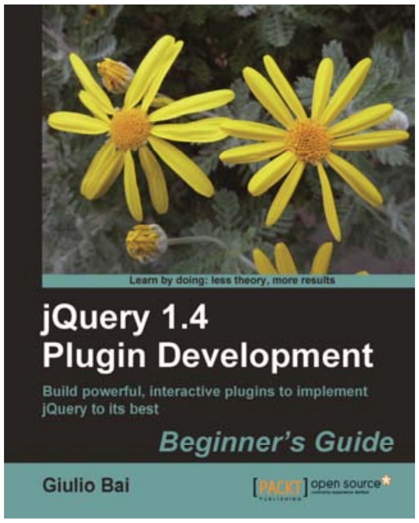
jQuery 1.4 Plugin Development

Beginner's Guide

Giulio Bai


Learn by doing: less theory, more results
jQuery 1.4
Plugin Development
Build powerful, interactive plugins to implement jQuery to its best
Beginner's Guide
Giulio Bai
PACKT open source

Chapter No.9

"User Interface Plugins: Tooltip Plugins"

In this package, you will find:

A Biography of the author of the book

A preview chapter from the book, Chapter NO.9 "User Interface Plugins: Tooltip Plugins"

A synopsis of the book's content

Information on where to buy this book

About the Author

Giulio Bai is a law student living in Modena, Italy who spends most of his time toying with stuff that doesn't have anything to do with law.

Even after trying to keep the list of his past achievements as short as possible, the number of projects he joined in (and invariably sunk short thereafter) makes it hard to narrow down his interests to programming and carousels alone.

It should be made clear that any claim of responsibility for those unfortunate ventures is wholeheartedly rejected—they never had the necessary potential to make it anyway.

I can't brag about this book with anybody if no credit for the beautiful JavaScript library jQuery is given to its author, John Resig.

Also, a bunch of thanks are randomly distributed to everybody I had any kind of contact with, in both real and virtual life, who have—no doubt—somehow helped me in writing this precious manuscript.

jQuery 1.4 Plugin Development
Beginner's Guide

jQuery is the most famous JavaScript library. If you use jQuery a lot, it may be a good idea to start packaging your code into plugins. A jQuery plugin is simply a way to put your code into a package, which makes it easier to maintain your code and use it across different projects. Although basic scripting is relatively straightforward, writing plugins can leave people scratching their heads.

With this exhaustive guide in hand, you can start building your own plugins in a matter of minutes! This book takes you beyond the basics of jQuery and enables you to take full advantage of jQuery's powerful plugin architecture to deliver highly interactive content o your website viewers.

This book contains all the information you need to successfully author your very own jQuery plugin with a particular focus on the practical aspect of design and development.

This book will also cover some details of real-life plugins and explain their functioning o gain a better understanding of the overall concept of plugin development and jQuery plugin architecture.

Different topics regarding plugin development are discussed, and you will learn how to develop many types of add-ons, ranging from media plugins (such as slideshows, video and audio controls, and so on) to various utilities (image pre-loading, handling cookies). You will also learn the use and applications of jQuery effects and animations (sliding, fading, and combined animations) to eventually demonstrate how all of these plugins can be merged and give birth to a new, more complex, and multipurpose script that comes in handy in a lot of situations.

What This Book Covers

Chapter 1, What is jQuery About?, covers what jQuery is and why we should use and prefer it over other libraries. Some basic concepts, as well as some history, are covered in this chapter that acts as an introduction to the real topic of the book.

Chapter 2, Plugins Basics, is our first real approach to jQuery plugins. It provides an in-depth description of jQuery's own plugin architecture, providing some examples and sample applications for some of the most popular plugins.

Chapter 3, Our First jQuery Plugin, as its name suggests, is about creating our first, working, and fantastic jQuery plugin! Step-by-step instructions are provided in order to guide even very beginners to the successful realizati on of their first plugin.

Chapter 4, Media Plugins: Images Plugins, discusses how images play a big role in today's Internet. Since we don't want to be left out, nor behind, in this chapter, we do our best to create a jQuery plugin that is very easy to use, customize, and at the same time, very effective and good looking. Besides, a gallery-like plugin will certainly enhance the user experience of our web pages!

Chapter 5, Media Plugins: Audio Plugins, shows us how, after images, sounds too can be used in a variety of different ways to hold the visitor's attention. Not only will we learn how to develop a jQuery-based audio player plugin, but we will also analyze the advantages and disadvantages of the HTML5 audio tag, compared to JavaScript solutions.

Chapter 6, Media Plugins: Video Plugins, presents a detailed guide to the creation of a video player plugin, and also offers some hints on how to better display video objects on a web page with the aid of JavaScript and/or HTML code.

Chapter 7, Form Plugins, shows a handful of different, but all extremely useful, plugins we can develop in order to improve our forms and offer an enhanced user experience on our website. A number of jQuery plugins are coded, step-by-step, and discussed to better understand what to use, how to use it, and in what circumstances.

Chapter 8, User Interface Plugins, offers many plugin examples and explains how the developer should tackle the problem, in such a way that the final result can be easily modified and integrated into an organized project.

Chapter 9, User Interface Plugins: Toolti p Plugins, explains that to get a fully working toolti p plugin, a series of preliminary steps is required. These include understanding mouse movement and events, positioning through CSS rules, and, last but not least, interaction with jQuery code to actually show and hide the toolti p element at our will.

Chapter 10, User Interface Plugins: Menu and Navigation Plugins, discusses how developing menu and navigation plugins with some additional effects to enhance their appearance and user experience is rather simple. The principles are explained in this chapter, as well as a number of different approaches that we might want to use to obtain a menu plugin.

Chapter 11, Animati on Plugins, discusses how fun-to-activate and nice-to-look-at animation plugins play one of the most important roles when it comes to user interaction. Be it a moving image or a bouncing shape, they are always worth the time spent coding them and actually amuse the visitor. We will learn how to make things move, bounce, fade in and away—nothing more, nothing less.

Chapter 12, Utility Plugins, shows how creating utility plugins (which can be easily used thanks to jQuery's own internal structure and which allow for a very effective integration) is a big plus. If we need some kind of function or method to take care of some repetitive task, we could speed up the process with just a few lines of code.

Chapter 13, Top jQuery Plugins, is a selection of the top 10 plugins. It briefly shows how they are customized on a website, their uses, their advantages and disadvantages, as well as provides a basic documentation that readers can easily use and refer to when (and if) they decide to mess with any of the plugins discussed in this chapter.

9
User Interface Plugins: Tooltip Plugins

We might want to start off by saying tooltips are very popular in today's web design. This is probably due to the value that the tooltips add to the overall look of a website and the sensible addition to a nice user experience that they will certainly contribute.

Also, as we have seen for many other plugins, which are not that difficult to create—provided we know exactly what to do and how to do it—there shouldn't be much of an issue with developing the tooltip plugin. It may actually take some time to understand how to position elements based on the mouse cursor position itself and how to move deftly with an increasing number of functions doing different things. However, we're now on the path to becoming experts, and fearless too!

The topics we're going to discuss include:

- Tooltip plugins in general
- Positioning the tooltip
- Merging pieces together
- Custom jQuery selectors

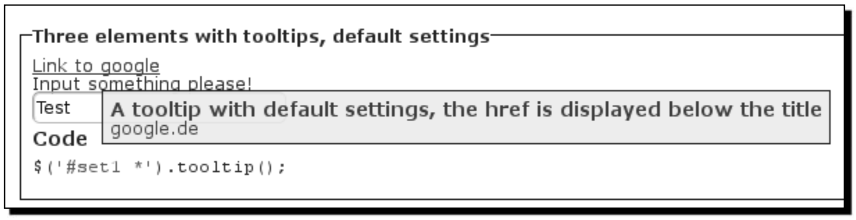
Before we get started, there is another little thing worth mentioning: we have decided to deal with tooltips and menus (in the next chapter) in particular detail. This has been done not only because these two topics are some of the most discussed and certainly stir up some curiosity, but because they also provide many different opportunities (more than other type of plugins, at least!) to introduce new concepts and ideas, even while keeping the complexity of the whole plugin at a minimum.

We can now go on to create our plugin, starting with basic functionalities, and subsequently adjusting its goals. We will add new, improved functionalities that, however, do not make the whole code look too difficult to understand—even after some time or for someone who's just starting out with jQuery.

Tooltip plugins in general

A lot has been said about tooltip plugins, but it's worth repeating the most important points with particular regard to the way tooltips are supposed to work, and how we want our tooltip to behave.

First of all, we might want to get an idea of what tooltips look like and a sample of what we will accomplish by the end of this chapter. Here is an example:


Three elements with tooltips, default settings

Link to google

Input something please!

Test

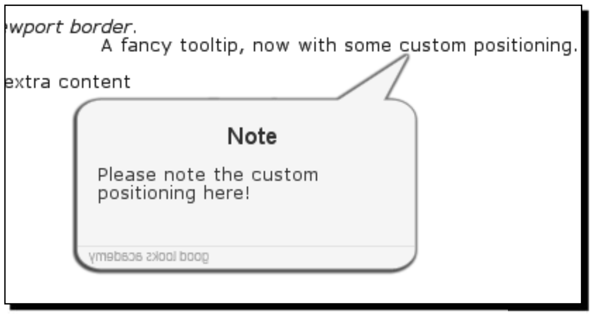
A tooltip with default settings, the href is displayed below the title

google.de

Code

```
$('#set1 *').tooltip();
```

Also, with some more work and proper application of effects, images, and other relatively advanced techniques, we can also obtain something more complex and nicer looking, thus giving the user the chance to specify the style and behavior for the tooltip, as follows:


wport border.
A fancy tooltip, now with some custom positioning.
extra content
Note
Please note the custom positioning here!

The idea is actually very simple. The elements we have selected will trigger an event every time we hover the mouse pointer over them.

The tooltip will then pop out, right at the mouse cursor position, retrieving the text portion from the `title` attribute of the said element.

Finally, whenever we move the mouse over the same element, the plugin will move and follow the mouse cursor until it goes off the boundaries of the element.

Positioning the tooltip

The first problem we have to face is, of course, how to make the tooltip appear in the right position.

It would be no trouble at all if we just had to make some text, image, or anything else show up. We've done it many times and it's no problem at all—just make their positioning absolute and set the right top and side distances.

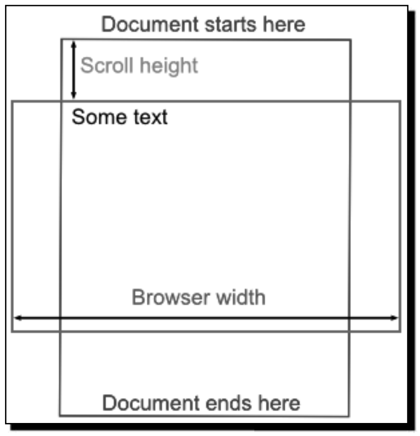
However, we need to take into account the fact that we don't know exactly where the mouse cursor might be and, as such, we need to calculate distances based upon the mouse cursor position itself.

So, how can we do it? It's simple enough; we can use some of the JavaScript event properties to obtain the position. Unfortunately, Internet Explorer always tries to put a spoke in our wheel.

In fact, the magnificent browser does not (according to this table, which is quite accurate: `http://www.quirksmode.org/dom/w3c_cssom.html#mousepos`) support *pageX* and *pageY*, which would normally return the mouse coordinates relative to the document.

So we need to think about a workaround for Internet Explorer, as jQuery (from version 1.0.4 onwards) does not normalize some of the event properties according to W3C standards (`http://api.jquery.com/category/events/event-object/`).

The following diagram (also provided in the code bundle) should clarify what the visible viewport is (that is, the browser window—the red box):



Whenever we scroll down, different parts of the document (blue) are shown through the browser window and hidden due to space constraints. The scroll height (green) is the part of the document currently not displayed.

Custom jQuery selectors

Suppose we have a page with some text written in, which also contains a few links to both internal pages (that is, pages on the same server) and external websites.

We are presented with different choices in terms of which elements to apply the tooltip to (referring to links as an example, but they apply to any kind of element as well), as follows:

- All the links
- All the links with a specific class (for example, `tooltip`)
- All the links with the title attribute not empty
- All the links pointing to internal pages
- All the links pointing to external websites
- Combinations of the above

We can easily combine the first three conditions with the others (and with themselves) using CSS selectors appropriately. For example:

- `$("a")`, all the links
- `$("a.tooltip")`, links having a `tooltip` class
- `$("a[title]")`, links with a `title` attribute (still have to check if empty)
- `$("a.tooltip[title]")`, links with a `tooltip` class and a `title` attribute

As for internal and external pages, we have to work with jQuery selectors instead.

Time for action – creating custom jQuery selectors

Although jQuery makes it easy to select elements using standard CSS selectors, as well as some other selectors, jQuery's own selectors are the ones that help the developer to write and read code. Examples of custom selectors are `:odd`, `:animated`, and so on.

jQuery also lets you create your own selectors!

1. The syntax is as follows:

```
// definition
$.expr[':'].customselector = function(object, index, properties,
list) {
  // code goes here
};
// call
$("a:customselector")
```

2. The parameters are all optional except for the first one (of course!), which is required to perform some basic stuff on the selected object:
   - `object`: Reference to current HTML DOM element (not jQuery, beware!)
   - `index`: Zero-based loop index within array
   - `properties`: Array of metadata about the selector (the 4th argument contains the string passed to the jQuery selector)
   - `list`: Array of DOM elements to loop through

3. The return value can be either:
   - `true`: Include current element
   - `false`: Exclude current element

4. Our selector (for external links detection) will then look, very simply, like the following code:

```
$.expr[':'].external = function(object) {
  if(object.hostname) // is defined
    return(object.hostname != location.hostname);
  else return false;
};
```

5. Also note that, to access the jQuery object, we have to use the following (since object refers to the DOM element only!):

```
$.expr[':'].sample = function(object) {
  alert('$(obj).attr(): ' + $(object).attr("href") + 'obj.href: '
+ object.href);
};
```

Merging pieces together

We have slowly created different parts of the plugin, which we need to merge in order to create a working piece of code that actually makes tooltips visible.

So far we have understood how positioning works and how we can easily place an element in a determined position.

Also, we have found out we can create our own jQuery selectors, and have developed a simple yet useful custom selector with which we are able to select links pointing to either internal or external pages. It needs to be placed at the top of the code, inside the closure, as we will make use of the dollar symbol ($) and it may conflict with other software.

Time for action – creating a tooltip plugin

We still need to think of a way to actually make the tooltip show up and disappear, which is no big deal after all. Let's see how.

1. You should have created the directory containing the right files by now; but anyway, call the plugin file `jquery.tooltip.js` and let's move on.

2. After copying and pasting the code from the various files we used before, we should end up with something like this:

```
(function($) {
  $.expr[':'].external = function(object) {
    return(object.hostname != location.hostname);
  };

  $.fn.tooltip = function() {
    return this.each(function() {
      $(this).hover(function(event) {
        // mouse hover
      }, function() {
        // mouse leaves
      }).mousemove(function(event) {
        // mouse moves
      });
    });
  };
})(jQuery)
```

3. We have to take care of a few things now:

- Check if the element is actually a link (who knows!)
- Check if the title attribute is not empty
- Make sure that the default tooltip is not displayed
- Apply CSS styling to the tooltip

4. We can then add the following code at the very start of the each loop. The event binding will, therefore, be conditioned to the fact that the selected element is an element with a `title` attribute.

Also, we will remove the `title` content so that the default tooltip will not show up (note we first save the title content so that we can use its text later on!).

```
var e = $(this);
var title = e.attr("title");

if(title != '') {
  this.title = '';

  // mouse hover, move and out events
}
```

5. Once the mouse pointer hovers, we need to create the `tooltip` element, insert the text, hide it in order to make it fade, and then apply the necessary CSS to it (update its position).

```
$('<div id="tooltip" />').appendTo("body")
                   .text(title)
                   .hide()
                   .updatePosition(event)
                   .fadeIn(400);
```

6. Eventually, we might want to create another short function to take care of the update position stuff—since we have to repeat it a couple of times and we have much more control in terms of what we need it to do with less code.

```
$.fn.updatePosition = function(event) {
  return this.each(function() {

    $(this).css({
      // we modify the distances by some pixels to make sure
      // the tooltip is visible and put in the best place.
      left: event.pageX+20,
      top:  event.pageY-20
    });
  });
};
```

7. As for the events related to the mouse moving out, around, and on the element, we have the following two lines to do the job very smoothly:

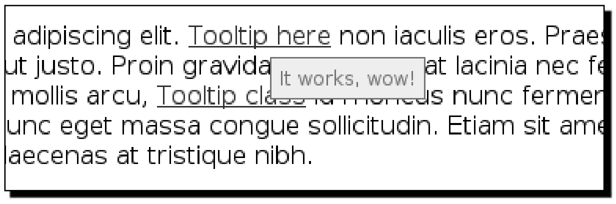
```
// mouse leaves
$("#tooltip").remove();

// mouse moves
$("#tooltip").updatePosition(event);
```

8. Last, but not least, we need to style the tooltip box to make it a little better looking and provide absolute positioning.

   Include the following code into a CSS file named after the plugin (that is, `jquery.tooltip.css`) and make sure that you link it to the page.

```
#tooltip {
  background: #fbf7aa;
  color: #a2844a;
  position: absolute;
  max-width: 150px;
  padding: 5px;
  border: 1px solid #a2844a;
  font-size: 12px;
}
```

9. Our tooltip is finally ready for using, and calling it as we described earlier certainly will work wonders!


adipiscing elit. Tooltip here non iaculis eros. Praes
ut justo. Proin gravida It works, wow! at lacinia nec fe
mollis arcu, Tooltip cla is nunc fermen
unc eget massa congue sollicitudin. Etiam sit ame
aecenas at tristique nibh.

10. And here is the complete code for our plugin. It has been broken down into smaller pieces and functions to make for better looking and more understandable code:

```
(function($) {
  $.expr[':'].external = function(object) {
    return(object.hostname != location.hostname);
  };

  $.fn.updatePosition = function(event) {
    return this.each(function() {
      var xy = getPageXY(event);

      $(this).css({
        left: event.pageX+20,
        top:  event.pageY-20
      });
    });
  };

  $.fn.tooltip = function() {
    return this.each(function() {
      var e = $(this);
      var title = e.attr("title");

      if(e.is("a") && title != '') {
         e.removeAttr('title')
    .hover(function(event) {
          // mouse hovers
          $('<div id="tooltip" />').appendTo("body")
                                   .text(title)
                                    .hide()
                                   .updatePosition(event)
                                   .fadeIn(400);
      }, function() {
```

```
                // mouse leaves
                $("#tooltip").remove();
              }).mousemove(function(event) {
                // mouse moves
                $("#tooltip").updatePosition(event);
              });
            }
          });
        };
      })(jQuery)
```

What just happened?

The whole process for getting a (finally) working tooltip plugin should be clear by now. Anyway, here's a quick checklist of what we have done and what needs to be done in order to develop a jQuery plugin similar to ours.

- Disable the default tooltip display for elements that match our selection and have a `title` attribute (setting it to an empty string is enough for the tooltip not to appear, but we have to remember to copy the said string to a variable so we can use the text contained afterwards).
- When the mouse pointer hovers for the first time on a link, a new tooltip division is created. The tooltip box will be appended to the body and will be given absolute positioning. So we can set its distance from the top and left edges and make it appear close to the mouse cursor.
- As the mouse pointer moves on the link, the tooltip should follow. We thus need to update its `top` and `left` values according to the mouse cursor location.
- Eventually, the mouse will move away from the link and the tooltip should disappear. The whole process will repeat over and over for each link in the page.

Also, a couple more points to discuss are directly related to this script's cross-browser support and further customization of the tooltip.

As for the first, even though we have actually never taken browsers into account until now, we should, of course, do whatever possible to get our plugin working on many different browsers.

This is to our advantage, after all. If the plugin isn't working smoothly in any situation, our web page will not look good, behave right, or do what we thought it would.

Step by step, plugin after plugin, we are slowly adding more and more concepts and code into our plugins. This makes for a more advanced approach to plugin development, taking care of even little imperfections and flaws that our software might have on different browsers.

As for what customization is concerned, not much can be said, actually. The tooltip box is contained in a DIV element and, as such, all the normal CSS rules apply.

With little effort, we can thus obtain a more complex and appealing interface for our plugin—provided we have some skills at CSS writing. Images can be added too, to give the tooltip some shadow or other effects that CSS-only solutions will not allow.

Have a go hero – add options

Finally, the user should be given the opportunity to customize the tooltip plugin according to their needs and liking. So an option-less structure (as it is now) isn't actually the best choice in terms of usability.

It's true that users can include their own CSS file and overwrite every single instruction. However, some tasks, such as making rounded corners, changing the background, and selecting the font color, are more suitable for immediate application directly operating on the code (and thus the function call) rather than going all the way for a separate stylesheet.

Add options to give the user the possibility to change the above-mentioned parameters in a quick and easy way. You should also parameterize the offsets from the cursor and can add, along with rounded corners, the animation used when the tooltip is shown.

Pop quiz

1. What are the tasks we have to carry out, and in what order, to successfully develop a working tooltip plugin for jQuery?
   A. Retrieve the mouse pointer's position, disable the default tooltip, create the tooltip and show it, and remove the tooltip object.
   B. Retrieve the mouse pointer's position, create the tooltip and show it, update the position as needed, and hide the tooltip object.
   C. Retrieve the mouse pointer's position, disable the default tooltip, create the tooltip and show it, update the position as needed, and remove the tooltip object.
   D. Retrieve the mouse pointer's position, create the tooltip and show it, hide the tooltip object, and update the position as needed.

2. Theoretically, a tooltip should disappear once the mouse cursor pointer moves off the element that we chose as the tooltip target.

   How can we recognize that, in fact, the mouse cursor has been moved away from the element and the tooltip should be removed or deleted?

   A. We can't. That's why we wait until another element, which has been assigned a tooltip, is hovered upon so that the previous tooltip disappears and the new one is shown instead.

   B. Adding another (even anonymous) function as the second argument of the `hover` event tells jQuery to run the contained code once the mouse moves out off the selected element.

   C. Using the `mouseoff` event, which jQuery provides by default, and is triggered whenever the mouse pointer, after having been over the selected element, is moved away from it.

   D. None of the above.

3. Which of the following code portions actually does what it's intended for?

   A. Select external links.

   ```
   $.expr[':'].sample1 = function () {
             return (object.hostname !=!= location.hostname);
         };
   ```

   B. Select external links.

   ```
   $.expr[':'].sample2 = function () {
             return (object.hostname == location.hostname);
         };
   ```

   C. Select external links.

   ```
   $.expr[':'].sample3 = function (object) {
             return (object.hostname == location.hostname);
         };
   ```

   D. Select links with title attribute matching one of the given strings.

   ```
   $.expr[':'].sample4 = function (a, b, c, d) {
             var e = eval ("([" + c[3] + ")]");
             for (var i=0; i<e.length; i++)
              if (a.title == e[i]) return true;
            return false;
         };
   ```

4. What is a custom selector?

   A. In addition to the class and ID selectors, jQuery provides quite a lot of new selectors, but we are also allowed to create our own in case we don't find what we need.

B. Just a way to tidy up our code, with no real application except for those situations where we actually need a function-like approach to a problem that couldn't be solved in any other way due to the particular code structure jQuery is written in.

C. Creating a selector lets us access a set of functions and method that we would otherwise never have had a chance to make use of. These new methods allow a simpler and understandable code-writing style we should look forward to only when writing particularly complex code portions.

D. None of the above.

Summary

By now the structure lying behind a plugin is rather clear and the points that all of the plugins seem to share come out with no problem whatsoever. This leads to a quicker and more confident approach to plugin development.

Also, the development pattern we have followed until now has, of course, maintained a certain linearity and coherency. But we have gradually introduced different, more advanced, techniques and have learned a few new tricks that might come in handy at a later time, too.

During the creation of our tooltip plugin we have faced some problems, for which we have provided some flexible yet direct solutions, thanks to the modularity and object-oriented approach jQuery lets us make use of, as well as the incredible options available to overcome a particular hurdle in many different ways.

Our code is getting cleaner and cleaner as we move along, and different functions do different things.

Specifically, a function is responsible for retrieving the horizontal and vertical coordinates of the mouse cursor, whereas a method has been designed with the only purpose of updating the tooltip position based on the distances from the top and left edges.

Our plugin main code eventually had the sole goal of merging these features into a single piece of code (that is the plugin itself).

With the successful realization of a working tooltip plugin, our first attempt at getting a better understanding of what a user interface plugin actually is, and how they are supposed to work to increase the overall quality and visual appearance of the web page, has happily concluded.

The next chapter will instead deal with a different aspect of the user interface: menus. Often it is overlooked and not given the right amount of importance. The fundamental groupings of links that link all of the web pages together are indeed a hot target for many user interface tweaks. We can easily experiment on these with the invaluable help of the jQuery library.