

Working with jQuery, Part 1: Intermediate JQuery: Using plug-ins to create and extend the jQuery functions

Skill Level: Intermediate

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

17 Feb 2009

The popularity of jQuery owes a lot to its decision to include a plug-in architecture. This decision allows any number of third-party developers to create and extend the jQuery functions beyond the original library functions. The result is hundreds of plug-ins that provide nearly any type of function needed on a Web application. This article describes this plug-in architecture and explains how jQuery can help your Web application behave just like a desktop application.

Introduction

In the past six months, since I wrote the first series of articles (see [Resources](#)) on the jQuery JavaScript library, a lot has happened in the jQuery space. Perhaps most exciting for those of us who have embraced jQuery is that Microsoft® has chosen to use jQuery in its Visual Studio suite, and has decided to make it the only JavaScript library included at this time. This is a tremendous show of support, one that can only help to solidify jQuery's position as the leading JavaScript library for Web applications. Another convincing point of jQuery's growing popularity is the updated Google Trends chart. The chart I featured in the previous articles showed that jQuery was beginning to pull away from alternative JavaScript libraries. Well, six months later, and the separation has become even more pronounced, as you can see from the updated Google trends charts shown in Figure 1 and Figure 2 (ignoring the stock market-like dip in December).

Figure 1. June 2008 Google trends of common JavaScript libraries

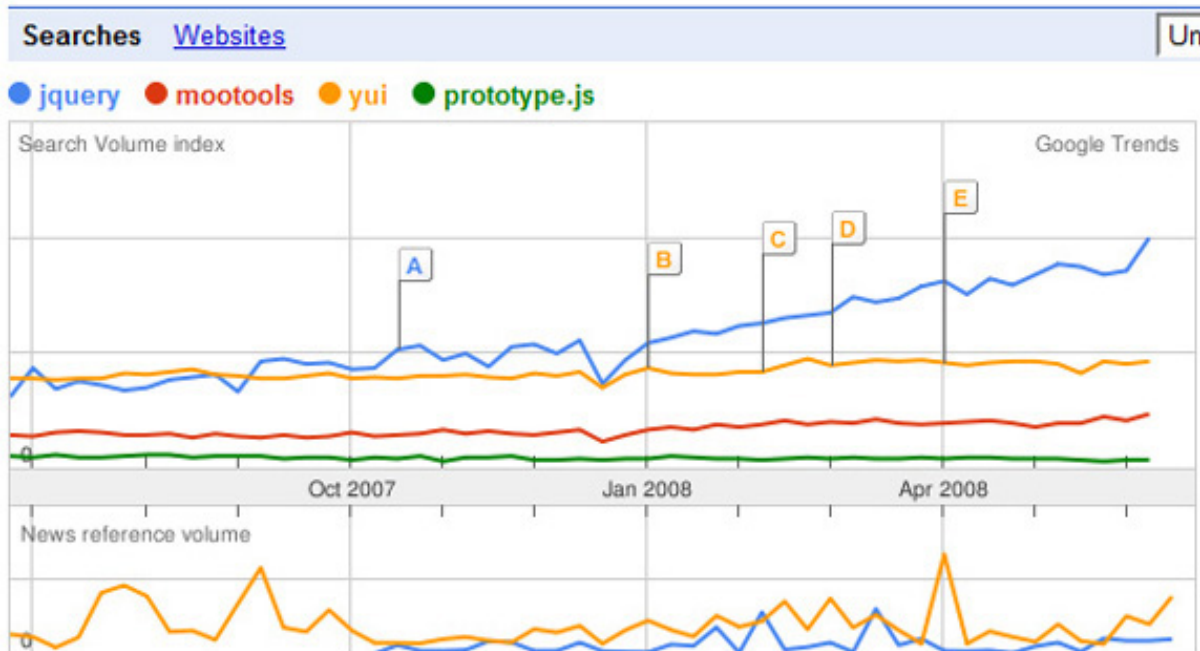
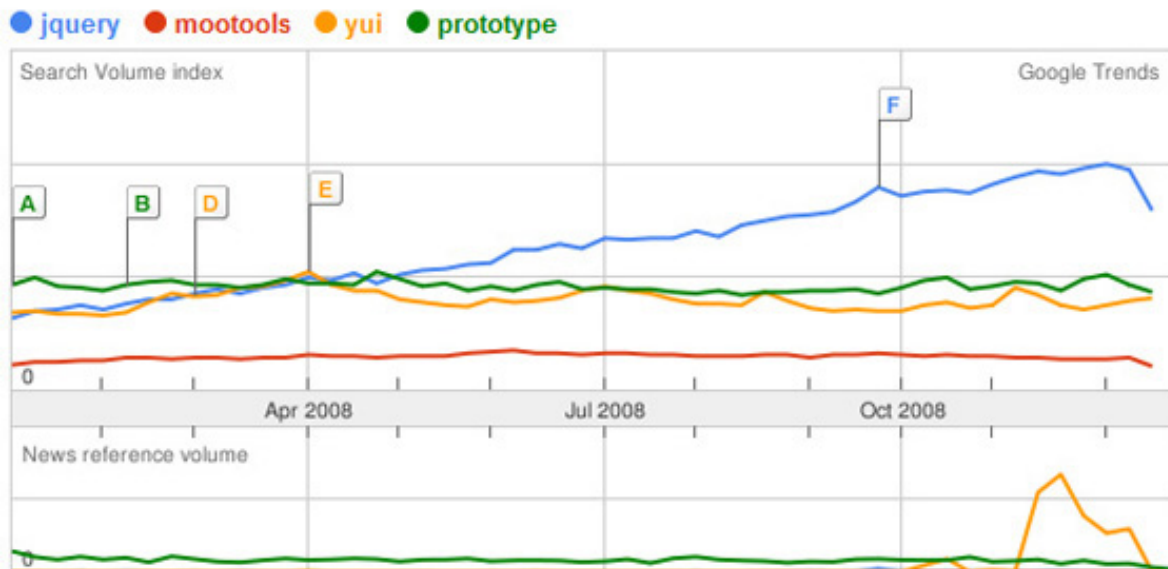


Figure 2. January 2009 Google trends of common JavaScript libraries



In this series of five articles, I'm looking to 'step it up a notch,' and take on some of the more intermediate-level topics in working with jQuery. This series touches upon plug-ins and plug-in development, the jQuery UI package, more advanced topics in creating jQuery widgets, more advanced Asynchronous JavaScript + XML (Ajax) techniques, and finally, looks at the performance of jQuery compared to JavaScript and other libraries.

This first article in the series discusses the plug-in structure used in jQuery. Plug-ins are likely the main reason jQuery has grown so much quicker than the other

JavaScript libraries. Because they have been embraced by third-party developers, hundreds of plug-ins have been developed to add to the jQuery functionality. Plug-ins, as their names suggest, are widgets or code modules that literally "plug-in" to jQuery and extend the functions that are built into the jQuery core download. The plug-in community has produced hundreds of plug-ins, and that's not even an exaggeration. Whatever issue you may face on your Web site, whatever widget you might be looking for (or your client is asking for), there is likely to be an existing solution in the jQuery plug-in library. Best of all, all of the plug-ins are free to download and use on your own Web site.

Plug-ins aren't just limited to user interface widgets, but they can include extensions to the jQuery syntax, additional Ajax functions, and other creative things that people might need to improve their development process. One of the cooler things is that people have translated several of the built-in JavaScript features (for example, the threading features, `setTimeout()` and `setInterval()`) into jQuery syntax. This lets developers have a completely jQuery environment, making things easier to work with and easier to maintain.

Plug-ins

The jQuery plug-in structure has many benefits. First, it allows you to use only the widgets and functions you want to use outside of the jQuery core. This is important in Web applications because each additional plug-in is an additional download, and additional traffic. By allowing you to only use the plug-ins you want to use, you can better manage your Web traffic. Second, it lets eager and ambitious third-party developers create interesting widgets and improve the jQuery functions by creating their own plug-ins, rather than attempting to put their ideas into the jQuery core code. This enables the collective creative juices of everyone who uses jQuery to expand the library, creating a nearly limitless growth of new ideas and new widgets. This is the opposite of the closed-end structure, which would require the jQuery team to review and approve each plug-in, which would create a real creative bottleneck. Third, this plug-in architecture created by the jQuery team is very easy to use, both for developers creating the plug-ins, and the developers using the plug-ins, which has factored greatly into the explosive growth of the plug-ins. However, all these benefits have one negative aspect: the fact that the plug-ins have no formal testing structure outside of peer reviews. So, while you can be assured that the jQuery core is thoroughly tested, by choosing to use a plug-in, you are placing your trust in a third-party to test them. For a mission critical Web application, this drawback should always be considered.

To use a plug-in, you simply have to include it on your page, much as you do any JavaScript file (including the jQuery file itself). So, if you need to use a plug-in on your page, you can add it right after the jQuery itself, as shown in Listing 1.

Listing 1. How to include a plug-in

```
<script src="jquery-1.2.6.min.js" type="text/javascript"></script>  
<script src="jquery.alphanumeric.plus.js" type="text/javascript"></script>  
<script src="jquery.blockUI.js" type="text/javascript"></script>
```

This article doesn't go into the details of how to write a plug-in or the specifics of how they work. Those details will all be in a future article, in which you write your own new plug-in and place it into the actual jQuery plug-in repository. Instead, this article goes through several of my favorite plug-ins. The goal is not only to show you the plug-ins that I tend to use every day, but also to give you a taste of what is available on the jQuery plug-in repository. Hopefully, you'll get interested enough in plug-ins to check them all out for yourself.

So, onward with my favorite plug-ins.

RightClick, ExtendedClick, and Wheel

One of the biggest goals of any Web application is to "fool" the users into thinking they are working on an actual desktop application. When I say "fool," I really mean that the Web application is trying to mimic the look and feel of a desktop application as closely as possible. Therefore, if a user typically expects a toolbar to appear in a certain place on their desktop application, the Web application should attempt to place the toolbar in the same spot. This eases the transition for users from the desktop application to the Web application, and therefore, makes the chances of success of the Web application greater.

For the most part, however, one area of Web applications that has not closely matched desktop applications is the response to mouse events. Some of you may be asking what I mean, because your Web application works just fine when you click the mouse on it. Sure, for the most part, every Web application handles the left mouse click perfectly fine. In fact, so many people have become accustomed to using only their left mouse button on a Web site, that a Web application is like an Apple application, using only one button. But, all your Windows® applications use two mouse buttons (sometimes even a third). The left mouse button activates commands, and the right mouse button presents options. People are very accustomed to this usage, so why are so many Web applications ignoring the right mouse button? Taken a step further, Web applications are also ignoring the CTRL+click and the Shift+click, as well as the mouse wheel. How can a Web application truly mimic the desktop application if it ignores so many of the mouse's actions. A true Web application would take full advantage of all the uses a mouse can provide.

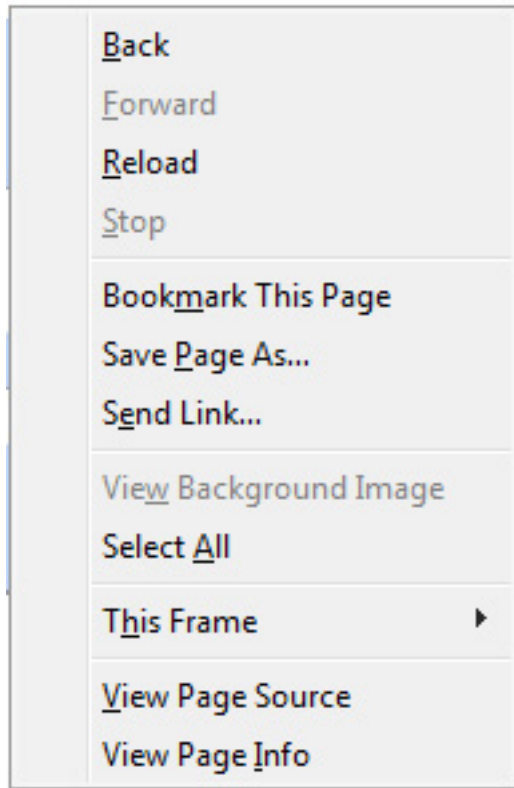
To truly see the difference between a Web application that ignores the right mouse click and one that doesn't, and to see how a Web site's functions can be enhanced by the extra mouse button, take a look at GMail versus Yahoo Mail. Sorry to say for

all you GMail fans, but in my opinion, Yahoo Mail is better in this respect. In GMail, the right mouse click is not treated differently than it is on any other Web page. So, when you right click on a message in GMail, it presents you with options like "Back," "Inspect Element," or "Select All." How are these actions even relevant to working with your messages? Well, they aren't of course, which really means the right mouse button is useless on this page. Contrast that with Yahoo Mail, and how it deals with the right mouse click. When I right click on a message in Yahoo Mail, I am presented with options "Open," "Print," "Reply," and "Delete," These are actions that normally go with a mail application. These choices make a complete Web application, one that mimics the type of mail application you'd see on the desktop (try right clicking on a message in Outlook, and see which Web application more closely matches the choices). Figure 3 and Figure 4 show the differences in the right click options.

Figure 3. Right clicking in Yahoo Mail

Open	Enter
Print	Ctrl+p
Reply to Sender	r
Reply to All	a
Forward	f
Select All Messages	Ctrl+a
Mark as Read	k
Mark as Unread	Shift+k
Flag for Follow-up	l
Clear Flag	Shift+l
Delete	
This is Spam	
Add Sender to Contacts	
View Full Headers	

Figure 4. Rick clicking in GMail



The first jQuery plug-in to help solve the mouse-click problems in Web apps is the appropriately named `rightClick` plug-in. It captures both a right mouse-click, and also a right mouse-down and a right mouse-up. Finally, the last thing it does is let you turn off the browser-specific right-click context menu. So, the right-click menu that appears in Firefox (Figure 4), will not appear if you turn it off, allowing you to create your own custom right-click menu without competing with the browser's default behavior.

Listing 2. `rightClick` plug-in

```
// set up the div that will capture our events
<div id=rightClickSample></div>

// when the right mouse is clicked on this div, increase the width
// by 10 pixels. Also, do not show the browser-specific pop-up
// menu
// This, of course, should be in the $(document).ready function

$("#rightClickSample").rightClick(function(e){
    $(this).width($(this).width()+10);
});

$("#rightClickSample").noContext();
```

Now, let's examine the next plug-in that can extend the mouse in Web applications. This plug-in adds the ability to capture the Ctrl, Alt, and Shift buttons. Some applications make regular use of these buttons (Adobe Photoshop, for example), so

some people are rather comfortable with these buttons when using their mouse. However, using these buttons with the jQuery core code is difficult and requires extra coding. Why not use the premade, and pretested plug-in?

The `extendedClick` plug-in offers different functions that represent the combination of different helper keys you can press with a mouse. As expected, the functions include `ctrlclick()`, `shiftclick()`, `altclick()`, `ctrlaltclick()`, `ctrlshiftclick()`, `altshiftclick()`, and `ctrlaltshiftclick()`. Unfortunately, these can only be attached to the left mouse-click button, and as you've seen from the right-click examples, it would be foolish to ignore half of the mouse. However, a plug-in to attach helper keys to the right-click doesn't exist (yet), so I leave that up to the ambitious out there to merge the two of them together and create a new plug-in.

Let's change the example from the right-click, and make the `div` grow 10 pixels when you press a Shift+left click and shrink 10 pixels when you press a Ctrl+left click.

Listing 3. `extendedClick` plug-in

```
// set up the div that will capture our events
<div id=extendedClickSample></div>

// when the left mouse is clicked with the shift key held down,
// grow the div by 10 pixels
$("#extendedClickSample").shiftclick(function(e){
    $(this).width($(this).width()+10);
});

// when the left mouse is clicked with the ctrl key held down,
// shrink the div by 10 pixels
$("#extendedClickSample").ctrlclick(function(e){
    $(this).width($(this).width()-10);
});
```

Finally, the last plug-in to bridge the gap between desktop application and Web application is the plug-in that handles the mouse wheel. I would wager a bet that most people have never even visited a Web site that allowed you to use your mouse wheel (excluding an HTML input element). However, that doesn't mean it's impossible to use the mouse wheel in an application. One use for a mouse wheel in an application is in Web sites that let you upload photos, to let you zoom in and out with the mouse wheel. Yes, you can do that on desktop applications now, but can you do that on most Web applications? Probably not.

For the mouse wheel example, let's use the same growing/shrinking `div` and let the mouse wheel control its size. So, when you move the mouse wheel up, it will grow the `div`, and if you move the mouse wheel down, you will shrink the `div`. And, if you don't have a mouse wheel, then you can just assume this is working perfectly.

Listing 4. Mouse wheel plug-in

```
// set up the div that will capture our events
<div id=wheelSample></div>

// attach an event handler for the wheel to this div
// notice that we use the e.delta to determine how many "notches" the wheel
// was moved. One notch is either 1 (if up), or -1 (if down). So, as
// a result, we can simply add it to the current width, letting the
// sign of the delta grow or shrink the div.
$("#wheelSample").wheel(function(e){
    $(this).width($(this).width()+10*(e.delta));
});
```

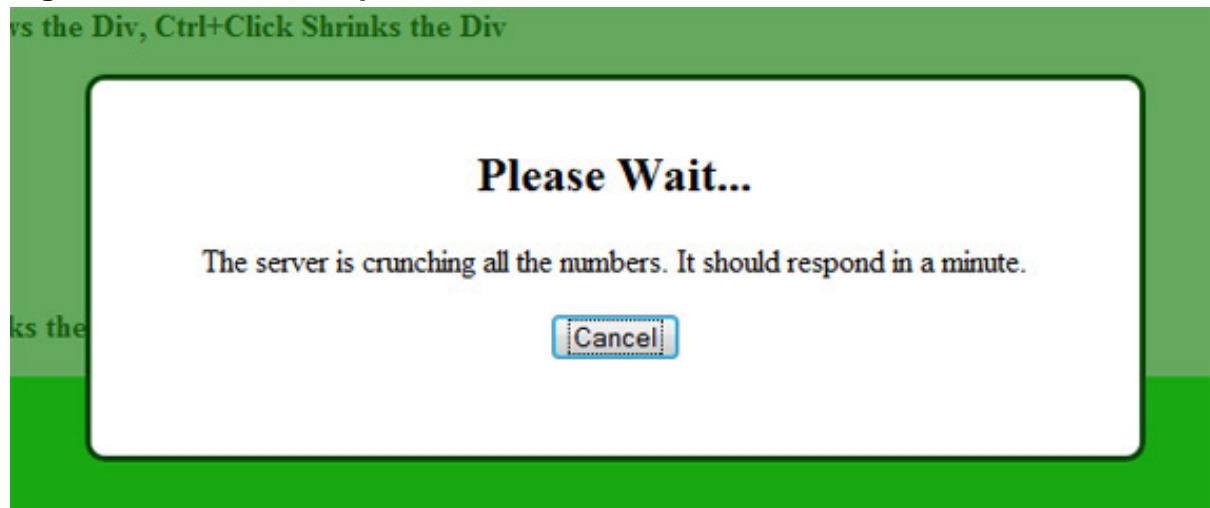
These three plug-ins, I feel, are very important in making a Web application look and feel like a desktop application. Instead of being limited to the left mouse-click when interacting with a Web application, a limitation for sure, these plug-ins allow the Web application to capture all forms of mouse interaction, just as a typical desktop application would do. Additionally, as will become increasingly common over the next few years, developers will be tasked with converting existing desktop applications into Web applications. These plug-ins will make that transition much easier, by supplying all the mouse interaction on the Web that users have had on the desktop.

blockUI

The blockUI is one of several plug-ins that offer the ability to create modal dialogs. Wait, you're thinking, JavaScript already offers some really cool modal dialogs like `alert()` and `confirm()`. Yes, I can't deny how awesome those modal dialogs are for developers, and how much users love them. They just look so good on a well-designed Web site. However, there's an even better option available to developers, one that actually lets you collect information from a user, and one that actually lets you make the modal dialog look like it belongs on your Web application. Note, alert/confirm fans can feel free to skip ahead to the next section.

All sarcasm aside, the alert/confirm functions have major drawbacks in terms of modal dialogs. First and foremost, they can only present information and can't collect any information, outside of an OK or Cancel on the confirm function. It would be ideal if there was a dialog window pop-up that offered more flexibility, and one that lets you add any elements you wanted to the dialog. In other words, a dialog window like you are used to in other programming languages. BlockUI is my favorite plug-in for this problem, as I find it easy to work with, and extremely flexible for working with many dialogs on the same Web application. Not only that, it offers many different types of dialogs to work with.

In the sample code included with this article (see [Downloads](#)), I create a modal dialog using blockUI, and you can see what it looks like before I delve into the code and options.

Figure 5. blockUI example

As you can see in Figure 5, there is a modal dialog on the page. The input to the rest of the page is blocked by the semi-transparent layer beneath the dialog, effectively blocking the user from interacting with the Web application, except the dialog itself. The code accomplishes this blocking with an IFrame, but it doesn't really matter, because the plug-in author is the one that tested it on all the different browsers and ensured it worked.

The first step in creating the dialog is to specify its message. A message can be anything from the text you want to display to the user (replicating the `alert()` function), to displaying HTML to the user, to creating a marked-up message (with bolds colors, and so on). However, I find this plug-in most valuable by letting you supply `divs` as the message, effectively letting you create your own dialog, with any layout you want, and with any number of inputs, buttons, and information you need.

Listing 5. blockUI plug-in

```
// this will create a dialog with our default text,
// effectively replicating the
// alert() function
$.blockUI({message: "This is a sample dialog"});

// however, we can add any HTML we want to the message, making it look closer
// to our own Web site
// $.blockUI({message: "<h2>Sample Dialog</h2><p style='color:green;'>
//   This is our message</p>"});

// perhaps most importantly, we can add a div to the message, allowing
// us to create our own dialog, with the look, feel, message, and input we want
<div id=loginMessage style="display:none;cursor:default;">
  <p><h2>Login</h2>
  <p>Username: <input type=text id=user>
  <p>Password: <input type=text id=pass>
  <p><input type=button value="OK" id=ok>
  <input type=button value="Cancel" id=cancel>
</div>

// and the jQuery code to show this is pretty much the same
```

```
$.blockUI({ message:$("#loginMessage") });
```

There's just two small issues remaining to discuss on the blockUI before you rush off to put it on your own site. First of all, I haven't closed the dialog yet. That's a minor issue, right? Second, all of the dialogs I created in the code snippet in Listing 5 have the default blockUI look/feel, which isn't desirable, because one of the big selling points of the plug-in was to create my own look and feel for the dialogs. Well, the blockUI comes with a snippet of CSS code, which you can take and throw in your own CSS file and customize any way you want, giving you back the ability to make the dialogs look the way you want them to. However, before you can override the CSS attributes of the blockUI, you must tell the plug-in to look in the CSS file rather than using its own defaults. In the code snippet shown in Listing 6, I display the code I have in the example code, which will show you how to close the dialog, how to use your own CSS, and what options in the CSS you have.

Listing 6. More blockUI plug-in

```
// These two functions tell the BlockUI that
// you want to use your own CSS
// code to define how the dialogs will work. The first line
// tells it you want to use your own CSS for the dialog,
// the second line tells it you want your own CSS
// for the semi-transparent
// layer between the page and the dialog.
$.blockUI.defaults.css = {};
$.blockUI.defaults.overlayCSS = {};

// when the show button is pressed, we'll display the dialog.
// we want to display our own custom DIV. However, note here
// that we want to override the CSS-defined height and width.
// After all, it would be difficult for an entire site to have
// predefined widths and heights if every dialog is slightly
// different.
$("#show").click(function(){
    $.blockUI({ message:$("#waitMessage"), css: {width:'500px', height:'160px'} });
});

// When the cancel button is pressed on the DIV dialog,
// we can close the dialog
$("#cancel").click(function(){
    $.unblockUI();
});

// the CSS you can override to make it look good on your site
div.blockOverlay
{
    -ms-filter: "progid:DXImageTransform.Microsoft.Alpha(Opacity=50)";
    filter: progid:DXImageTransform.Microsoft.Alpha(Opacity=50);
    -moz-opacity:.70;
    opacity:.70;
    background-color: #228518;
}

div.blockMsg
{
    width: 20%;
    top: 20%;
    left: 30%;
    text-align: center;
}
```

```
background-color: #fff;
border: 3px solid #044600;
-moz-border-radius: 10px;
-webkit-border-radius: 10px;
-ms-filter: "progid:DXImageTransform.Microsoft.Alpha(Opacity=100)";
filter: progid:DXImageTransform.Microsoft.Alpha(Opacity=100);
-moz-opacity: 1;
opacity: 1;
padding: 15px;
color: #000;
}
```

In my experience, this plug-in comes in handy in my situations. I have found it very quick to implement when I want to replace the `confirm()` function in JavaScript, not to mention how much nicer it looks. I have also used it in more involved situations, using it as a dialog to upload files on an e-mail application, or as a photo lightbox in a picture viewing application. Combining it with Ajax calls can save a lot of page reloading, by presenting information that's not immediately available by getting the information and overlaying it on the screen. Effectively, it enlarges the screen area with which to display information. Finally, it removes the alert/confirm function limitations and lets you design your software with dialogs in mind, just as you would a desktop application.

printArea

The `printArea` plug-in fills a nice niche in Web applications, and I've found it to be quite useful. At its core, it simply lets you print a specific HTML element. This is a real bonus over the alternatives to printing on a Web application, which is simply to call the `window.print()` button and hope for the best. As anyone who has ever worked with Internet browsers can tell you, you're unlikely to have two pages print out exactly the same on different browsers. Also, a problem arises when you want to print out all the text inside of a text area and nothing else. Until now, this was very difficult.

As I said, this plug-in lets you print specific elements on a page, and not print the rest of them. Examples where I have used this plug-in are in an e-mail application, where you allow a user to print a message that's contained in a text area. I also have used it on pages that display reports, where I don't want the various buttons and graphics on the page to be printed with the report. An example people are probably also familiar with is printing airline tickets. The ticket on the Web page usually has lots of information and graphics, whereas the one that gets printed just contains a bar code and basic boarding information.

Listing 7. `printArea` plug-in

```
<p>An advertisement that you don't want printed out.
<p>Another advertisement that you don't want printed out.
<div id=printable>
```

```

<table width=40% cellpadding=3 cellspacing=0 border=1>
<tr><th>Name</th><th>Age</th><th>Height</th></tr>
<tr><td>John Q</td><td>23</td><td>6'1"</td></tr>
<tr><td>Jane Q</td><td>23</td><td>5'1"</td></tr>
<tr><td>Jimmy R</td><td>23</td><td>5'6"</td></tr>
</table>
</div>
<p><input type=button id=printButton value="Print">
<p>All the annoying disclaimer text you don't want printed out.

// Capture the click on the "Print" button. Then, you can call the printArea()
// function on the "printable" div we used to wrap the portion of the page we want
// printed. The only thing that will get printed to the printer will
// be the table, and all the annoying text won't be.
$("#printButton").click(function(){
    $("#printable").printArea();
});

```

This plug-in offers a really easy and straightforward way to control the printing of elements on a page. As an added bonus it works equally well on Internet Explorer and Firefox, and can be used to control how your page gets printed in both browsers, saving you some major headaches. The only drawback is that it relies on the user to press *your* Print buttons, or other areas of the page that you can control. If the user presses the browser's Print button, it will revert to the old problematic printing. Doubly troublesome, you can't stop that print button from executing, or even issue a warning about using it. So, you have to rely on your users being smart enough to press your own Print button. The other alternative, of course, is to simply print it automatically under certain conditions, taking the user out of the equation.

AlphaNumericPlus

It is common in user interface design to code for lower user assumptions. A good example of lowering your user assumptions is when working with text fields on a form. You have a field set up to capture phone numbers, but you know someone will enter letters.

You can prevent this incorrect input in three places. The first is checking for bad input on the server, before it gets entered into the database. So, some Java™ code or PHP code can be written that will check all the input fields and decide which ones have valid input and which ones don't and then send a message back to the client if they aren't valid. The second level of error checking is to check on the client side, using JavaScript. By checking these conditions on the client side, you can save on network traffic and get quicker responses to the user. The final level of error checking takes full advantage of the "lower user assumption" design pattern, which is to not even let them enter bad information in the first place. In the example discussed here, why send letters to the server for a phone number, or why have JavaScript check for letters before submitting the form. The best solution is to simply not let them type any letters in at all.

This plug-in is built on that idea, that the Web application's text fields should only

accept the input that I deem appropriate. So, for a field that should only have numbers, I want my text field to only accept numbers. If I want it to only have letters, it should only accept letters. You can see where I'm going with this.

Listing 8. AlphaNumeric plug-in

```
<p>Alpha-only: <input type=text width=20 id=alphaOnly>
<p>Numeric-only: <input type=text width=20 id=numericOnly>

// This code will prevent unworthy characters from being entered
// into our text fields, assuring valid input
$("#alphaOnly").alpha();
$("#numericOnly").numeric();
```

The standard AlphaNumeric plug-in ends there, though I took it and expanded it for my own purposes, so I will discuss those additions here (they are included in the [download](#)). I thought that if I could control when only letters or numbers were entered, I could also expand it to include special characters like \$, %, or &. And assuming that, I could attach certain rules to text fields depending on their use in the form. For example, a text field that is being used for e-mail addresses should allow all the characters that are valid in an e-mail address beyond the standard numbers and letters. So, an e-mail address field could accept _, -, @, and ".". All the other special characters are invalid and would cause an error if used in an e-mail, so we block those from being typed. Are we 100% guaranteed that this will be a valid e-mail address? No, the user could still type something ridiculous like "a@a@.com," but it's a first step.

The additions that I've made to the standard AlphaNumeric plug-in deal with decimals, currency, e-mail, phone, clocks, and dates. Listing 9 shows my code.

Listing 9. AlphaNumericPlus plug-in

```
// will only allow numerals and the .
// and , characters (because I'm thinking international
// here, and some countries use a , for a decimal)
$("#decimalOnly").decimal();
// same as the above, but it adds support for the "%"
$("#percentOnly").percent();
// as described above, only allows valid e-mail characters
$("#emailOnly").email();
// allows only numbers and the ( and ) and - characters
$("#phoneOnly").phone();
// allows only numbers and the : character
$("#clockOnly").clock();
// allows only numbers and the / and - characters
$("#dateOnly").date();
// this is the only one that's slightly different, in that
// it requires you to pass in the valid currency symbol, so you can
// pass in a pound sign or euro if you're using this overseas
$("#currencyOnly").currency("$");
```

This AlphaNumericPlus plug-in is the first level of defense against invalid input from

users. By not allowing invalid characters in certain instances, it not only minimizes the chance of errors, it also offers immediate reinforcement to the user that they are not entering valid information. So, if a user attempts to enter a letter in a phone number field and nothing appears, the user is immediately notified that they have an error. This quick response is often appreciated by users, rather than making them wait until the end of the process to fix all the errors.

The second level of defense is the client-side check upon form submission. In this case, you can check that the data is all in a valid format; for example, checking that the phone number has the correct number of digits. Finally, the last level is the server-side check, where you can double-check that the format is valid, and also check the information against anything already in the database. A successful three-pronged attack is the best solution, with this plug-in serving as that first line. Though not perfect (you can still enter an invalid format, although all the characters are valid), it presents immediate feedback for users.

Calculation

Another plug-in that I have found useful in certain situations is the Calculation plug-in. Like most good plug-ins, it does only a few things, but it does them well. Like its name implies, it calculates numbers in fields and lets you find the sum, the average, the maximum, or the minimum of them. It also lets you parse a number from a field. The part that gives this plug-in its greatest flexibility is its ability to sum numbers from any type of element. So, if you pass it an array of elements that includes a DIV, an INPUT TEXT, and a SPAN, it will find the numbers within those fields, convert them to numbers, and return the sum. This can be useful in certain situations, and can save you from a lot of headaches trying to error-check every mathematical function.

One warning about this plug-in, though, is that it breaks the standard plug-in decree that all plug-ins return the jQuery object itself. In every other plug-in discussed in this article, and every other plug-in I've worked with, the function returns the jQuery object, meaning it doesn't "break the chain." This does not adhere to that. By returning numbers, it breaks the chain, so that you can't daisy-chain jQuery functions together after calling it.

Let's look at a slightly more extended and involved example for this plug-in. The widget shown in Figure 4 has a set of fields that allow a user to enter percentages. The widget checks that all of the fields add up to 100%, and if they don't, will show the sums of the percentages in red, to indicate an error. You likely encounter a widget like this in your financial planning pages, asking you to decide where to place your contributions.

Figure 6. 401k contribution widget

Enter the percentage allocation you wish to place your 401k contributions into. All contributions must add up to 100.

S&P 500 Index	<input type="text" value="5"/>	%
Russell 200 Index	<input type="text" value="32"/>	%
MSCI International Index	<input type="text" value="23"/>	%
MSCI Emerging Market Index	<input type="text" value="34"/>	%
REIT Index	<input type="text" value="0"/>	%
Total	94	%

Listing 10. Calculation plug-in

```
// Set up the table that contains the widget.
// Note that we add a class to it called
// "percentSimple", which will be used in the jQuery
// code, as well as adding a unique
// ID to the table "sortTable1".
// Also note that each text field that will have a percent
// typed into it has the
// class of "percent" added to it.
// Finally, note in the table footer a field called "percentTotal"
<p><table width=300 class="percentSimple" cellpadding=0 cellspacing=0 id=sortTable1>
<tbody>
  <tr><td class="left">
    S&P 500 Index
  </td>
  <td>
    <input type=text class="percent textfield"> %
  </td>
</tr>
  .....
<tfoot>
  <tr><td>
    Total
  </td>
  <td>
    <span class=percentTotal></span> %
  </td>
</tr>
</tfoot>

// take advantage of our previous plug-in and limit the percent fields
// to just numbers
```

```
$(".percent").numeric();

// capture any keyup events from all the "percent" fields
// when these occur, we are going to recalculate the total percent
$("table.percentSimple input.percent").keyup(function(){
    // figure out which table this occurred in (in case there's more than
    // 1 widget on a page
    var table = $(this).parents().filter("table.percentSimple");
    var ID = table.attr("id");
    // Find the sum of all the "percent" fields, by using
    // our calculation plug-in
    var sum = $("#" + ID + " input.percent").sum();
    // cache the span called "percentTotal"
    var totalField = $("#" + ID + " .percentTotal");
    // update the total in the "percentTotal" field
    totalField.html(sum);
    // if the sum != 100%, then we'll attach an error to it
    if (sum != 100 && sum != 0)
    {
        totalField.addClass("error");
    }
    else
    {
        totalField.removeClass("error");
    }
});
```

As you can see, this plug-in has limited usage, but it is very effective and saves time when used correctly. Though I have only included the max/min, average, and sum functions, I can picture someone who's quite ambitious putting together an entire spreadsheet plug-in, which includes every function found in the Microsoft Excel application. Picture this spreadsheet plug-in providing functions for standard deviation, payment, and future value (because I work with finance, I know those functions best). Further, imagine that the spreadsheet plug-in could be used by Web sites looking to provide basic spreadsheet functions on their pages or allowing users to create their own spreadsheet-like pages without entering any complex formulas.

Timeout/Interval

One of the more frustrating things for me about JavaScript is its incompatibility with common thread designs. When I moved from Java user interfaces in Swing to the JavaScript Web applications, I found that I could not replicate in JavaScript the common multithreaded interfaces found in the Java code. Instead, JavaScript uses the `setTimeout()` and `setInterval()` methods, which are somewhat like the threading design I was used to in Java code, but not completely the same. The `setTimeout()` method takes a string argument, which serves as the callback method, and a number, which serves as the timeout, in milliseconds. Likewise, the `setInterval()` method takes the same arguments, although this method calls the callback method over and over, with the timeout being the break between calls.

Compare this to the Java thread design, which lets you create a thread object and then call `start()` and `stop()` on the same object, controlling when it runs, and defining the `run()` method right there in the class definition, and you can see the

difference between the two designs. The JavaScript thread design lets you start threads easily, but it's sometimes difficult to stop them. This is because the JavaScript thread design relies on IDs to stop timeouts/intervals. These IDs are passed back from the `setTimeout/setInterval` function. Take a look.

Listing 11. JavaScript threading

```
// creates a thread with ID of threadID,  
// which will call myCallBack()  
// every 1 second (1000 ms = 1 sec)  
var threadID = setInterval('myCallBack', 1000);  
  
// will stop the thread from calling myCallBack()  
clearInterval(threadID);
```

The code in Listing 11 wouldn't really be practical, though, because you'd be turning off the thread before it even runs once. Calling `setInterval()` and on the next line calling `clearInterval()` effectively does nothing, because there's no pause after the `setInterval()` call. The workaround to this is shown in Listing 12.

Listing 12. More JavaScript threading

```
$(document).ready(function() {  
  // store the ID as a global variable  
  var threadID;  
  
  // creates a thread with ID of threadID, which will call myCallBack()  
  // every 1 second (1000 ms = 1 sec)  
  threadID = setInterval('myCallBack', 1000);  
  
  // will stop the thread from calling myCallBack()  
  $("#hypotheticalStopButton").click(function(){  
    clearInterval(threadID);  
  });  
});
```

This code would work because it stores the `threadID` as a global variable that can be referenced anywhere in the JavaScript code. This solution would work fine for a simple example, but what happens when you start introducing more threads into the Web application and the interactions start becoming more difficult. It would be harder to maintain code that has a bunch of global variables in the JavaScript code, hoping you pick the right thread to stop/start at various times. Add to this problem another situation that is common in Web applications: you want a function to be called 10 times, no more and no less. That adds another global variable into the mix. Plus, you can't create an inline function in your thread creation, you have to reference a stand-alone function. You can see the complexity starting to increase rapidly as the complexity of the threading increases on a page.

The Timers plug-in aims to simplify the issues of working with threads on a Web application in JavaScript by really changing the way you work with them, and, in my opinion, making it closer to Java code. Instead of relying on the `threadID` to get

passed back from the `setInterval()` function, the Timers plug-in changes the design to let you choose the thread name upon creation, making it unnecessary to store all of the thread IDs to use other places in the code. Along with other beneficial changes, it's this subtle change that has made working with threads in JavaScript much easier. Take a look at Listing 13 to see how the sample code in Listing 12 can be changed to work with the Timers plug-in. See if you agree with me that it appears to be much easier to work with.

Listing 13. JavaScript threading With timers

```
// attach an event to the start button,
// kicking off the thread
$("#start").click(function(){

// using the Timer plug-in, we attach a thread to a page element
// we call the "everyTime" function, which is analagous to the
// setInterval function.
// This function takes several arguments:
// - the timeout interval
// - the name of the thread, notice how we can choose the name here
//   and not have to worry about storing the value anywhere
// - the function to call each interval, notice how we can define
//   the inline function here, without creating an extra function
//   elsewhere, simplifying development of the thread.
// - (optional) how many times to run it before stoppping, so if
//   you put a 10 there, it would run it 10 times and then stop
// - (optional) a true/false of whether to start the next interval
//   if the previous one isn't completed yet
$("#timerSample").everyTime(500, 'growSquareThread', function(){
  var t = $(this);
  t.height(t.height()+5);
  t.width(t.width()+5);
});

// attach an event to the stop button. Notice how we can simply
// call a stop with the name of the thread, without worrying
// about the threadID
$("#stop").click(function(){
  $("#timerSample").stopTime("growSquareThread");
});
```

The Timers plug-in really makes working with threading in JavaScript easier. Even though the changes may seem subtle at first, when you take a look you can hopefully see how it will make your code much cleaner. Primarily, you can define the entire thread in one place. That means defining the start mechanism, the delay, the number of loops to make, and the code to run each iteration. Not only is this easier to work with than traditional JavaScript threading, I'd even say it's easier than Java threading. With the Timers plug-in, the entire thread itself is self-contained in a block of code. No more making references to functions that could be anywhere in the code. No more tracking `threadID`'s in global variables. No more complicated code to keep track of how many iterations a thread has gone through. Stopping the thread is just as easy. Some trigger to stop it can simply use the same thread name that it was created with. No need to even know about the `threadID` any more.

Conclusion

This brings to a close this article on plug-ins, but it by no means should end your exploration of all the plug-ins available in jQuery. I presented a mere eight plug-ins of the 200-400 plug-ins that are available on the jQuery plug-in site. Though I only presented a small sampling, I think the ones that I have chosen to discuss here represent important plug-ins that can help bridge the gap between a desktop application and a Web application. These plug-ins make an effort to fill in the gaps that currently exist in JavaScript and jQuery. Think for a second, what you've learned from these eight plug-ins, and how they could help you in your work, and then think what the other hundreds of plug-ins could do for you. I encourage everyone to spend some time and just browse the plug-in repository on the jQuery site. That is how I found a few of the plug-ins mentioned here, just browsing the site and looking through each of them. The other primary way of finding a good plug-in is to run into a requirement in your own work. From now on, the first thought you should have is "is there already a plug-in that does this?". After all, why repeat work that someone else has already done.

Finally, let's review the plug-ins I covered in this article and how they improve the Web application. The three mouse-click plug-ins enable a Web application to move beyond the simple left mouse-click, and capture all the possible interaction you can receive from a mouse. Because desktop applications already do this, these plug-ins really fill a user interface hole that existed in Web apps. The next plug-in I discussed was the blockUI, which lets you create dialog windows that can match the look of your Web site, allowing you to design your own dialog windows. Because Web applications were previously limited to `alert()` and `confirm()`, this is a big improvement in how you interact with the users of your Web site. The next plug-in discussed was the printArea plug-in, which gave you the ability to print only specific portions of your Web page, while excluding others. Again, this is something that desktop applications could do, but was limited on Web applications. The next two plug-ins were the calculation and the alphanumeric plug-ins, which made the creation of specific widgets easier. These ideally will help to reduce errors in your code and provide a quicker response mechanism to your users. Finally, you looked at the Timer plug-in, which remodeled how you work with threads in JavaScript.

The one underlying theme of all the plug-ins I presented in this article is that though a Web application might seem more limited than a desktop application, in how it looks, feels, and is coded, in fact, there is a way of producing code that can bridge all those gaps. With jQuery, those gaps can be filled through the use of plug-ins. The plug-ins presented here go a long way to making sure your Web application behaves just like a desktop application, which of course is the ultimate goal.

Downloads

Description	Name	Size	Download method
Zip file containing the sample application	examples.zip	33KB	HTTP

[Information about download methods](#)

Resources

- Download the [1.2.6 Minimized jQuery](#), which was the latest stable version at the time of this article, and drop it in your own code.
- Read the complete [jQuery API page](#) to see all of the available functions in the library.
- Check out all the [available plug-ins](#) for jQuery.
- The [iPod Menu Selection](#) is one of the coolest jQuery widgets available.
- Get a thorough and complete background on CSS, JavaScript, and any other Web language at [W3Schools](#).
- Read the other articles in my jQuery series, which deal with an introduction to the library.
 - "[Working with jQuery, Part 1: Rich Internet applications with jQuery and Ajax: Bringing desktop applications to the browser](#)" (developerWorks, September 2008)
 - "[Working with jQuery, Part 2: Rich Internet applications with jQuery and Ajax: Building tomorrow's Web applications today](#)" (developerWorks, September 2008)
 - "[Working with jQuery, Part 3: Rich Internet applications with jQuery and Ajax: JQuery: Building tomorrow's Web apps today](#)" (developerWorks, October 2008)
- "[Ajax overhaul, Part 2: Retrofit existing sites with jQuery, Ajax, tooltips, and lightboxes](#)" (developerWorks, May 2008) takes a look at some of the other plug-ins available for jQuery.

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.

Trademarks

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.