

Working with jQuery, Part 3: Rich Internet applications with jQuery and Ajax : JQuery: Building tomorrow's Web apps today

Effects and Ajax

Skill Level: Intermediate

[Michael Abernethy \(mike@abernethysoft.com\)](mailto:mike@abernethysoft.com)
Product Development Manager
Optimal Auctions

28 Oct 2008

JQuery is emerging as the JavaScript library of choice for developers looking to ease their creation of dynamic Rich Internet Applications. As browser-based applications continue to replace desktop applications, the use of these libraries will only continue to grow. Get to know jQuery in this series of articles that takes a look at JQuery and how you can implement it in your own Web application projects.

Introduction

JQuery has grown rapidly in popularity over the past few months to become the JavaScript library of choice for Web developers. This is coinciding with a rapid growth in the use and need for Rich Internet Applications (RIA), which are looking to replace desktop application with browser-based applications. Everything from spreadsheets to payroll apps to e-mail applications are replicating the desktop experience in the browser. As these applications become more numerous and more complex, a JavaScript library becomes ever more important as a solid foundation on which to build. JQuery appears to be that foundation of choice for developers. This series of articles looks to explore jQuery more in-depth and to provide a solid foundation from which you can build an RIA quickly and easily.

In the [previous article](#), you learned the three fundamental components to building an

RIA and how to add interaction to any page. The first module was the Event module, which lets you capture any interaction on the page from the user and respond to it programmatically. This lets you attach code to button presses, mouse movement, and so on. The next module you examined was the Attributes module, which explained how to get/set values on page elements, and how you can think of them as data objects with variables. These variables contain much of the information you will be using to decide which type of response to provide to users. Finally, you took a look at CSS manipulation, and how to change the layout, colors, fonts, and so on, of any element on the page without reloading the page. With these three modules, you learned the three basic elements of interactive Web pages — capture user interaction (Events), gather information (Attributes), and provide feedback based on the combination of events and information (CSS).

In this article, you will take the three basic elements of interactive Web pages a step further, providing the "cool" effects and features that are a part of the more advanced Web applications you see today. These additional modules aren't vital to providing an RIA, but they are the added features that users will remember, and ones that will greatly expand the scope and features available in your RIA. The first module you will look at is the Effects module, which includes features such as hiding elements, moving them around, fading them in and out, and so on. In other words, the "sparkle" that makes Web pages cool. The final module discussed is the Asynchronous JavaScript + XML (Ajax) module. This is what most people equate with RIA. Ajax gives Web applications the ability to communicate with the server, passing information to it and retrieving information from it, without reloading the page. (Contrary to some opinions on the Web, Ajax *is not* simply a cool JavaScript animation.) You'll find that jQuery provides incredibly easy-to-use Ajax tools, and in fact, makes using Ajax as simple as calling another JavaScript method.

The sample application in this article wraps up everything by showing how the Effects and Ajax modules can fit into the demo Web application, a Web mail application. I'll add some Effects to the demo to provide some pizzazz, and more importantly, add some Ajax code so that the mail application will display messages without having to reload the page.

Effects

The Effects module, given its name, may be assumed to contain only the animations and effects that "more serious" Web pages can avoid. But that's not entirely the case. Nearly every application will have a case when a certain page element needs to be hidden from view, or its view toggled depending on the state of another page element. These types of changes are vital to an RIA, because they let you get all of a page's elements in one page load, and then display only the information needed by hiding/showing certain elements. The alternative of reloading the page is not a good solution. Think of a combo box that has two choices, one that hides the div and that one shows the div. Obviously, it would be easier and more efficient to simply

hide/show the div with client-side code, rather than reloading the page to hide/show the div with every combo box change. Whether you want to simply hide/show it or make it fade in/out is up to you.

The most basic effects, as mentioned above, are the `show()` and `hide()` functions. These are rather straightforward; they obviously show and hide certain elements on a page.

Listing 1. Hide and show functions

```
// shows every <p> on the page
$("p").show();

// hides every <p> on the page
$("p").hide();

// hides every other <p> on the page
$("p:odd").hide();
```

In addition to these basic operations, both the `show()` and `hide()` functions offer alternatives that give you a little more control over how the show and hide work. The documentation describes it as a "graceful" show/hide, which translates, for a `show()`, to a combination of a fade in and a sliding out.

Before looking at some examples, let's take a step back and talk about the arguments you can pass into these Effects functions. Each function (with the exception of the generic `show()` and `hide()` functions) allows you to pass in a speed and a function to call when the effect is complete. The speed lets you control how fast or slow you want the effect to be. This argument can be either a String of "slow", "fast", or "normal". Additionally, if you need to control exactly how long an animation takes, you can also pass in the number of milliseconds as the argument. The second argument to the Effects functions is a function itself, which gets called when the effect is finished. This is important when tying together several effects as one larger effect, because it lets you reliably control when one effect is finished and when the next effect can start.

Listing 2. Complex effects

```
// the img with an ID of "picture" should start hidden
// when the "showPicture" button is pressed, show the img with an ID of "picture"
// and animate it, so it appears quickly in a fade In and slide out, and when
// it's done being shown, show the caption of the picture, which is
// always in the span right after the <img> tag

<input type="button" id="showPicture">

<span>This is the picture's caption</span>

// jQuery code inside the document.ready() function

$("#picture").hide().next().hide();
$("#showPicture").click(function(){
```

```
    $("#picture").show("fast", function(){
        $("#picture").next().show();
    });
});

// notice how it took more text to describe what you want to happen than it took
// to actually code it!
```

The Effects module has other functions that are very similar to `show()` and `hide()`, and ones that ultimately do the same thing; they just do it in a very specific manner. The `slideDown()` and `slideUp()` functions are both functions that show and hide a page element. However, they do so by animating the element by sliding it down or up (you can see the painstaking approach to the naming convention!). Like the enhanced `hide()` and `show()` functions I just talked about, you can control the speed of the slide and the function that gets called when they are complete. Furthermore, there is another option for showing/hiding page elements, and that's the `fadeIn()` and `fadeOut()` functions, which as their name suggests, will fade the page element until becomes transparent, and then disappears. These options also allow for custom speeds and functions called when complete.

There is one interesting function that doesn't totally hide or show a page element, and that's the `fadeTo()` function, which lets a page element be made partially transparent. I think this is a very important function in RIA, because transparency is a great way to place emphasis on certain elements on the page, and can be used to display disabled areas of the page. For example, you can have several tabs on a page, and have all the non-selected tabs show some transparency to reinforce the fact that they aren't selected. Or, on a page form, you can have all the elements without focus show some transparency to reinforce to the users which `Form` element has current focus. Plus, transparency is just cool. You can follow the general mantra of "If Apple does it, it MUST be cool" when designing anything.

Listing 3. Using `fadeTo()` to add coolness

```
// make all the Form elements on the page show transparency at 60%, except
// the one that currently has focus, which will not have any transparency.
// This is Apple cool!

$("#input").fadeTo("fast", .60);
$("#input").focus(function(){
    $(this).fadeTo("fast", 1);
});
$("#input").blur(function(){
    $(this).fadeTo("fast", .60);
});
```

The final function I want to address in the Effects module is both the coolest and the most error-prone function. It's a custom animation method that lets you define all the parameters in the animation, and jQuery will take care of the rest. You provide an Array of parameters with their final values, and jQuery determines their current values, and, using the defined speed of the animation that you pass in, will smoothly

animate the page element until it reaches the final values you provide. Listing 4 shows an example of the custom animation method. Obviously, there are an infinite number of possibilities in this function, so I recommend just playing with it if you feel you need to do your own custom animation.

Listing 4. Custom animation method

```
// when the button is clicked, make the div with an ID of "movingDiv"
// have the custom animation provided.
$("#myButton").click(function(){
    $("#movingDiv").animate({
        // will increase the width, opacity, and fontSize of "movingDiv"
        // and do it in 5 seconds
        width: 700;
        opacity: 0.5;
        fontSize: "18px";
    }, 5000);
});
```

Ajax

Currently, the big buzz about every Web page is "Does it use Ajax?," but does everyone understand what Ajax truly is? A Google search on "Ajax" turns up millions of results (as do most searches), but it seems to indicate that there's some confusion as to what exactly the term Ajax encompasses. Ajax is *not* a really cool animation on some page, or a pop-up window with a really cool shadow underneath it. Just because Ajax is cool, it doesn't mean that every exciting thing on a Web page is Ajax. Ajax, at its core, is simply a way for a client-side Web page to pass information back and forth with the server without having to reload the page. So while Ajax can't give you cool effects on a page, it does provide the means for a Web application to mimic a desktop application. So, the buzz surrounding Ajax is warranted - its widespread use is directly responsible for the rapid growth in RIAs you are seeing today.

jQuery makes working with Ajax incredibly easy! I'm not exaggerating with that statement either. For anyone who's ever had to work with Ajax without a JavaScript library, had to work with `XMLHttpRequests`, had to work with the differences between the Microsoft® and the Firefox version of `XMLHttpRequest`, had to parse through the return codes, and so on, and so on, I can tell you that jQuery has made Ajax as simple as a single function call. Seriously! What took 100 lines of code to do before can now be done in 3 or 4 lines of code. What an incredible time-saver. Personally, I know that before jQuery, the idea of adding Ajax functions everywhere seemed like a lot of work. Now it's extremely simple, and has led to my applications taking full advantage of everything that Ajax can offer. If it's as simple as a normal function call, why wouldn't you want to use it?

Take a look at the functions you will most likely use in your own Ajax needs: the

`post()` and `get()` methods. These functions behave much like you would expect from any jQuery function at this point in your reading, by allowing you to specify the URL to call and the parameters to pass in, and then allowing you to specify the function when the Ajax method returns. In this sense, because of the way these two methods are set up, they truly make calling an Ajax method in jQuery exactly like calling any other method in jQuery. Take a look in Listing 5.

Listing 5. Post and get Ajax methods

```
// this code would be in a php file named myTest.php
// why did I switch to PHP for the Ajax examples? Java/JSP gets tough because
// you need to show the code in the Servlet, which isn't necessary with PHP. The
// functions work equally well in both PHP and Java/JSP though.
<?php
    echo "This is my return String";
?>

// here's how simple the Ajax calls are in jQuery
// This is a post function
$.post("myTest.php", {}, function(data){
    $("#p").text(data);
});

$.get("myTest.php", {}, function(data){
    $("#p").text(data);
});
```

As you can see from these two examples, the functions are nearly identical to other jQuery functions, and are far easier to use than without a JavaScript library. There are several arguments you can use to expand the functions of the Ajax calls. The first argument is obviously the URL to call. This can be the PHP file, a JSP file, a Servlet — basically anything that will process the request. It doesn't even need to respond to the request (as you will see in the example application later). The second argument, which is optional, is to pass in data with the post/get. This is in the form of an array. Usually, you will pass in information contained in Form elements, userIDs from the page, and so on. Anything that the server-based file will need to process the request. The third argument, also optional, is a function to execute if the Ajax function returns successfully. This will usually contain code to process the results of any information passed back from the server. Listing 6 shows some examples of these first three arguments, before I talk about the fourth argument.

Listing 6. Post with optional arguments

```
// place a username and password input field on the page
<input type=text id="username">
<input type=password id="pass">

// call a server-based PHP file that will process the information passed to it
$.post("myFormProcessor.php", {username: $("#username").val(),
    password: $("#pass").val()});

// conversely, this PHP file could also return information to the function, from which
// you could process the results
```

```

$.post("myFormProcessor.php", {username: $("#username").val(),
                               password: $("#pass").val()} ,
      function(data){
        // the data variable contains the text returned from the server, you don't
        // have to set anything up ahead of time, jQuery will do it all for you
        if (data != "ERROR")
          $("#responseDiv").text(data);
      }
    );

```

You can quickly see that working with Ajax in jQuery is very straightforward and easy. The functions can get more complex, however, when you start working with more complex information from the server than simple text Strings. In more complex Web pages with more involved Ajax calls, the return data could be in the form of XML. The return data could also be in the form of JSON objects (JSON is basically a protocol for defining objects in JavaScript code). jQuery allows an optional fourth argument to these get/post methods that lets you define ahead of the time the type of return value you are expecting from the server. You can pass the Strings of "xml" for XML strings, "html" for HTML strings (or plain text), "script" for JavaScript code, or "json" for JSON objects. So, for example, by specifying the return object as a type "json", jQuery will automatically convert the response String from a server into a JSON object, allowing you to reference it immediately.

Listing 7. Specifying the return type in Ajax

```

// specify the return object to be a type JSON object, and process the
// return object as a JSON object, referencing fields in it without
// casting it to any object or evaluating it

$.post("myFormProcessor.php", {username: $("#username").val(),
                               password: $("#pass").val()} ,
      function(data){
        // jQuery has already converted the data response into a JSON object, so
        // you can immediately reference its fields, providing cleaner-looking code
        // and allowing future changes, and in my opinion, making it easier to work
        // with than XML responses
        $("#username").text(data.username);
        $("#address").text(data.address);
        $("#phone").text(data.phone);
      },
      "json" // here you specify that the return type is JSON
    );

```

The other Ajax function I want to spend some time on is the `load()` function, which allows users to load a specified page and get HTML back as a result. Based on that description, it doesn't sound too exciting. However, you also have the ability to parse the return information using jQuery code just as you parse your own Web pages on startup. What does this mean? Well, based on the ability to load any Web page, and then parse it with jQuery, you have at your hands a very effective and easy-to-program page scraper, from which you can gather any type of information from any page. Let's take a look.

Listing 8. Example app of load() function

```
// create a very primitive stock price quote by calling Yahoo's stock quote, and then
// scraping the information from their pages.
// in this case, look up IBM's stock price, and place it in the text field with an ID of
// "stockPrice"
// the span with the ID of "yfs_l90_ibm" contains the stock price
$("#stockPrice").load("http://finance.yahoo.com/q?s=ibm #yfs_l90_ibm").text();
```

The last two functions from the Ajax module I'd like to point out are really utility functions that greatly assist in working with Ajax. Like I've pointed out many times already, many of the interactions between client and server revolve around a Form, and the elements it contains. Because this type of communication is so common, there are two utility functions in jQuery to assist in the construction of parameters getting passed to the server, either in the form of an HTTP query string or a JSON string. You can use both utility functions to assist you in your Ajax needs. They are handy because they both can encapsulate an entire form, no matter how many elements get added/removed/changed during development. Listing 9 shows an example of this.

Listing 9. `serialize()` and `serializeArray()` functions

```
// the serialize function will look at every Form element inside the specified element
// and automatically construct an HTTP String that contains all the information
// of the elements, in the form of <element name>=<element value>&
// for example, "firstname=Michael&lastname=Abernethy"
// this can then be attached to the URL to pass the information via an Ajax call
$.post("myFormProcessor.php?" + $("#myForm").serialize());

// further, a similar thing could be done with the serializeArray function
// which will convert a Form into its JSON equivalent
$.post("myFormProcessor.php", {json: $("#myForm").serializeArray()});
```

Bringing the lessons all together

To bring together all the lessons, you'll take a final look at the demo Web application, the Web mail application you've become familiar with over the past few lessons. I'll add multiple Ajax calls from the client-side to the server-side in order to gather information. I'll utilize Ajax to get message information when you read a message, and also utilize Ajax methods to take care of deleting messages. Then, I'll combine this with some Effects, so that when a user deletes a message, it will immediately remove those messages from the screen, even though the user is not reloading the page, and the actual deletion is occurring asynchronously using the Ajax call. At the conclusion of today's lesson, you should be able to see how easy it is to use Ajax calls in your own Web applications, how you can utilize them to really mimic a desktop application, and finally, how Effects can be used to enhance the usability of the application.

Listing 10. Sample Web app - deleting messages

```

// First, let's look at how you handle deleting a message.

// The first step is to create a button that will actually delete messages
<input type=button id="delete" value="Delete">

// next, you'll add a checkbox in each row of the table, so that users can select
// which messages they want to delete. You'll use these checkboxes later, and
// the information contained in them is equally important (that's called
// foreshadowing!)
// Notice how the value of each checkbox is the message.id!

<tr class="messageRow" id="<%=message.id %>">
<input type=checkbox name="delId" value="<%=message.id%>" class=selectable>

// Now that the HTML is complete, look at the jQuery code to execute these deletes

// First, attach an event to the delete button, so when it's pressed, it will
// start deleting the checked messages

$("#delete").click(function() {
    deleteMessages();
});

// Finally, let's define the deleteMessage() function, because that contains the meat of
// today's lessons.
// Because this is the culmination of every lesson, let's look at everything I did to
// get this working!
// Note 1 - I loop through each of the checkboxes that are checked by passing in a very
// specific search parameter, to find only the members of the "selectable" class that
// are checked.
// Note 2 - because the value of the checkbox is the same as the ID of the table row in
// which it is contained, you can use the Effects module to hide the entire table row,
// by passing in the value of the checkbox, and getting the table row back, and then
// hiding it.
// Note 3 - I make an Ajax call to actually delete the message from the DB. I have
// to pass the messageID to the server so that it knows which one to delete. That
// information is contained in the checkboxes value, which I pass with the Ajax call.
// Because I don't really care if it's successful or not, I ignore any reply from
// the server.

function deleteMessages()
{
    $(".selectable:checked").each(function() {
        $("#"+$(this).val()).remove();
        $.post("<%=HtmlServlet.DELETE_MESSAGES%>.do", {delId: $(this).val()});
    });
}

```

For the second example, take a look at how I read messages, which shows how Ajax can also be used in jQuery:

Listing 11. Sample Web app - reading messages

```

// You've seen most of this code previously in the example from last article, so let's
// focus on the Ajax portion of the code.
// Note 1 - I make an Ajax call with all 4 arguments defined. I have to pass two
// variables to the server in order to read the message. The first is the message
// ID number, because I need to know which message I want to read. The second is
// the current view...for reasons I can't recall (not important really).
// Note 2 - The fourth argument into the Ajax function is "json", indicating that
// I expect a JSON object back from the Ajax call. jQuery will automatically

```

```
// convert the response String into a JSON object.
// Note 3 - Notice that I handle the JSON object directly, without using an
// eval() function, because jQuery has already created the object. I can
// reference its fields directly.
$(".messageRow").dblclick(function() {
    if ($(this).hasClass("mail_unread"))
    {
        $(this).removeClass("mail_unread");
    }
    $.post("<%=HtmlServlet.READ_MESSAGE%>.do", {messageId: $(this).attr("id"),
                                                view: "<%=view %>"},

        function(data){
            if (data != "ERROR")
            {
                // using JSON objects
                $("#subject").val(data.subject);
                $("#message").val(data.message);
                $("#from").val(data.from);
            }
        }, "json");
    $.blockUI(readMess, {width:'540px', height:'300px'});
});
```

Conclusion

The importance of JavaScript libraries like jQuery will continue to grow as applications are ported from the desktop to the browser. These applications will continue to become more and more complex, making a solid cross-browser foundation like jQuery a necessity in any Web application project. JQuery has begun to distance itself from other JavaScript libraries and is becoming the library of choice for many developers due to its ease of use, and ability to do everything they need it to do.

In this third article in the series, you learned about two modules that can really add richness to your application, and make it blur the lines between a desktop application and a Web application. The most powerful addition with this lesson was the Ajax module, which lets you greatly simplify your use of Ajax, by making it as seemingly simple and straightforward as making any other jQuery method call. You saw the power of Ajax as well, through several of the examples, and saw that it is a great tool to use to increase the responsiveness of applications without suffering the annoyances of page reloads or delays. You also learned about the Effects package, and saw that, used properly, animations and hiding/showing page elements are strong reinforcements of proper UI design. Used effectively in combination, Ajax and Effects can add to a Web site's dynamism tremendously.

Finally, you looked again at a sample Web application, and saw that with the addition of Ajax modules, you could read and delete messages without reloading the page. Then, you saw that you could delete messages in the Web application, and, with a combination of Ajax and Effects, could remove the message from the user's page and the DB, all without reloading the page. To the user, the deletion of a message is a transparent process, and programmatically you could make it behave that way as well.

This also wraps up the JQuery articles about the core libraries that ship with every JQuery release. These three articles have exposed you to every module contained with JQuery, and ideally has shown you that working with JQuery is easy and straightforward. Additionally, you should see that no matter what Web application you are working on, JQuery should be the place to turn for any type of RIA, as it provides a solid foundation for all your JavaScript needs. The articles in this series took you through JQuery to the point that you should feel extremely comfortable using it in your own code. The [first article](#) showed you how JQuery works functionally, and how you can use it to search for page elements, loop through them, and access them like any Array object. The [second article](#) introduced you to the three modules that are fundamental to providing richness. Finally, this article provided the final pieces to the framework to produce a complete, complex web application.

Downloads

Description	Name	Size	Download method
Zip file containing the sample application	jquery.zip	69KB	HTTP

[Information about download methods](#)

Resources

- Download the [1.2.6 Minimized jQuery](#), which was the latest stable version at the time of this writing, and drop it in your own code.
- Read the complete [jQuery API page](#) to see all of the available functions in the library.
- Worried about jQuery's cross-browser compatability? Check the list here for [cross-browser support](#).
- Check out [really cool advanced effects](#) that you can create with jQuery.
- Get a thorough and complete background on CSS, JavaScript, and any other Web language at [W3Schools](#).
- Get a general overview of jQuery in this introduction article "[Simplify Ajax development with jQuery](#)", (developerWorks, April 2007).
- "[Ajax overhaul, Part 2: Retrofit existing sites with jQuery, Ajax, tooltips, and lightboxes](#)" (developerWorks, May 2008) is a look at some of the plug-ins available for jQuery, and how they expand the usefulness of the library.

About the author

Michael Abernethy

In his 10 years in the technology field, Michael Abernethy has worked with a wide variety of technologies and clients. He currently works as the product development manager for Optimal Auctions, an auction software company. His focus is on Rich Internet Applications and making them both more complex and simpler at the same time. When he's not working at his computer, he can be found on the beach in Mexico with a good book.