then let me do a third to show you how this works. And I'm just going to type in the name yellow here and make it kind of obvious. So now as I go in, and keep in mind this could, you know, probably be on a div or maybe a row of column or data, but I'll click, we'll click again, and we'll click again, and so you can see every function you put it toggles through those different options. Very nice when you need it. So that's just kind of a bonus one that looks like hover, but you know, it's a little bit different there. So that's an end to our section on handling events in jQuery, and specifically on how we can handle the hover, and then there's your quick look at the toggle shortcut.

Summary

In this module we talked about how jQuery can be used to really simplify our life when it comes to handling events in a cross browser way. So we talked about how in Internet Explorer IE8 really and earlier they handle events and attach to events quite a bit differently than all the other browsers. IE9 gets it right and makes it much easier but there's still going to be IE8 and earlier hanging around. So jQuery takes all that out of the picture. It provides several shortcut functions; we talked about some of them such as click, dblclick, mouseenter, mouseleave, and more. And provides functions such as bind, which is more of a low level API that allows you to pass in one or more event names that you want to bind to a particular DOM element that we found with a selector. What's nice about bind is you simply separate those event names with a space and it automatically will call the function that you define as either—either of them if there's two, or if there's more, any of them I guess—are triggered. Unbind we also talked about can be used to clear events or strip those off of a given DOM object. You can, if you pass no parameters, it strips everything or you can pass specific event names to just remove those. From there we went into the live and the delegate and these two were used to allow future children added into the DOM programmatically to automatically be associated with an event and an event function. The difference was live was in jQuery 1.3 and it bubbles events from those children up to the document object, which ultimately handles triggering the event and calling the event handler I should say. The delegate goes in and it actually allows you to supply a selector context so instead of having to bubble everything up all the way to the document object, which could be expensive depending on how deep your events are going, delegate allows this selector context so that it only bubbles up to that specific parent—hopefully that's just one or two levels up from these children that are ultimately leading to the event firing. So those are really, really nice though if you want to just hook things up once, and as I showed with the phone demo, allow different phones to be added as they click and not have to always add additional code to wire those new text boxes or whatever it may be off to an event function. The final thing we talked about was the hover and I showed how hover is really just a shortcut function, but it simplifies the mouseenter mouseleave and provides a really simple way to really to either pass it two functions representing both the mouseenter and mouseleave or one function and then you can combine that with things like toggleClass, so jQuery really provides a lot of simplicity and ease of use when it comes to events, and it even makes it easy to work with the event object, which we also talked about and get to some of those properties cross browser. So one of the things I've been mentioning throughout the course, I would recommend that if you get some time head off to jquery.com and check out the events section because there's even more in there that wasn't covered here; this is a kind of introduction to it to get you started and the more you know the more powerful you are as a developer, so I would definitely take the time to go explore what's there so that you know what's available.

Working with Ajax Features

Introduction

AJAX is one of the most important features introduced into browsers over the past 10 years, in my opinion. It allows us as developers to integrate a variety of data into our applications and make it so we don't have to reload the page every time a user clicks on something or blinks for the most part. And so in this module, we're going to talk about the AJAX features available in jQuery, how we can use those to get data from a server and integrate it back into our page using some of the features we've already talked about in jQuery. And we've talked about how to manipulate the DOM and how to handle events and use selectors to find DOM elements. So we're going to tie all that together and start working with some data that we're going to retrieve remotely. So the agenda for this particular module is we're going to start off by talking about the core features available in jQuery when it comes to AJAX, and specifically we'll focus on some of the functions that are there. From there, we'll talk about how we can get HTML data easily into an application. And this is really useful actually. You may just want to load help content as a user clicks on something, or you might want to actually load parts of a page, but instead of just returning say JSON data or XML, you're actually just going to return HTML that the server renders and then update the DOM with that HTML. We'll also talk about how we can make GET and then POST requests. And so GET requests are just, you do this every day when you type the link into the browser and hit Enter. That would be a GET request, or any time you load a script using the source attribute of the script tag, that's a GET request to the server. We'll talk about that, and then we'll also talk about post, which allows us to send more data to the server. So post is similar, but different way it works behind the scenes. And then we'll introduce finally the AJAX function. Now the ones I just mentioned we're going to talk about that allow us to load and get and post and those types of things, really they rely on, there's a kind of underlying function called AJAX that gives you all the power. It's the most low level API for AJAX, it gives you the most flexibility, but it does require a little bit more coding to use. I actually prefer AJAX in general and I usually will write wrapper functions around it because it gives me the ultimate in flexibility. So we'll talk about all those and then you can choose which one you'd like to use in your application when you want to exchange data back and forth between a browser and the server.

jQuery Ajax Functions

JQuery provides excellent support for performing asynchronous JavaScript and XML calls to a server, AJAX. Now the term AJAX is a little bit misleading now days because most developers aren't actually sending and receiving XML data from the server. Now you can do that, and some are certainly, but normally if you're using AJAX now days, you're probably using JSON data, JavaScript object notation, or maybe even sending just raw HTML data back and forth. Regardless, jQuery supports all of those. It does support XML data as far as getting the data or sending the data, JSON data and HTML, and it makes it very easy to work with. Now AJAX in a nutshell is simply a way to update parts of a page without reloading the whole page. So in a lot of apps out there, that are web-based anyway, you'll typically have to wait when you click a button or a drop-down or whatever it is for the entire page to reload. AJAX calls allow us to kind of take out entire page reloads out of the equation, and instead when the user clicks a button, we might just reload parts of the page. It makes our apps way more efficient as far as the end user goes, whether or not the performance is better, it definitely seems, the perceived performance is better, and normally in most cases it will be better because you're not reloading all of the data for an entire page. So throughout the module we'll be talking about some functions, which I'll show you in just a moment, that'll allow us to make AJAX calls. And the nice part about this is we can do it in a cross-browser way. Now IE6 definitely does it differently than all the other browsers, and we know that IE6 is still hanging around out there unfortunately. And as a result, we still have to deal with it. So the nice thing about jQuery, it abstracts all that out, we don't have to worry about what the browsers support, it tests that for us and then applies it appropriately so that we can either load an ActiveX control, if it's say IE6, or if it's IE7, IE8, IE9, Chrome, Safari, Firefox, and the others, then there's a built-in object that's available to use. Now to see why that's important and the differences, let me pull up a little page here that illustrates this. So here's an example from Microsoft on how things have changed between IE6 and IE7, and then on up of course. Most of the major browsers now days have a built-in XMLHttpRequest object, and so it's just a native object that you can call using the new keyword. Well, prior to IE7, you had to actually use an ActiveXObject. So you'd create a new ActiveXObject and then give it a different string depending on the version and things that you wanted to use. So jQuery automatically takes care of that type of code for us, and a lot of other issues that are associated with that, so that we don't have to worry about it, and that's nice. And what's really nice about it is that we can do this with just a few simple functions that we can call into. So let's take a look at those. So here's some of the functions that jQuery provides. We have the load. Loads typically use when you want to load HTML data. This could be useful, for instance, in a help scenario where maybe there's a little question mark icon, and when the user clicks it, you want to go out to the server, and based on what they clicked, grab the HTML content for that particular topic and then show it. Well, you don't want to reload the whole page just to load some help, so this makes it very easy to do it. We're still going to make a request to the server, but we're going to do it in a way that's very fast and performant. We also have get and post functions, and these allow us to do what they match up with. We could do a GET request or a POST request and work with it. If you want to get JSON data, there's actually a getJSON API function, and this is one I use a fair amount because I'm typically working with JSON data when I'm sending data back and forth from the server. Now when it comes to sending HTML, XML or JSON, JSON is definitely when it goes over the wire, pretty dang efficient. So something that's good to do. Now it does mean when you get the data back in your script, you're going to have to write code to pull the data out of the JSON, which is very simple, I'll show you that coming up in the demos, but you'll have to do that, pull it out, and then update the DOM. Now the final function is the kind of low level API, which is AJAX. Now I use this one probably the most out of any of the others, and the reason for that is it gives you the full power available to use. So AJAX allows you to specify things like content types and what's the URL to call, what type of data am I getting back. You can specify different callbacks for errors and success and stuff like that. So when we walk through these, I'll show the differences and explain how they work. Now all of these can be used to call REST APIs, you could even call web services if they return the appropriate data for your application, or just native HTML URLs, if that's what you want to call as well. So let's delve into the different functions here and I'll show you some demos and examples of how these different functions can work. We're going to start off with the load function. So let me talk about what it does and then I'll show you how it works.

Loading HTML Content from the Server

JQuery provides a really simple way to load HTML content from a remote server, and you can do this through the load function. So in this section, we're going to talk about the load function and the different parameter values that can be passed in. So here's an example of the signature of the load function. You can see that we can pass in a URL, we can pass in some data, which you'll see in a moment could be something like JSON data that gets passed up to the server, and then you can have a callback function, which we'll see at the end of this section. Now the only one of these three that you have to pass is the URL. So let's look at an example of that first. So moving on down, you'll notice we're going to handle the DOM ready, and then we're going to wire up a button and the click event specifically to this callback function. Now when that button is clicked, we're going to go ahead and go out to HelpDetails.html, you'll notice this is a relative URL, so it's going to load it relative to the current page where this is running, and when that data comes back, we're going to place it inside of this MyDiv ID selector. So with one simple line of code, jQuery can go out to the server, grab the HTML data, and then automatically append it to my div here. It makes it really nice and easy to work with. Now there may be cases though when you want to do a load, but you don't want to put the entire content of this HelpDetails.html. A real life example would be if you had some type of help documentation in HelpDetails, then you might want to filter that based on what they clicked on to give context-sensitive details. So an example of that is shown here. And you'll notice that we do the standard load. So we have the ID selector that determines what the data, where the data I should say, goes into, my div, we call .load, we give it HelpDetails.html, but you'll notice that we do an ID selector on MainTOC. Now this is the context of the selector, it's not the document we're currently working with, it's the document we just loaded, that's why it's in the string here. So what's going to happen is it's going to go out, load HelpDetails.html, then filter that data by this ID selector, place the return result, all the nodes that are returned into the children of my div. So it makes it really easy to filter. Now, it's not going to do this at the server, it is going to download this HTML document, then it will filter based on my TOC and apply it to MyDiv. So unfortunately it's not going to be a really efficient thing if you had a huge, huge page because it is going to download the entire page and then it will do the filter in there, but it is fairly useful if HelpDetails just needs to be downloaded once and cached, then you could apply different ID selectors to it and it makes it pretty interesting. So really, really easy to work with you can see. Now the second parameter you can pass is the data. So data can be passed to the server in cases where it needs to return something based upon that data. So in this example, we're going to call Customers.aspx, and by doing that, we don't want to return all the customers. This example, which I'll run in the demo coming up will simply return columnar data. So customer data in columns, but we only want to return 25 customers in this example to be placed inside of MyDiv. So what this'll do, is you can see the second parameter, you pass in a JSON object, we have a PageSize property with a value of 25, that'll be passed up to GetCustomers.aspx. In this case, C# code will then analyze the page size and determine how many customers to return back. Now keep in mind this could be PHP, this could be Python, Java, it doesn't really matter. It works with anything as long as it can return HTML data to you. So really useful though if you need to pass some data up to the server and the server expects that data, and you can of course pass more than one property, since we're only showing one in this example. So that's the second parameter, the data. Now you can also go in and have a third parameter, which is optional as well, which is a callback function. Now this one uses two of the three parameters. So you can see in the code here we have this NotFound.html that we're going to tie into. Now, that purposely is named that way because it doesn't exist on the server we're going to assume, so we just want to demonstrate in this example how we can have a callback function that can check the status of the message, and if it's an error, then we can grab some details about that error. So the three parameters passed into the callback function shown here are the response, the data coming back, status, which is going to be if we're an error or not, and then the XMLHttpRequest object will have details about the error if there is one that occurred. Now in this case, since NotFound isn't available, then this will definitely give an error, the function will be called either way, success or failure, we'll detect that we had an error, and then we'll ride out the status message to an alert. So really nice in situations where you need to determine if it worked or not, and then write your code to handle the situation where maybe it didn't. And so let's take a look at a demonstration of how we can use the load function with some of these different parameter options.

load() Function Demo

Let's take a look at how we can load HTML content from a remote server into our web application with just a little bit of code using our load function. So you can see I have a reference to jQuery, and at the time of this last module, jQuery 1.5 had just come out, so we're going to show that, and we're going to use our standard ready. So we'll say document.ready, we'll give it our function, and there we go. So now what I'm going to do is when this button is clicked with an ID of MyButton, I'd like to go in and grab some HTML that's going to be located in this HelpDetails.html file. So the first thing I'm going to do is add a selector for MyButton, so let's go ahead and do that, and then we'll add a click, and when it's clicked we're going to execute a function here as well. Now I want to do the load. So I'm just putting a comment and say Load HTML from HelpDetails.html. So let me open this file and you'll see that this one actually has the body and the head and the HTML tag striped out, we just have the core HTML that we need mainly because if this is only going to be used for loading parts of a page, then there's really no reason to load all the other stuff because it obviously adds to the HTML payload, so we want to keep that minimal. Now this one is pretty simple. You could see that we have a div with a MainTOC that goes to there, and that just links down to some anchors here. Alright, so it'll load that, we can just use our simple load function, it's about as easy as it can get, and I want to load it down into here. So we can come in, we'll do an ID selector on that, and we'll simply say load. And you'll notice that I'm only getting a, b, c as the parameters. Now we happen to know what those mean at this point, we have the URI, we have the data for the b, and we have a function callback. Now the reason I'm not getting the full IntelliSense is for this particular project I don't have the vsdoc file, so you can see how it really adds a lot if you looked at the demos in the previous modules because in those I have the vsdoc file in place, so definitely you probably want that. Now, what we're going to do is pass in the URL, so I need to go up a level. I'm going to do this relative to this current page, and so we have UsingLoad, and if we go up from there we can get to this particular guy. So we could go from either the root of the server or we can just go up, I'm going to keep it simple, and so we'll say HelpDetails.html. And that's it, so we're only going to pass one of the possible three parameters that you can pass in the load. Now if we go ahead and run this, we'll click the button and you can see how fast it loads that. Now it's local, so of course it's going to be really fast, but it's still very good performance and it automatically updated the OutputDiv for us and didn't have to write a lot of code to do it, very, very nice. Now the other thing that you can do with this, as I showed earlier, is we can come in and let me open HelpDetails, and let's say that we only want to load this particular div, or maybe we have another div in here. In fact, let's just do that, let's just add another one and we'll add an ID of, we'll call this SubTOC, and I'll just put This is my sub table of contents. So if we only wanted to load that and display it, we can use an ID selector of SubTOC along with a URI, which is really kind of clever actually. Now keep in mind it is going to download the entire file, so that part doesn't change, there's no way to influence that without doing some server side magic there, but at least when it downloads it, especially if it's cached, then we're going to be able to grab just this piece. So let's go ahead and do that, and all we have to do is inside of the string for the URI, we can simply put our ID selector of SubTOC, very, very simple. So now we'll go ahead and View in Browser, and there we go. This is my sub table of contents. So what's nice about this is at this point it's cached, and the reason for that is this by default, as I mentioned earlier, is a GET request. Now if I do pass in data as the second parameter, we pass in some type of JSON object, then it's going to switch over to a post, and then it won't be cached, but at this stage it is cached, so now it's going to be really, really performant, really fast to work with. So that's a super easy thing we can do there. Now we can also come in, and as I showed earlier we can load with a callback. So I already have all the code done here, so we're going to hook up our HelpButton to a function when it's clicked, but when we load, NotFound.html is nowhere to be found actually, that's why it's called NotFound, and as a result, we want to be able to handle that error. So this'll call the function that you see right here. So what'll happen because it's not found, we'll get a particular error. So let me go ahead and run it. Hit Get HTML and you can see Error is error. Now just to show that that actually got called, there's Error plus the statusText of our XMLHttpRequest object. And there's even more details we can get through that object as well, but this provides an easy way to see, do we have an error or not, and then we can act upon that, so pretty simple to use. The other thing we can do is pass data up. Now this example calls a dynamic page, it could be PHP, it could be Python, it could be Pearl, it could be whatever it is you're using on the server side, but the server side in this case is an ASP.NET page and it accepts a page size of 10. So this is our own custom JavaScript object notation object and you'll see GetCustomers right here, so let me show you that. This uses a DataList control that's part of ASP.NET to simply write out dynamically some trs and tds columnar data based upon customers. You can see we have a HeaderTemplate, an ItemTemplate representing each row, and then we have our FooterTemplate here. Well that part is specific to ASP.NET, so that may change obviously if you're doing PHP or something else, but we can come into the code behind here, the actual code that drives it, and what I want to point out is when the request comes in to this page, we're going to grab the PageSize. Now if there isn't one passed we're going to default to 50. Otherwise, we're going to take that PageSize that was passed in, convert it to an integer, and then we're going to use that to just really render some fake customers here. We'll have John 1, John 2, and so forth. We'll then take those customers, we'll bind it into our DataList, and then output the HTML through these calls right here. So if we run this, it's nice because we're able to hit a dynamic page even though this is just the simple load function and we're able to pass up our PageSize. So let me go ahead and run that. And now that it's loaded let's go ahead and click Get HTML and there we go. So you can see we passed up the page size and it returned the specific number of objects here based upon, let me go back to that, based upon the loop that we were doing. So we're actually doing while it's less than the PageSize. So, very easy to change that. We can come in, let's go ahead and up that to say 25. And if we run it now, you'll see that we get the appropriate number of customers while it's less than 25. So, very, very nice, very easy. So load is not just for hardcoded HTML pages, you can also use it for dynamic situations, and that's why the particular data parameter here that the load accepts is really useful because now we can pass in some different data based upon what the end user wants to see. So that's an example of using the load function. Very, very easy way to get HTML into your applications.

Making GET Requests

The load function provides a great way to load HTML content from the server, but there's going to be times when you want to be more flexible. You might want JSON data, XML data, HTML or just even simple strings of text. So in this section we're going to talk about another function that's available in jQuery called get, and get gives you more flexibility for making these GET requests as the name implies. So you'll notice in the signature of the get that we have $.get, so you don't give it a selector first, you say really it's like doing jQuery.get, the jQuery object has this get function hanging off of it. So we call it a global function. And the first three parameters are the same, the URL, any data that gets passed up, a callback function, but we also have the option if we need to pass in the dataType, and this can be XML, HTML, JSON or even raw strings, and you can even do scripts this way if needed. So in this example, we're going to use the get function directly. So again, notice it's a little bit different than some of the, most of the stuff actually we've covered in the course because usually we do it through a selector. Well in this case, because get is a global function off of the jQuery object, we can just call it directly, we're going to say go grab this HTML, but instead of the load automatically updating the HTML, we have to write some code to do that because after all, we didn't give it a selector on where that HTML should go. So you could see that we have a callback function, which passes in the HTML data, if that's the case, which it would be with this one, and then we're going to go do an ID selector and update ultimately the innerHTML with the html data that was returned. So with get, you have to write a little bit more code, however, you could still do a GET request just like a load does by default, but you have more control over not only the dataType, but also how the data that comes back gets updated. It provides a very useful feature that's simple to use though. It's interesting to note while we're on this, obviously a get does a GET request, but a load by default does a get, however, if you pass in data as the second parameter, it'll actually fall back to a POST request in that scenario. So one of the useful tools I like to use is the jQuery source code viewer that I want to show you real quick because sometimes you might want to know what are these jQuery functions doing behind the scenes. For instance, how did I know that load is hardcoded to always use HTML data and how's that look versus get? So let me show you an example of that. Now you should always go to the HTML documentation through jQuery at the api.jquery.com because that'll provide some great details on the parameters and things that are passed. And so if you go there and you want to use the AJAX functionality, if we go back to All, we'll see what you would see by default, we'll hit on Ajax, and that gives us all the AJAX features. And here's get, here's the four parameters we were just talking through. Now the jQuery source viewer that I find useful is something I use in times where I read the docs, but I'm a little bit unsure about how one particular parameter is used. Well, if you go to this URL here, and I'll probably say the last name wrong, but james., I'm assuming it's padolsey.com/jquery, then you can get to this jQuery source viewer. And this actually allows you to type in functions, even select versions, and then it will go in and provide the source code of what is jQuery actually doing behind the scenes. Now this is also a great way to learn scripting techniques, but we won't bother with that now. What I want to show you though is that behind the scenes, it looks like the get is actually making an AJAX request. Now you'll also notice it hardcodes the type of the request to be a GET request, but the dataType of the response that's going to come back is not hardcoded. That comes up from that fourth parameter if it's supplied. If it's not, it'll figure it out, or try to, on the fly. And then you'll notice we can pass in the URL, any data, and then here's our callback. Now we're going to be talking about the AJAX function later in this module, so I'm not going to go into details here, but you'll see that all the AJAX functions available in jQuery are going to be relying on that. Now if we go to the load function, which we've already looked at, you'll see with the load function that here's those three parameters, and if I scroll on down a little bit, it also uses the AJAX function behind the scenes, but the dataType is hardcoded to be HTML. So whereas the get function we're talking about now allows flexibility there, you can pass HTML, XML, JSON, even a script or a string as the dataType, the get is way more flexible whereas the load, it hardcodes it to be HTML. So I like to use this particular source code viewer to kind of see what's going on behind the scenes, learn a bit about what jQuery is doing, and it's a great learning tool as well for learning some nice JavaScript techniques. Now in addition to the get function, we also have a global one called getJSON. Now getJSON pretty much does what is says, it uses the get to grab some JSON data from the server. Now you'll notice the parameters are identical to the load. So we have the URL, the data to pass up to the server, if any, and then a callback. Now those are illustrated here. So we call through the jQuery object, we call getJSON, and now we need to pass it the API, the docs, whether it's Pearl or Python or ASP.NET, the document or web service or REST API, whatever you want to call it, that is capable of returning this data to us. So in this case, CustomerJson.aspx is going to return JSON data of course, we're going to pass up to the page an ID property of 1, so it'll grab 1 customer. Once that data is returned back, that'll be passed into our callback function as this data parameter, and because it is JSON data, we can just get to the properties of the JSON data directly. So, we have a FirstName property and a LastName property it looks like in this scenario, which will simply pop up into an alert. So it's a very, very easy way to work with JSON data. Now you could certainly do the same thing with the get going back to here. Instead of passing HTML, I can definitely pass a page name or a service, a REST API that allows me to get back JSON and process it, but if you're always just going to be working with JSON directly, this definitely is a little bit easier to work with in that particular case. So let's look at an example of using the get and the getJSON functions with jQuery in some code here.

get() Function Demo

The get function provides a simple way to perform a get against a server that provides up some data, and so I'm going to show you an example of how we can interact with that and use it in an application. So you can see that we have a button here, and this will be a pretty common theme throughout the AJAX demos, you've seen this in the load, we'll see it in the post, and even in the AJAX. But when that MyButton is clicked, we want to execute the function here, and so what I'd like to do is call the global function called get, and we're going to call into this HelpDetails now. If you saw the load tutorial, the demonstration then you've already seen this, it just contains some simple data in it that we want to display. So if we want to go ahead and do a GET request instead of using the load, then we could simply put in the URL, so I'm going to go up one level with the dot dot there, and then we're going to give it a callback. And inside of here, when the data comes, unlike load, which automatically loads it into the selector we would have supplied, we have to tell it. So we have the OutputDiv here and we're simply going to select that with an ID selector. And we'll update the HTML with the data, so very, very simple. Now, not as simple as load, but we do have a little more flexibility because when we get back this data, we could certainly do something with it instead of just updating it, but let's go ahead and run it. And when we click the button here, we should see the HTML content come back and we're pretty much in business, we're ready to go. We can also go in and we can use the get to get other types of data. So you'll notice over here I have a Customerjson.aspx. And as mentioned throughout these, it could certainly be PHP or some other format if you'd like, but what we're going to do is simply set the content type being returned to JSON, we're going to take a customer, and we're going to serialize it to JSON data and simply write it out to the Response.OutputStream. So we're going to take this customer, take an ID that's passed in, add it to the ID right here, and then return it. So this is C#, but in essence we're just taking an object in memory and serializing it down to JSON. So if we wanted to call this using the get, then we can do really the same type of approach. So I'm going to copy this, except we're going to use all the different parameters here. When I type in the get, you'll notice it takes four parameters, as was talked about. So we can pass in the URI, we can pass in data, we can pass in a callback function, and we can even specify the type of data we want to deal with. So we have the URI, but I'm going to go ahead and change that now, and let's change that to CustomerJson. I'm going to pass in data for the ID and just to kind of confirm, I'm actually going to match the case there just to keep it consistent, and we'll do an id of say 5, there's our callback, we'll go ahead and do something with that. And then finally after the callback, we're going to say we're going to work with JSON data. So now we can come in and once we get this data back, we can obviously do something with it in our callback. So for this example, let me just go ahead and just do an alert to start off and let's do data., and we should be able to get to the FirstName property that was specified right here, which should be John. So very much like what we saw up above, the difference being that number one, we're going to supply some data, number two we're going to specify the type of data we want to work with. And actually it looks like we need to fix that. So let me go ahead and run this and you can see we get back John. So as expected, we get the data and we're pretty much good to go. Let's just go ahead and fill this out all the way and we'll do data.LastName. And let's also put ID, we'll do data.ID, and let me just wrap that down a little bit. So let's run it now and now we should be able to see that the ID should be 5, so it will basically roundtrip the ID from the client up to the server and then back. And there we go, we have ID 5 and we have John Doe. Now, that certainly works, but after all we did talk about a getJSON function, so let's go ahead and take a look at how we could simplify this and not have to specify for instance this parameter if we just want to get JSON anyway, it's a lot easier to work with if we just use that function directly. So let me go over to this particular area and we're going to do the same type of thing, when the button here is clicked, we want to go in and do something with it. So we'll just pop up an alert again in this example, but we're not going to do get, we're going to do getJSON. And so I'm going to call that same exact CustomerJson.aspx that I saw earlier, we'll pass in an id, we'll do 5 again, and then from there we need our callback function. So we're going to get our JSON data back and we're going to go ahead and end that right here. So inside of the data, now it's the same thing as you just saw, so we can come in and we can do an alert. In fact, let me just go ahead and grab what we've already done since we already know that's the type of data, and we'll pass that in. So it's pretty similar overall of course, but we don't have to specify the parameter here. In fact, let's take a look at the signature and you see we only have three, we have the URI we want to call, or URL, we have the data, that's the b, and then the c would be our callback function. So let's go ahead and run that, and same thing. So, a little bit simpler, but really the same effect as calling get. I'm a pretty big fan of hey, if you don't have to type extra code, let's not type extra code, and you can see that keeps it pretty easy to work with. So that's an example of using get and getJSON to be able to call and get different types of data in our applications.

Making POST Requests

GET requests are really useful in jQuery and through the get, the load, or the getJSON you can make those, but anytime you make a GET request, it's going to be cached in the browser by default if that's how it's setup, and you are limited on the amount of data that can be passed up because the data is going to actually be encoded and placed on the URL. Well, sometimes you might want to make a request that there's a lot more data involved, large amounts of data are posted up, in that case you probably are going to want to go with a POST request. Now POST requests are not cached in the browser, so when you make a POST request in jQuery, there won't be any caching going on due to the nature of a post, but you do have that advantage of being able to post large amounts of data, just like you do with a form, you might fill out a large form, and that's going to be a post. So the post function, the API looks identical to what you're used to with the get and the getJSON that we just saw. You're going to have a URL, the data, and the callback. Now all three of those are used with load, get, and getJSON, and then of course the get also had a datatype you can pass in, which we've talked about can be XML, JSON, HTML, and even some others. So the call itself looks identical to the get. In fact, you'll see it's a global function right off of the jQuery object, but in this case we're going to actually post PageSize of 15, this JSON object, up to GetCustomers.aspx. In this case it looks like it's going to return html data based on what the code is doing, and we also didn't pass in a datatype here, so jQuery tries to figure that out for you. So the data will come in, we'll go ahead and update the innerHTML of this particular div. So that's pretty standard in what we've seen to this point and really just almost identical to what we did with the get. Now let's look at some other examples of post here and what you can do with it. Post can also be used to hook up to, if you're doing .NET, services such as WCF Services. And I want to point out something that's a little bit unique with this. So in this example, you'll see we're going to post data up to a Windows Communication Foundation Service, or WCF Service. We can tell because of the .svc extension. Now if you're not doing .NET, you can probably skip forward here and jump right to the demo, but for those of you that are, when you call a WCF Service, you can mark the service using some different attributes as JSON-enabled operations. So this GetCustomers is what we call an operation, and what it's going to do is by default return SOAP, Simple Object Access Protocol, but you can also configure it with WebGet and WebInvoke attributes to return JSON data. Now, by default though, when WCF, if you just hook it up the normal way, returns JSON data, when this callback gets passed in, you'll notice that we say data., but we're going to go to this d object. Well, what'll happen is the JSON data that's returned will be wrapped with this d object, so d will actually be the name of it representing I suppose data, and we're going to say let's go into the data by going to the 0. So in this case, once we get that, we're going to store that as our cust object representing our customer, and then we can get into the FirstName and LastName, and that's something that you definitely might have to play with if you've ever called into a WCF Service and can't figure out why you can't get the data properly. Now what I like to do to figure that type of stuff out when I'm having problems is use a tool called Fiddler, and let me pull that up real quick and show you how you can get this free tool. If you pull up your browser and go to fiddle2.com, you can get to the Fiddler site. The tool itself is actually a desktop app. It's one that I use a lot. In fact, you can see right down here on my toolbar I have Fiddler, and what Fiddler is an HTTP proxy tool. It allows you to inspect traffic as you're making AJAX calls, or really any HTTP calls, and it will show you what's going on with the messaging. It's really easy to use. So, of I pull it up, in the user interface you'll see a request will be made right off the bat out to the Fiddler site to check the versions. You can see this updatecheck.asp isBeta False, and so over on the left side here, this is where all your requests would be made. So if we make an AJAX request, then it'll show up in here, and then if we go over to TextView, we can actually see it doesn't look like any data was sent up other than the isBeta False on the querystring, but if we come down into TextView here, we can see that some information was returned back, which looks like it's the version data for the most current version of Fiddler. So you can actually use this to inspect your messages, and in the demo coming up, I'm going to actually show you how to use that with your POST requests. It makes it really easy to inspect your data. So POST requests are really useful any time you want to pass up some data to a service, which you can get away with a get, but sometimes you might have nested objects and it's difficult to mark that up in a way that it fits on the URL, or you might have large enough amounts of data that you just can't trust the fact it's going to fit on the URL, some of them might be truncated or chopped off. So, now that we've talked about load, get, and post, and specifically focused on post here, let's look at an example of how it can be used within a web application.

post() Function Demo

So up to this point in the module, you've seen demos on how to use the load function to use HTML and how to use the get function when you want to make a GET request to the server. But there may be times when you want to execute a POST request. Now post, as I mentioned earlier, doesn't allow the actual URL and the contents to be cached by default, and that's just how a post works. The benefit though is you can post large amounts of data whereas when you do a get, you're limited to what can be passed on the querystring, and there is a limit to that. So to demonstrate the post, we're going to do really the equivalent of the get, very, very similar to what I did earlier. But let's go ahead and we're going to call into GetCustomers.aspx, and just as a refresher, when this particular page on the server runs, it's going to check up on the querystring, or in this case in the request collection because we're doing a post, and it's going to say hey, do we have a PageSize. So it'll either check the querystring or the form collection. In this case, because it's a post it'll be in the form collection. And if there isn't one, we'll return 50, otherwise we'll convert that to an integer and we're going to use that to just simply create some customers on the fly here. So it'll return an HTML table. So let me go up to GetCustomers.aspx, we're going to pass up that PageSize I just showed you, we'll just do 15 in this case, and then we have our callback function. Now this is going to pass us the data. So, we'll come into here and now what we want to do is simply grab our OutputDiv, do an ID selector on that, very simple, and we're going to update the HTML with the data because that's what this page returns is some HTML data. So if we go ahead and run that, let's do View in Browser, and you can see it works. Now I mentioned in the section about posts that there's some nice tools available out there that you can actually inspect the traffic going back and forth, and one of those tools is Fiddler. So, what I'm going to do is open up Fiddler, and as I mentioned this is an HTTP proxy tool, so it allows you to actually view the traffic going back and forth between the client and the server, the browser and the server. So I'm going to go ahead and let's remove the call that Fiddler first makes to check the version to make sure it's the latest, and we're going to go ahead and just minimize it, but keep it going, and let me reload the page and hit the button and let's see what we get here. We get nothing, not good. Okay, well that's actually by design in this case. What's going on is the way localhost works, Fiddler can't tie into that, it's an internal IP to the operating system, so it doesn't actually see that traffic. So there's a couple options here. One is I could use my machine name if I had it setup to support that. In this case though, I'm going to use the second trick, and the second trick is you simply put a dot to basically trick it so that it still works on localhost, but it doesn't really think it's localhost, it's I don't know, a quasi localhost I guess, and I'm going to hit Enter. Now this will load up and let's go check Fiddler at this point and there we go. So there's the request being made, there's also a request out to look at the favicon, we don't have one, here's a request to get jQuery, and you can see what it's requesting right up here in the Request Headers. So you can see we have a GET request, another GET request, and then it does a get on the favicon, which doesn't exist. So let me remove all those now. So we'll come back and now let's hit Get Customers. It'll go out to the server, return them, and let's go back to Fiddler now. Okay, so there's our call. First off, it's a POST request, and of course that makes sense, we're using the post function. And if we look down we can see some information, it looks like this was a form-urlencoded content type, it was made using the XMLHttpRequest object, here was the Referer, which is the page we're currently working with. So, if I switch to TextView, we can actually see the data passed from the client up to the server. And there we go, PageSize of 15, very simple. So this is the request area, this area down here is the response area. So I can hit TextView and here is the HTML response we got back, and you can actually hit View in Notepad if you want to pop that up and see it a little bit easier. Now that's what Fiddler does out of the box, it just allows for a really easy way to see the traffic between the client and server. And again, you can go, if you go to Help it'll show it here, but you can go to fiddler2.com, and it was created by Eric Lawrence who, as far as I know, still works up at Microsoft, but fiddler2.com. Let's go ahead and clear that out, let's leave it going, and we're going to move on to another example of using post. So the next thing I'm going to do is I showed how we can call into an ASP.NET page, or it could be a PHP page, it doesn't really matter, but we returned HTML data. Now let's go into UsingPostToCallAService. Now this is going to be a WCF Service, Windows Communication Foundation. Now keep in mind it could just as easily be a Java service, it could be based upon Python, there's many different services of course supported out there. So I'm going to open this up, we're going to right- click on our .svc file, which represents our service, and go to View Code. And you can see that there's two operations supported. We have a GetCustomers and then we have this insert customer. Now we're going to demo the GetCustomers in this particular example. So it just hardcodes a list of two customers, so we have John Doe and Jane Doe, that are going to be returned, and we're simply going to write out the first customers, which will be John of First and LastName. So, although you can certainly just call into more of a REST API, or just a page that returns some data, you can also call into services that support this type of transaction between the client and the service, or operation I should say. So let's go ahead and go back into here and what I'd like to do is when we go in and the button is clicked, so let's go ahead and add that back in, so we'll do MyButton (Typing), and now we're going to do our global post, so we'll say post, but this time I want to go to the CustomerService.svc. I don't just want to go to that file though, because remember we have two operations, so I want to call GetCustomers. And the way we do that is we simply put a slash and we call GetCustomers here. Now in this one I'm not going to pass any data up, and the reason for that is you'll notice that GetCustomers doesn't take any data, so we're going to pass null for that, but I also want to come in and we're going to do a callback function, which is going to be the data. So let's go into here and what I'm going to do now is grab the first customer, as mentioned, which should be John Doe, and because this is a WCF Service, it'll actually wrap it in a JSON object called d. So I'm going to say data.d, 0, and that's going to be my first customer, or I could have done 1 to get into my second customer. Now to show this works, we're just going to come in and we'll come into our OutputDiv and we'll simply write out to the html cust.FirstName plus cust.LastName. Now the reason I can do that is if we look, look at the properties here that are returned from the service, now this is C#, but they'll be serialized for us. And what'll happen is we'll get back JSON data that'll have those same properties inside of it, that's why I'm able to access these directly. So it's really, really easy to work with. The catch is knowing about this if you're working in the .NET world and using WCF Services. Okay, so we have our function, let me go ahead and end our bracket here. Now if I run it as is, if you recall, jQuery will try to figure out the datatype, but we actually want it to be JSON data. So what I'm going to do is leverage the fourth parameter and I'm going to pass in, in fact you can see that fourth parameter there in the IntelliSense, let's see if I can get that back, there we go, and so I'm going to pass in that I'd like this to be JSON data. So there's my post, we're going to call up to the service, we're going to tell it which operation to invoke, we're not passing any data, here's my callback function, which will simply update the OutputDiv with the FirstName and the LastName, and then we're going to explicitly say that this should be JSON data. So I'm going to leave Fiddler going here, and you can see it's empty right now. Now if we run it as is, as I just showed earlier, you're not going to see a whole lot, but let's make sure that everything works, and you can see John Doe is returned, so that's perfect, that's what we wanted. But if we go back to Fiddler, we don't see anything because of the localhost thing I talked about. Let's go ahead and we'll fix that, let's put a dot, we'll hit Enter, this'll load, and now we're going to invoke it, and John Doe is returned. Now let's look at what Fiddler gives us. So this is the call when the page loaded. In fact you can see the page right there, here's the call when we actually did the jQuery script, let me return or remove these guys, and here is the call, which is a post, which is our post operation. Now you can see what it's calling here, CustomerService.svc/GetCustomers, you can see that it's application/json as far as what the client is passing up, and that's because we did that ourselves of course. Here's the data being passed back. So we have, let me open this in Notepad, we have JSON data. Now as I mentioned by default, WCF will wrap the data returned in this d object. So you can see we have d:, then we have an array, and the array ends right there, so we have the start and the end of it. Now the type of data in the array is a customer object, and you can see that it has a FirstName of John, an ID of 0, LastName of Doe. We also have another type called, which is a customer, FirstName Jane and LastName of Doe. So the trick here, if you're doing the things in the .NET world and you want to use WCF Services out of the box, if you see this behavior and you can't figure out why, then let's go back to the code here, you can't figure out why you can't just say data 0 .FirstName. Well, you might want to pull up something like Fiddler to check out what the JSON is, and that way you'll know if there's a d that you have that wrapper object to worry about. Now there is a way to change that, that's beyond the scope of this course, but you can make it where it just returns the raw JSON without the wrappers, that is possible in WCF. So that's another example of using the post. That's also an example of using Fiddler, which is an excellent tool, I use it for everything from AJAX calls to working with Silverlight to working with anything that exchanges data across the wire.

Introduction to the ajax() Function

The final function we're going to talk about available in jQuery that's related to AJAX is actually called ajax. And if you're a Lord of the Rings fan, it's kind of the one function to rule them all. As I showed earlier, the load, the get, getJSON, post, all of those rely on the ajax function behind the scenes, so it's the heavy lifting one that does all the actual work to communicate with a remote site. So the ajax function has a lot of different properties you can pass in, and by properties, you're going to see I mean JSON properties, but here's a few of the key properties you can pass when you call this function. So you can pass what's the contentType of the data going up to the server, is it just multipart form-encoded, post, is it JSON, is it XML, what do you have there. You can also pass what is the data that you're passing up. This could be simple strings, it could be JSON data, those types of things. What's the dataType of the data coming back. Oftentimes with ajax-enabled apps now days, it's JSON, but it can be other things like XML or HTML. If there's an error, there's a callback you can hook up, a function that you can assign to the error property. There's a success, when it was successful and when it's just totally done, there's even a completion you can pass. And you can even influence the type of request made, is it a get or is it a post. Now an example of using some of the properties is shown here, and there's a lot of properties, I'll show that at the end of this section, but here's some of the key ones I just mentioned. So we have the url, but you'll notice that these are marked up in a JSON object, there's the start and end brackets for that object. So the url is of course what are we calling to get our data. This could be a REST API, a web service, whatever it may be. The data that's going to be passed up, well in this case, customer is a JSON object that you'll see in the demo, and this allows us to pass up the data to the server just like you saw with the get and the post. What's the dataType of the data, of the data being passed back from the server. It's going to be json data in this example. If it's a success message, then you can pass that a function. So in this case we're getting back the data that's returned, the status, which is it an error or not, and then you could get even more details through the XMLHttpRequest object, which is this third parameter. Now in this case, because we're calling a service, which is a WCF Service as I mentioned in the post section, it actually wraps the data with this d object in many cases. So in this case we're returning a single object that has a status of the insert operation and a message because it might have failed about the insert operation. Now it was an array, we could get d, do the brackets and get to 0, 1, so on and so forth as I mentioned in the post section. Now you can also have an error callback, which is very useful, and of course something you typically want to have, and so this allows you to get to details about the error and either notify the user through a pop-up alert or maybe some other UI element that you want to show. Now there's even more JSON properties for this ajax function that you can work with, so let me pop up the jQuery documentation page real quick and talk through a few of those. So if you go to the standard api.jquery.com page and then click on the Ajax item on the left here, you can get to all the ajax functions and then ultimately get to the global ajax function. So, the ajax function only takes one parameter, and as I just showed earlier, that's a JSON object that has these properties. Now there's a lot of stuff you can do with this. You can set whether or not the call is async, which I would recommend leave that as true, but there may be a few times when you need to change it. You can say if you want to cache it, before the data is actually sent you might want to do something to the headers that are sent up or you might want to do something to the user interface right as that call is about to be made. There's a completion callback. So we have success, we have error, and then you have this complete, which lets you know when everything is all done. You can set the contentType of the data being sent up to the server. You can come in and set the data that's passed up, there's even filters you can apply, the dataType coming back from the server, and you can see the different options that can be passed there. What's nice about these, and I mentioned this is the get and post, is if you don't pass this, JSON data is pretty easy to figure out by jQuery, so it'll make intelligent guesses whether it's JSON or XML or HTML for you, but if you're having problems, you might want to set that dataType. Moving on down, here's the error function callback we can assign, and the list goes on and on and on with a lot of different things you can do such as the timeout, getter post, the url, and even username and password if you want to login to a site. So a lot of different things can be set, they provide a lot of great documentation here about how to use callback functions like this one. This would be really analogous to what I showed you with the get and the post where you simply pass in the URL followed by a callback function, but because we're using the ajax function, we have to do that through these JSON properties. So, I won't go through all the rest of them, there's a lot here if you ever want to do something called JSONP, you can even do that through the ajax function. In fact that's what you'd want to go through to work with that. Very nice, JSONP in a nutshell is a great way to make calls to cross-domain type of services. So definitely read through this if you're going to be sending data to and from the server and you'd like to use this more low level API, which is the ajax function in jQuery.

ajax() Function Demo

If you need complete control over how an ajax request is made from the client up to the server and how the response is handled, then you can use jQuery's ajax function. So in this demonstration, I'm going to show you how to use that function. Now of course if you're doing more simple things like you just need to get some JSON data, you need to get some HTML, you need to get a post, then I've already covered in the previous portions of this module some demonstrations and information about the load, the get, the post, getJSON, and those types of shortcut functions, and really they're just wrappers around the ajax function that I'm going to show here. But if you want complete control over how the request is made and want to be able to tweak things here or there, then if you go with the ajax function you can do that because it gives you really fine grain control over the different parameters. So, to start, you can see that I have, let me switch the design actually, a textbox for First Name and a textbox for Last Name, then a simple button. Now what we'd like to do is when the user fills in this information and hits submit, I'd like to make an ajax request up to the server, using the ajax function of course, and I want to pass this up as a JSON object. The server is then going to handle that and convert it in this case to a C# object, we'll simulate an insertion into the database, and then we're going to return a different type of object, which will be serialized to JSON for processing with jQuery. So that's all shown right here. You can see I have this operation called InsertCustomer, it accepts a customer object and has a cust parameter. Now this'll be important, you'll see this in just a second. You'll notice it returns an OperationStatus object. Now the OperationStatus object just holds some state about what's going on. So if you're in debug mode you could pass the StackTrace, otherwise you can pass things like the InsertedID, the Message, true or false, did it work. The customer object, very simple, it just has the customer ID, the FirstName, and the LastName. Okay, so that's kind of what's going to happen from the server side. Now, this could be Ruby on Rails, it could be PHP, some type of Python framework, it really doesn't matter. I'm using WCF in this example just to demonstrate the concept, but keep in mind, you can do this with anything that can handle the different formats, JSON, XML, even text. Alright, so once the customer is passed up, we'll kind of simulate and insert and we're going to return an OperationStatus object. We're going to simulate that it worked, have an ID, and then have a message about what was just done. So let's switch back over to here and you'll notice that the button is already wired up to the shortcut click function in jQuery. So what I'd like to do first is I need to go grab the values of the textboxes and convert those, I'm going to convert them into a JSON object. So we're going to call that customer. Now remember that we need to pass up this parameter called cust. And the way we're going to do that is we're actually going to say cust equals, that'll be the name, and then the value is actually going to be a JSON object, but I need to convert that JSON object into a string. So first let me just create the JSON object and then I'll show you a technique for converting it. So we're going to grab the FirstName from this particular guy right there, the textbox, and then we're going to grab the LastName and we'll grab it from this textbox. Okay, so that's our simple JSON object that we're ultimately going to pass up to the server. Now the problem is I need to get this in a string format because I need the name and the value, name as a string, value as a string. This is going to make a GET request, so it's going to put it on the querystring. So the way I can convert what really is a JavaScript object into a string is we can use a JSON stringify script. Now the JSON stringify script is available if you go to JSON.org, and Douglas Crockford wrote it who has a lot to do, pretty much all the main stuff with JSON, and we're going to use that script, which I've already referenced so you can see right here, to actually stringify this, or convert it to a string. So the way you do that is once you have the script reference, you can say JSON.stringify, and then in parentheses you can give it the value that you want to stringify there. So what that'll do is basically convert this JavaScript object that's up in memory at this point once it runs into a string, we'll then assign that, and kind of concatenate it together with our name. So we have our name and our value, we're ready to go. Okay, so that's what I'm going to send up to the server. Now, to send it up to the server we're going to use the ajax function. And so I'm going to actually pass into it the JSON object parameter. And so this allows us complete control over the different things we want to do. So we're going to go in and I'm going to say the url property is going to be, and I'm currently in the UsingAjax folder, but I want to call into this service. So we're going to say dot dot to go up one, or you could certainly go from the root of your site, and we'll say CustomerService.svc/InsertCustomer, and that's that operation that you just saw on the server side. The data we want to pass is our customer. Now the dataType that we're going to get back is going to be json data, that's what we want back. The final two things I'm going to add, I'm going to add a success function, and for that we're going to have the data, the string status code, and then the XMLHttpRequest object, should we need it. And then finally, I'm going to add an error property, and we'll add a function for that that looks like this, XMLHttpRequest, the status, and any error information. Okay, now in the error function I'm just going to come into here and we'll just say alert Error occurred, nothing real fancy on this one, and we'll pass it the status. So I'm not going to do much there, but what I'd like to do with the success is I want to update this div that's right below the button. So I'm going to grab that, we'll do an ID selector, get my pound on the front. Okay, now I'm going to update the HTML of it and we're going to kind of keep it simple to start. We're just going to say Insert status, and we want to know did it work or not. So I can pass in the data, and this is going to be the JSON data that's passed back. Now just as a real quick review, remember I'm passing back this OperationStatus object, so we should have access to these properties using JSON. So right now I just want to write out true to get it going. So we're going to say data, and because this is a default configure WCF Service, it'll wrap the JSON in a d property value, .Status. And we'll update the HTML of our div down here with that particular value. Alright, so let's go ahead and we'll call into that. We'll hit Submit and you can see that's working great. So let's also write out, we can of course get to the Message. And so let's go ahead and do that. So I'm going to wrap this down so we can see it a little bit better, and we'll say we want the Status plus, let's go ahead and do a line break, and then we'll do data.d.Message. So this will grab us the actual value that we're going to get back in the OperationStatus message and there we go, Customer Inserted, John Doe. If we go back over into the service, you'll see that's indeed what was passed back, Customer Inserted, and then it's going to be John Doe because that's what we passed up. So, this shows how you can use the ajax function to control really whatever you want to control, and there's a lot of additional properties that you can configure. This is a GET request. For instance, we might want to turn the caching on or off based on what we're doing. Now gets are going to be cached by default, but we may not want to do that. So there's a lot we can do there. Now if you go over to the jQuery documentation and click on Ajax on the left there and go into this ajax function, I showed a little bit of this earlier, but I use this overload, so I actually pass the URL by using the URL property, but we could have done it up here as well, you can see in jQuery 1.5, which I'm running right now, but if we come on down, for instance I could come in and say hey, I want to set the cache to true or false based on what we're doing. And so if set to false, it's going to force requested pages not to be cached by the browser, and that may or may not be good depending on what you're doing. So a lot of different things. The contentType of the data being sent up we can also control through the contentType property and there's a lot of stuff. jQuery 1.5 adds some converter and filter features that are nice, you can see that here, and the list goes on, so, we used a few of these though. Now, something I've used it for, and I'm not going to demo it here, but I at least want to bring it up is JSONP requests, cross-domain requests where you're not calling back to a page or service on your server domain, you're calling another one that you trust. Well, you can do that with the AJAX function as well. So, it's very, very flexible and there's some information about that on the jQuery documentation page. So that's an example of doing it. Just to show you what's going on behind the scenes, I have Fiddler fired up here, let me go ahead and remove all session. And I've shown this in a couple other demos. Let's go ahead and run it and let's see what's going on behind the scenes with our data. So, because it's localhost, I'm going to put a dot right there so Fiddler will recognize this, we'll submit it, hit Submit, get back our data. Let's go check out Fiddler now and it should be this bottom one, and yeah, there we go. So here's our get. You'll notice that the cust equals, and there's our kind of stringified version of our call that was made. And we can see the types being sent up, the XMLHttpRequest was used, and here's what was sent back. So again, because it was a WCF kind of default configured service, it wraps it in the d, the type of the object passed back is OperationStatus, and then we have all that other data that we just wrote out. So really, really nice for inspecting your data. So that's an example of using AJAX. It definitely is a little more involved, but at the same time if you need that control, it's really, really nice to work with and I think because it allows you to pass in a JSON object, I think it's pretty easy to work with. They have really good documentation on the page, on the jQuery.com site. So, that's how you can get started with it, you've now seen the whole gamut of them, we've seen get, we've seen post, getJSON, load, and then here's the most low level API that's available in jQuery for AJAX calls.

Summary

In this module, we've talked about a lot of the different features available in jQuery that make it easy to work with AJAX. Specifically, we focused on a few key of the key functions available such as load, get, getJSON, post, and AJAX, and I've showed you how those can be integrated into your jQuery applications. Load, as we talked about, allows you to load HTML content from a server, and very easily assign that HTML into a given selected DOM node. So you simply give it a selector, call .load, and then give it the URI, URL, whatever you want to call it, and it'll go out and grab that. We also talked about the get. Now get provides a little more flexibility because it doesn't do just HTML, it can also do JSON and HTML and XML. And then we talked about getJSON, which of course is a GET request, but specific to retrieving JavaScript object notation data. We also talked about a little bit of a difference between get and post, and I showed how post really is the same API as get, it's just that it executes a post request to the server. And then finally, I showed how ajax is the core, the low level API function that's available in jQuery that really all of the others tie into and use. So the sky is the limit when it comes to working with AJAX and jQuery because you can work with all kinds of different data, the cross-browser issues aren't issues because that's abstracted out from us, and it's just really easy to get going and actually I think kind of a pleasure to work with, it makes it fun to exchange data and to add those cool features into your web applications.