

Working with jQuery, Part 2: jQuery: Building tomorrow's Web applications today

Events, attributes, and CSS

Skill Level: Intermediate

[Michael Abernethy](#)

Product Development Manager
Optimal Auctions

23 Sep 2008

This second article in the [jQuery series](#) looks at how to add more interaction to any Web site to create a dynamic Rich Internet Application. Learn how jQuery utilizes a combination of events produced by user interaction, information gathered from the Web site itself, and the ability to change the look and feel of the application without reloading to create these RIAs quickly and easily.

Introduction

jQuery has grown rapidly in popularity over the past six 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 (RIAs), which look to replace desktop applications with browser-based applications. Everything from spreadsheets to payroll and e-mail applications are replicating the desktop experience in the browser. As these applications become more numerous and more complex, a JavaScript library will become 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 explores jQuery in depth and provides a solid foundation from which any developer can build an RIA quickly and easily.

In [the last article](#), you learned the basics of jQuery, including how to set up jQuery in your development environment and how its core functions work. You saw how jQuery makes searching and finding elements on the page easy with the Selection

and Filtering methods, and how you can find exactly the single element or groups of elements you are looking for. Next, you went through the various methods that jQuery provides for you to traverse through the selection results, with many of the functions following those you are likely familiar with in other programming languages. Finally, you went through a sample widget, the Select All/Deselect All check box, and saw how jQuery can make this widget in only a few lines of code.

In this article, expand your jQuery knowledge by looking at flashier functionality, and add some actual "richness" to your plain old Internet application so you can label the demonstration application an RIA. First, I'll start by showing you how jQuery handles events. Events are defined as mouse clicks, highlights, dragging, and so on. Note that these are not limited to just buttons and such, but can be made to handle a mouse click on any div, span, and so on. Next, I will move on to discuss how to get and set attributes on the objects in the Web page. This includes how to get text from form elements, innerHTML from divs, and even how to figure out which classes are attached to which elements. The final section will discuss how to modify the CSS characteristics of the page elements, without reloading the page or adjusting an external stylesheet.

The sample application will wrap up everything into additional widgets for the Web mail application, showing how you can create a client-side, rich application, changing colors, sizes, and locations of objects based on interactions with the page. (These interactions are limited to client-side only today—the next article will add server-side interaction as well.) By the end of this article you should have all the jQuery tools in place to create your own RIA and impress your clients.

Events

The Events module in jQuery is really the first step in building interaction to a Web application, since events are commonly the trigger for things to happen on the page. As I pointed out in the introduction, you shouldn't think of events as only occurring with Form elements—in fact, any element can trigger an event, and you can take advantage of this to build custom widgets more easily and add some interaction that may be unique and not tied to a specific Form element itself.

That being said, most of the events *are* based around the Form elements, and it is easiest to show demonstrations of each of these methods by using them. Before diving into the functions available, note that the Events module follows a certain pattern for each function. Each event function contains two forms: one without any arguments, and one that contains a function as an argument. The difference between the two is important, and it is consistent from function to function. The function with no parameters will actually trigger that event. In other words, a call to `click()` will actually cause the button to be clicked. The `click(function)` will be called when that button is actually clicked, or when its `click()` function is called. Confusing? It probably is based just off the description, but seeing an example

should clear it up.

Listing 1. jQuery Event methods

```
// make the "myButton" click. This will cause the button to click and any actions
// tied to it will occur - for example, it could submit a form, or other
// jQuery actions could be tied to it.
$("#myButton").click();

// use jQuery to setup what will actually happen when the "myButton" is
// actually clicked.
$("#myButton").click(function(){
    $("#myDiv").toggle();
});

// A common pattern in jQuery when setting up actions on a page is to trigger the
// action to occur initially when the page is loaded. This occurs frequently
// with AJAX setups, where the values come from the server.
// In this example, the myDiv has its visibility toggled every button click. When
// the page is loaded, we call click() immediately, which toggles the view
// as soon as the page views (not a practical example, but you should see the design)
$("#myButton").click(function(){
    $("#myDiv").toggle();
}).click();
```

The events I list below *all* follow the design I describe above and have two functions associated with them. I list only the first here for succinctness:

- `blur()` - Called whenever a Form element loses focus, like when a text field that has focus is tabbed out of
- `change()` - Called when a Form element loses focus and its value has changed since it has gained focus. Internet Explorer and Firefox handle this slightly differently.
- `click()` - Called when a page element (doesn't necessarily have to be a Form element) is clicked on
- `dblclick()` - Called when a page element (doesn't have to be a Form element) is clicked on
- `error()` - Called when there's an internal error on the element, which differs from browser to browser, and probably hasn't been seen by many people
- `focus()` - Called when a Form element gains focus
- `keydown()` - Called whenever a page element has a keypress done on it/in it
- `keyup()` - Called whenever a page element has a keypress released on it/in it
- `keypress()` - Called when a keydown and keypress are done over the

same element in quick succession

- `select()` - Called when text is selected in a text field, and not when things are selected in a combo box (that's a change event)
- `submit()` - Called when a Form is submitted

There are additional functions that do not follow the pattern I've outlined above and contain only a single function which you can call. I'll list these exceptions here, since they are not as commonly used:

- `resize(fn)` - called when an object is resized
- `scroll(fn)` - called when an iframe is scrolled
- `load(fn)/unload(fn)` - happens when objects are loaded/unloaded on the page

Finally, some events are tied to the mouse, as you could probably expect. I've included them in their own third section because they are commonly misused, and jQuery, having realized this, has replaced some of them with unique functions. These are here only to map directly to the underlying DOM events, but for all practical purposes, alternative methods exist that you will likely use instead of these. The `mousedown(fn)` and `mouseup(fn)` methods are called when a mouse is pressed or released on an element. However, often times the `click()` method should be called instead, since it will also be thrown as an event, and it is more in line with expected behavior and is less error prone. Think about what happens when users press their mouse on a button, realize it's a mistake and move their mouse off of the button before releasing it. If they release it over another page element with the `mouseup(fn)` defined, what behavior should take place? Ideally, the use of these two functions should be limited to dragging and dropping interfaces, where the click would not make a suitable replacement.

The final two methods of the Event module, `mouseover(fn)` and `mouseout(fn)`, are quite common in most Web sites today. They are used to display hover help, shadow boxes for displaying pictures, and color changes based on the location of the mouse pointer. JQuery realized that these two functions were going to be used quite often and also realized that most people would not use them correctly, leading to countless errors. It's not that people would purposefully introduce bugs into their code, but they would not be able to code for the nested components and other intricacies of the pages. So, jQuery has added a method to the Event module to replace these two functions, called the `hover(fn1, fn2)` function.

Listing 2. jQuery Hover method

```
// This demonstrates the hover() function as implemented by jQuery. You define two
// functions: what happens when the mouse moves over the specified element
```

```
// and what happens when the mouse moves off the element.  
// In this example, each row in a table will get a red background when  
// the mouse moves over it and a white background when the mouse leaves.  
$("tr").hover(function(){  
    $(this).css("background", "#0000ff");  
},  
function(){  
    $(this).css("background", "#ffffff");  
});
```

Attributes

Part of what makes a page interactive is its ability to get certain information from one area of the page and transfer it to another location. This example can be as specific as getting information from a text field and placing it into a table, or it can be as broad as taking information from a combo box, transferring it to the server, and putting the response from the server into a different combo box. At its core, interactivity is the result of the transfer of information on a page.

There are many different ways to store information on a page, and especially different ways to store information in individual elements on the page. You can probably imagine that a simple `<p>` will not contain as much information as a text field (which may or may not be true), and as a result there are different functions for accessing this information. Similarly, you can already probably come to the conclusion that for every way to get information from a page element, you can also set the information on those elements. In effect, each page element is a data object, which contains variables encapsulated by getter/setter methods. The differences between this JavaBean-model and what actually occurs in jQuery are the method names and the limitation that some elements will not be suitable for certain functions.

Before diving into these methods, let's look at what information *could* be stored in page elements. Something as simple as a `<p>` may only contain a CLASS or ID as information. Something like an `` may contain far more information, things like the "src", the "alt", the "width", and the "height". Finally, something as complex as an `<input type="password">` may contain things like "defaultValue", "maxLength", "readOnly", or "accessKey".

This diversity of potential variables has caused jQuery to create a generalized function to access them. This function is the `attr(name)` and is the generic way to access information from any page element. Take a look at some examples to see how this works.

Listing 3. jQuery attr() function

```

```

```
// Calls to the attr() function will return the following
$("#spacer").attr("src"); // will return "/images/space.gif"
$("#spacer").attr("alt"); // will return "blank"

// Similarly, you can access the ID in the same way
$(img).each(function(){
    $(this).attr("id"); // will return "spacer"
});
```

This function becomes very useful when you are trying to add interactivity on a page and do it in a clean way. In fact, before the `data()` functions were added (see below), you usually had to squeeze the information you needed into one of the available variables. For example, if you had a page with two frames, and the first frame showed tabs, and the bottom frame showed the contents of each tab, you could set up something like this:

Listing 4. `attr()` in action

```
<!-- This would appear in the top frame as a tab. The CSS file would control how
the tab appears, and the only HTML code needed would be this -->
<td>
    <div class="tab" id="/messages.jsp">Messages</div>
</td>

// This code would appear in the jQuery section. Notice how we get the ID from the tab,
// and use that information to set the bottom frame, named 'content' with the content
// on the page "messages.jsp"

$(".tab").click( function() {
    window.parent.frames['content'].location = $(this).attr("id");
});
```

In addition to *getting* the attribute values on each element, you can also *set* the values. This has the impact of changing the element's look or behavior programmatically.

Listing 5. Changing attributes with `attr(str)`

```
// will change the image source, and the image displayed on the page will change
$("#img").attr("src", "myimage.jpg");

// will change all the links on the page to go to one specific page
$("a").attr("href", "mypage.html");

// will change the maxLength on all password fields to 10 characters
$("password").attr("maxLength", "10");
```

The Form elements on a page have a special function that can be called on them in order to get the value attached to them. This is particularly handy when working with forms and validations, and you will likely use these functions quite often when creating interactive Web sites with Form elements.

Listing 6. Form element `val()` functions

```
// will get the text contained in the text field and check that it's not blank
$("#textfield").each(function(){
    // use the val() function to get the text inside the textfield
    if ($(this).val() == "")
        $(this).next().text("Error");
});

// on a new password page, this will compare the new one with the confirmation,
// to make sure they are equal
if ($("#newPassword").val() != $("#confirmPass").val())
    $("#newPassword").next().text("Error");
```

There are also functions which allow you to get information contained within certain tags. Why might this be useful? Well, you can get all the information within a certain `<td>` tag, and replace it, or you can turn all the text in a `<p>` to lowercase. There are two separate ways of getting these attributes, and you can't use the `attr()` function to do so. Like all the other attribute functions, these have corresponding setter methods as well. The first is the `html()` function, which will return all the innerHTML of a certain tag. The other is the `text()` tag, which will return all of the text inside of a certain tag. What's the difference? The `html()` function will return text that includes HTML tags, while the `text()` tag will strip them, and return only the interior text. Look at the following examples to see how they differ.

Listing 7. html() vs. text()

```
// this will examine every <td> tag, and if the value is blank, it will insert
// a "-" into it, as a placeholder.
$("#td").each(function(){
    // check the text of the table cell
    if ($(this).text() == "")
        $(this).text("-");
});

// this will convert every paragraph's text to lowercase
$("#p").each(function(){
    var oldText = $(this).text();
    var newText = oldText.toLowerCase();
    $(this).text(newText);
});

<-- This shows the difference between text() and html() -->
<div id="sample"><b>This is the example</b></div>

$("#sample").html(); // will return "<b>This is the example</b>"
$("#sample").text(); // will return "This is an example"
```

Finally, a recent addition to the jQuery library for attributes is the `data()` functionality. This grew out of the jQuery UI project and has been recently incorporated into the jQuery project as a whole. In essence, the UI project developers felt they didn't want to "hack" the available attributes for certain page elements and wanted a way to create their own attributes, as many as they needed, in which they could store information. Think back to the example of the tabs from above. I "hacked" the link inside the ID of the DIV, which probably isn't the ideal

method. However, given the limitations in jQuery's previous releases, this was the only option. With the inclusion of the `data()` functions, you can provide a more elegant solution to this problem. Think of the `data()` functions as a way to access an internal Map that is contained on each page element. A Map is simply a collection of key-value pairings. This allows developers to create any custom attribute they want for a page element and attach any value to that attribute. Ultimately, this could lead to easier code, and as projects grow larger and larger, easier maintenance as well. Let's rewrite the example from above using the new `data()` functionality:

Listing 8. The new `data()` function

```
// create the div like we did above, but without any specific information. In this
// way we can create a generic HTML layout and customize it in our jQuery code.

<td>
  <div class="tab"></div>
</td>

// Now customize each tab in the jQuery code.

$(".tab").eq(0).text("Messages");
$(".tab").eq(0).data("link", "messages.jsp");
$(".tab").click(function(){
  window.parent.frames['content'].location = $(this).data("link");
});

// Taking this a step further, you can picture all this information coming from
// an external properties file via a Java array. This would be the code on a JSP
// page.

<%
  // array containing tab names
  String[] tabNames;
  // array containing tab links
  String[] links;

  for (int i=0; i<tabNames.length; i++) {
%>
  $(".tab").eq(<%=i%>).text("<%=tabNames[i]%>");
  $(".tab").eq(<%=i%>).data("link", "<%=links[i] %>");
<% } %>

$(".tab").click(function(){
  window.parent.frames['content'].location = $(this).data("link");
});
```

CSS manipulation

The final part of this lesson is manipulating the CSS on a page dynamically without adjusting the stylesheet or reloading the page. This will give us the ability to add some basic effects to the page by changing simple things like color, font, and so on. The CSS portion of jQuery is actually the original inspiration for the entire library. The goal was to make programming CSS easier on pages and, as you can see, the project has grown substantially since then. The original intention of the project still exists, and jQuery provides methods to make programming with CSS much easier. I

will show you, however, that the traditional functions provided by jQuery for CSS manipulation are actually not suitable for today's Web environment, and I will show you other functions (also in jQuery) that you should use instead.

The basic two functions for manipulating CSS on a page allow you to pass in a single attribute as a String and then a value as a String, or you can pass in an Array of String/String values all at once. Both functions pretty much do the same thing and will change the CSS on the page easily.

Listing 9. `css()` functions

```
// change the background of every div to red
$("div").css("backgroundColor", "#ff0000");
// - or -
$("div").css("backgroundColor", "red");
// - or -
$("div").css({backgroundColor: "#ff0000"}); // notice the braces and lack of quotes
```

You can see that these functions are fairly straightforward, and you can pick them up in no time. However, you should also see the problem with them given the current trends in Web page design. Normal Web pages have their style removed from the page and placed in an external file or section of the code, in a stylesheet. You would definitely *not* want to place style code into the JavaScript code if you can help it. That would make future changes to the look of the site difficult.

Luckily, there are replacement functions that do a fine job of giving you the separation you need, while still making the CSS manipulation easy and straightforward. These functions allow you to add and remove classes from page elements. By placing the style for these classes in external style sheets, you can keep the separation of style, data, and events that is crucial in complex pages. Take a look at some examples:

Listing 10. Preferable CSS manipulation - `addClass()` and `removeClass()`

```
// will add the "input_error" class to any form elements that fail to validate
// you can picture the "input_error" class in the external CSS file defining
// a red border and red text

$(" :textfield").each(function(){
    if ($(this).val() == "")
    {
        $(this).next().text("Error");
        // this will turn the text field's border/text red
        $(this).addClass("input_error");
    }
    // this tests if the text field has the class attached already
    else if ($(this).hasClass("input_error"))
    {
        $(this).next().text("");
        // this will remove the class, restoring a normal border/text
        $(this).removeClass("input_error");
    }
});
```

As you can see from that example, adjusting the CSS by referencing classes defined in external stylesheets is the preferable way to manipulate the CSS on a page. It allows the Web site creator to change the way an error message is handled *across the entire site* by changing one stylesheet, rather than trying to hunt down every instance of the code, as would be required by calling preceding `css()` methods. Though those methods are easy to use, and straightforward, they aren't really the solution for a large Web application and should be avoided in favor of the `addClass()` and `removeClass()` methods.

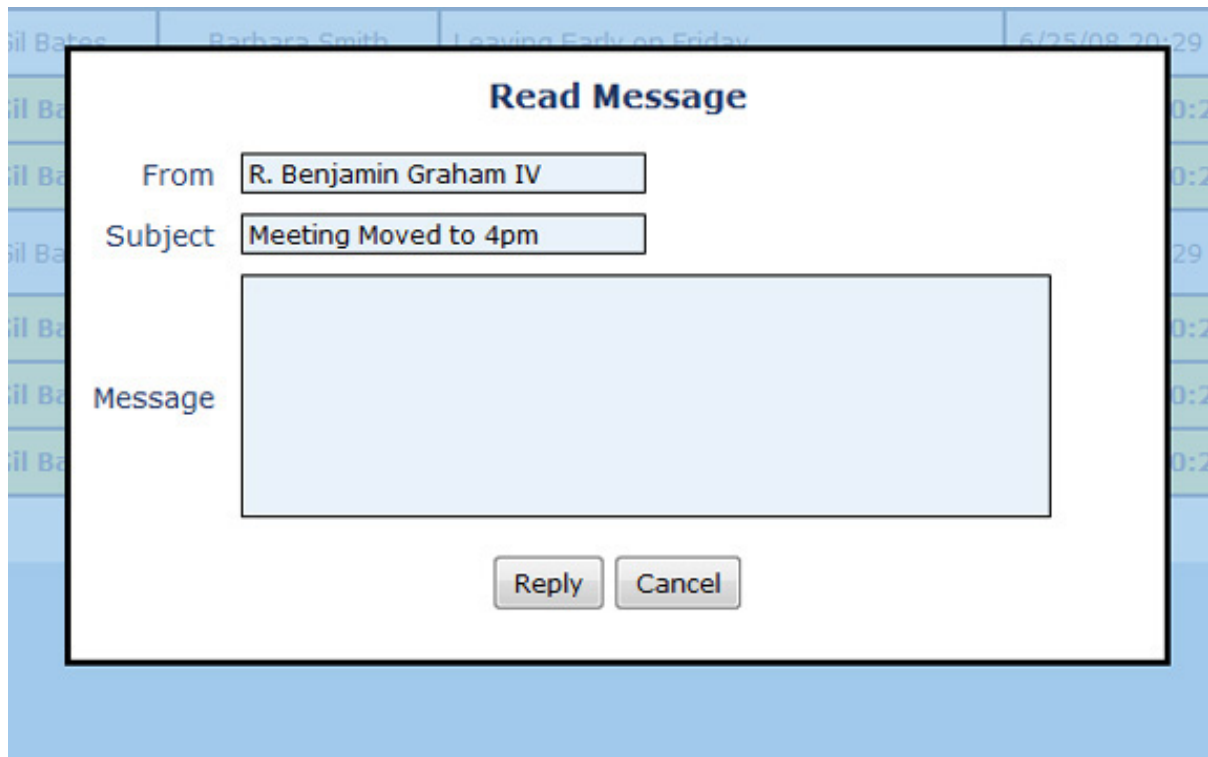
Bringing it all together

To bring it all together, let's look again at the sample. It is a Web application with interactivity attempting to create an RIA and give users the impression of working with a desktop application for their e-mail. In this example, you'll utilize the Event, Attribute, and CSS modules to define how the Web mail application deals with single mouse-clicks and double mouse-clicks. You can see the expected behavior from the screen shots below. When a user single-clicks on a table row, the row will change colors to highlight his or her current selection. When a user double-clicks on a message, it will let the user read the message, but also change the row's background color if the user is reading a new message, indicating that the message is no longer unread.

Figure 1. Single-click on a row

<input type="checkbox"/>	To	From	
<input type="checkbox"/>	Gil Bates	J Query	Nice Example
<input type="checkbox"/>	Gil Bates	Needie Joe	Give me Mon
<input type="checkbox"/>	Gil Bates	LotteryWinner	You've Won t
<input type="checkbox"/>	Gil Bates	Barbara Smith	Leaving Early
<input type="checkbox"/>	Gil Bates	Amy Jones	Status Repor

Figure 2. Double-click on a row



Listing 11. Bringing it together

```
// First we add the rows to the table. Each row is a member of the "messageRow" class.
// We also give an ID to each row, and this ID is the message number itself, which is
// gotten from the Java data object. Keep in mind this sits in a for loop in a JSP file.

<%
  for (int i=0; i<messages.size(); i++)
  {
    MessageData message = messages.get(i);
%>
    <tr class="messageRow" id="<%=message.id %>">

// Now that the table has been laid out, we can define our jQuery code to capture single
// mouse clicks and double mouse clicks.

// Notice how we capture a single click on the table row with the click() function. Next
// notice how we use addClass() and removeClass() instead of manipulating the CSS
// directly with a css() function. This lets us change the stylesheet underneath
// the code without modifying our jQuery code.
$(".messageRow").click(function() {
    $(".messageRow").removeClass("message_selected");
    $(this).addClass("message_selected");
});

// Now we capture the double click on a table row. Ignore the AJAX methods with the
// post() function, which we'll get to in the next article.
// We use the dblclick() function here to capture double clicks.
// Notice in the AJAX call, how we get the ID out of the table row that was double
// clicked.
// We pass this ID to the server in order to get the information about the message back
// from the server. We defined the message number in the JSP code, so that the ID
// contained the message number.
$(".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){
        // Do AJAX stuff here
    });
});
```

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 second article in the series, you expanded your jQuery knowledge by looking at interaction on a Web page and saw how to achieve basic interaction on the client (without getting information from the server). You started by looking at the Event module, which defines how page elements react to a multitude of interactions, including mouse interaction, keyboard interaction, and focus interaction. You saw that events are the big driver for interactivity on Web pages, and that they don't necessarily have to be attached to Form elements in order to be used. I then moved on to discuss Attributes, and how to properly get attributes from and set them on page elements. You saw that the generic `attr()` function can be used generically on every element, and that the Form elements have special functions to get their values. You also saw a new addition to jQuery, the `data()` functions, which serve as a sort of HashMap on every page element, letting programmers create any attributes they might need. Finally, you looked at how to modify the CSS of a page element without having to reload the page. You saw that while the `css()` function is easy and straightforward, it is probably in the best interest of you and your team to replace these functions with the `addClass()` and `removeClass()` functions, in order to keep the style of the page separated from the jQuery code.

The final part of the article tied together the three modules you learned and showed how the sample Web mail application deals with mouse interaction, differentiating between a single mouse-click, by highlighting the row clicked, and a double-click, which marks the message "unread" if appropriate and then makes an Ajax call to the server for the message-specific data, passing the message number to the server in the process.

In the next article in the series, we'll take a deeper look at the Ajax functionality in jQuery and see how it removes a great deal of the complexity working with Ajax, making it as simple as calling a JavaScript function like you normally would. Additionally, we'll take a look at the Effects module in jQuery, which is useful for creating additional interaction and visual clues to users—and let's be frank, it's cool, too! The final part of the next article will wrap up the demo Web mail application and our jQuery lessons, hopefully convincing you that you should add this library to your own Web applications.

Downloads

Description	Name	Size	Download method
Zip file containing the sample application	jquery.zip	68KB	HTTP
War file containing the sample application	jquery.war	68KB	HTTP

[Information about download methods](#)

Resources

Learn

- 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](#)", (Jesse Skinner, developerWorks, April 2007).
- "[Ajax overhaul, Part 2: Retrofit existing sites with jQuery, Ajax, tooltips, and lightboxes](#)" (Brian Dillard, developerWorks, May 2008) is a look at some of the plugins available for jQuery and how they expand the usefulness of the library.
- Expand your Web development skills with articles and tutorials that specialize in Web technologies in the developerWorks [Web development zone](#).

Discuss

- Participate in [developerWorks blogs](#) and get involved in the [developerWorks community](#).

About the author

Michael Abernethy

In his 10 years in technology, Michael Abernethy has worked with a wide variety of technologies and a wide variety of clients. He currently works as the Product Development Manager for Optimal Auctions, an auction software company. His focus nowadays 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.