jQuery Puts the Fun Back into Browser Scripting

Rick Strahl

Ok, I admit it. For many, many years I hated JavaScript. I hated writing JavaScript code, and even more I hated the pain that goes along with dealing with different browsers using reams of script code. I got into ASP.NET and stuck to server code. After all, that's what ASP.NET makes really easy. Even when ASP.NET AJAX came along, it didn't really facilitate my work doing client-side coding.

Today I still hate the same JavaScript problems as they haven't gone away, but thanks to a recently gained better understanding of JavaScript and a small JavaScript client library called jQuery, I no longer dread the days when I have to write client-centric AJAX or UI script code.

Looking back, it was not until the AJAX wave started forming a few years back that I even considered getting more seriously interested in JavaScript. It took me some time after that to slowly get up to speed on modern JavaScript principles I had never even thought about up until then; effective use of closures, the Prototype model, class definitions, using functions as code, all of which require some getting used to when you are primarily dealing with static languages like C# and ASP.NET as I do for most of my work.

But even knowing JavaScript reasonably well is still not enough-these days it'd be silly to work with raw JavaScript for DOM programming and not use some sort of library to help bridge the browser-specific quirks and provide utility functionality to make it easier to work in a browser-agnostic environment. When I started my JavaScript quest, I actually built my own library from scratch because there were few libraries out there and they were sparse in features as it was. But today, many highly functional and solid libraries exist so you won't have to go that route and you have a number of quality choices available.

The JavaScript library that really grabbed me is jQuery. jQuery is a relatively small library that is based on a few very simple and intuitive principles. To me, this library strikes the right balance between size, feature set, and ease of use. And after using this library for the last year or so it feels like "this is how JavaScript and DOM programming should be!" and I can hardly imagine doing client-side coding without it any more.

jQuery can bring tremendous productivity gains and it's easy to learn and work with. It's one of those tools that has drastically changed how I think about the client side of Web development in combination with ASP.NET and frankly, it has helped me improve my skill set and understanding of HTML, CSS, and DOM programming significantly. It's also made me tremendously more productive and-more importantly-confident in being able to tackle complex UI and front-end logic in JavaScript reliably.

Key Features of jQuery

jQuery is an open source and therefore free library. You can grab it from: www.jQuery.com

The site contains downloads of jQuery in various compressed and uncompressed formats as well as excellent documentation to get you started with examples and syntax for each of the API functions available.

There's quite a bit of functionality provided in jQuery and the following highlights some of the key features:

- **DOM Element Selectors:** Selectors allow selection of DOM

elements using CSS selector syntax (or an extended version thereof). Using selectors makes it very easy to select single or multiple elements by things like Id, CSS class, relationship to other elements, inheritance, etc., or combinations thereof. You can also pass in a DOM object or an existing jQuery instance to provide a selector. While it may look strange at first to use CSS expressions via code and string expressions, if you're familiar with CSS 2.1 and later and CSS selector syntax, you'll be right at home with jQuery's selectors which use the same syntax plus a few useful extensions to even further simplify selection of elements. The power of selectors lies in easily getting at specific elements in the document, whether it's the selection of a single element or a group of elements. A simple example: It's trivial to select all 2nd column TD elements in a table with a simple selector like this: $("#gdEntries td:nth-child(2)").

- **The jQuery Object-the Wrapped Set:** Selectors result in a jQuery object or what is known as a *wrapped set*. The wrapped set is an array-like structure that contains each of the selected DOM elements. You can access the selected elements directly like an array (i.e., $(selector)[0] gives you the first DOM element), or use jQuery operational methods against all the elements in the wrapped set. You can also use the .each() function on the wrapped set to walk through each item and fire a function in response to each item in the iteration.
- **Wrapped Set Operations:** Iteration is one way to deal with the wrapped set, but the real power of jQuery kicks in via the operations you can perform against the selected elements. Intuitive methods on the wrapped set allow you to do things like addClass()/removeClass() on elements, set or get css() styles directly including smart logic that accounts for browser differences of assignment types (number and string translations mostly) and values (for example, opacity does the right thing on all browsers). You can set and retrieve attributes with attr(), or retrieve or set a value with val(), text(), or html(). You can clone selected DOM elements or create new elements on the fly just by providing an HTML string and then take the selection and inject it into the document with methods like appendTo(), prependTo() or reversely use a parent element to append() or prepend() the new selected element(s). There are basic but useful effects methods that can be applied to show() and hide() elements in a smart way that checks for opacity, display, and visibility and adjusts all to show or hide elements. You can do all of this and much more against all of the selected elements. There's also simple event binding and unbinding that makes it a snap to attach event handlers and reliably unbind them with many of the common event handlers represented by simple functions like click(), mouseover(), keypress() and load(). And that's only a small subset of functionality available. jQuery has about 100 wrapped set methods available plus a number of static utility functions that are globally applicable.Most wrapped set operations are also chainable-they return the jQuery wrapped set object as a result. This means you can chain together many operational methods in a single command. Effectively this means you can select once and operate many times against the same selected set and even filter or expand the wrapped set with methods like find(), filter(), not() or add(). The beauty of many of these functions is that they do things you actually want to do, and are intuitively overloaded. Methods like val() or text() act both as retrieval and setter methods. Methods that deal with numeric values can take either text or numeric values. CSS assignments to browser-dependent tags are automatically fixed up. Although the number of functions provided by jQuery is relatively small, many of the functions provide overloaded functionality to perform intuitive behaviors. The end result is that you have a relatively small API to learn, but a much broader range of functionality that is available through it.
- **Small Footprint:** jQuery is a fairly compact base library yet it's feature-packed with functions that you'll actually use. In

my relatively short time using jQuery, I've gone through well over 85% of the jQuery functions with my code, which points to how useful the library is. All this functionality provided ends up in a compressed size of just under 20k (96k uncompressed with comments). For that you get selectors, a whole slew of operations that can be performed on the wrapped set, AJAX functionality, a host of utility functions for object/array manipulation, and a number of basic effects. For most other things there are plug-ins that generally tend to be small in size and very specialized to give you a modular design. Size matters (to me at least) and jQuery gets high points for "bang for your buck."

- **Easily Extensible:** I think of jQuery as a language and DOM extension library that provides a core set of highly useful features. It's small and tightly focused on providing functionality in a few key areas and no more) and the jQuery team works diligently to keep the library small. To make it easy to create more complex functionality, jQuery includes a simple plug-in model that makes it extremely easy to hook into the wrapped set and provide your own plug-ins. A key aspect of this easy extensibility is the already large and quickly growing jQuery plug-in community (plugins.jquery.com). Plug-ins exist for just about any task imaginable and while the quality of plug-ins varies, most are highly functional and include good documentation and examples all in the same jQuery style that the core library puts forth. In fact, the jQuery team requires documentation and examples as part of its plug-in listing process. All plug-ins come with source code so you can look at how they work and extend or modify them. There's also a forthcoming official jQuery.ui plug-in library that adds key UI features like draggables, droppables, sortables and a few compound components like datepicker, accordion, slider, tab, and dialog controls for example. If you need some specialty functionality, chances are that a plug-in already exists that provides the functionality you're looking for. And if it doesn't, it's easy enough to start creating the functionality yourself with the help of jQuery or other plug-ins as a baseline.

Why jQuery?

As I mentioned earlier, I' used a number of JavaScript libraries before I decided on jQuery. jQuery grabbed me immediately because I was able to do some really nice looking and practical stuff within just minutes. The effects probably got me first because they were so easy and integrated in this small library. After experimenting just a few minutes more I had written ten lines of jQuery code that would have taken me 50-60 in raw DOM code. Later, as I got better with jQuery and read through the documentation more closely, I halved those ten commands down to five.

jQuery code tends to flow smoothly so even when you reduce your code you don't just end up making it terse and unreadable. jQuery genuinely feels easy to use, to me, and it remains readable even though chained jQuery statements can get lengthy (although you can also break up long statements). The chainability of commands allows a single selector to have many operations applied to it. Most jQuery functions return the jQuery object, which makes it possible to chain operations against the wrapped set together. It's not uncommon to chain long lists of operations to a single selector. For example:

```
$("#gdEntries tbody tr:even")
    .not(":first-child")
    .not(":last-child")
    .addClass("gridalternate")
    .css("border","solid 1px lightgrey");
```

Some samples I've seen have as many as 20 functions applied plus some inline callback handlers, which can get ugly. However, that's really a choice of preference-it's entirely possible to create a jQuery and store it in a variable and then use the variable to apply operations so you can break up lengthy commands.

The jQuery API

jQuery is easy to use because jQuery's syntax is easy to pick up. Once you get the basic concept of selectors and you get a feel for a handful of the wrapped set functions you commonly use, you are well on your way to productivity.

You can best learn about jQuery's API on the jQuery site and read the documentation links: http://docs.jquery.com/Main_Page.

Take a look through the various sections and check out the examples to get a good idea of what's available. If you'd rather see all functionality at a glance for reference or if you're a hot shot and don't read documentation, let me point you to a couple of reference cheat sheets that you can print out and keep handy:

You can find a full color jQuery reference sheet from ColorLab at http://shrinkster.com/y65.

Here is another jQuery cheat sheet in black and white for easier printing: http://shrinkster.com/y68.

The jQuery API is manageable in terms of size. Functionality is grouped into a general set of usage areas such as DOM manipulation, CSS, Attributes and Values, Traversal and Filtering, Event Handling, AJAX, and a few more general utility functions. Functions have short and obvious names so there's no ambiguity-it's a very clean API implementation. Using jQuery in your JavaScript code doesn't feel like a heavy handed bolt-on, but rather like something that should have been there all along in the first place. It's consistent and intuitive and there's practically no learning curve-you can jump right in.

Selectors and Wrapped Set Operations

Element selection is such a common thing in DOM programming and having a clean CSS-based selector syntax to select groups of elements greatly reduces the amount of code to find and update elements on the page.

The jQuery API contains around 100 functions that can be applied against the wrapped set, plus a few static functions that provide what are called utility functions, which include object and array manipulation and most AJAX functionality.

With selectors alone you get a lot of control over complex tasks. If you add operations like the ability to hide() and show() elements easily, consistently and optionally with effects, cloning and removing of objects, and an easy browser-independent way to manipulate the style, class and attribute properties of elements, you all of the sudden find yourself with much more control over the browser with just a few short lines of code. The amount of code you write and even the amount of DOM knowledge that you have to have is reduced significantly.

jQuery normalizes many DOM and browser-specific features for you so that you can, for the most part, bid adieu to browser-specific code paths in your script code. In addition, jQuery normalizes many style tags that have different meanings or behaviors in different browsers. For example, the opacity() function works with a single fractional decimal value regardless of which browser is used even though IE doesn't support the opacity style and instead uses a filter. The text() property appropriately gets and sets values appropriate for each browser and element type. Furthermore, width() and height() work with string or numeric values and are normalized across elements so window and document return the same type of values as DIV tags do. Many functions in jQuery are overloaded so the functions can both retrieve and set values depending on whether you pass a value in or not. You can call text() to retrieve a value from the selected element(s) or use text("Hello World") to set a value on all matched elements. Simple design choices like this that reuse and overload rather than bloat the API with excessive functions keep jQuery compact and very approachable.

Event Handling

Nowhere is browser independence and consistency more useful than with event handling. jQuery provides a host of event handler functions that you can apply against the wrapped set to allow for consistent event handling. Functions like click(), keydown(), keyup(), mousedown(), mouseup(), load(), resize(), etc., map most common DOM events and provide a singular handler implementation that take a callback function as a parameter. jQuery then passes the callback handler a normalized event object that has a common set of event properties that are consistent across browsers. You get source and target object references (if applicable), common keyCode translations, and a common way to abort events or cancel event bubbling. jQuery also provides the *this* pointer consistently in event handlers as the DOM element that originated the event. In addition to mapping most standard DOM events, there are also manual binding methods like bind() and unbind(), one() (which fires a specified event exactly once) and trigger() (which triggers an event on elements) functions that provide manual control over event handling using the same principles. Event handling code is super easy-check out the following snippet which implements hover row highlighting on a table using the hover() function plus a click() handler on all of the table:

```
$("#gdEntries tbody tr")
    .hover( function(e) {
        $(this).addClass("gridhighlight");
    },
    function(e) {
      $(this).removeClass("gridhighlight");
    })
    .click( function(e) {
        window.location =
          "otherpage.aspx?id=" +
                    $(this).attr("id");
    });
```

This code selects all rows in the table and applies the hover highlight by handling the in and out events and adding or removing a CSS class. jQuery's hover() function wraps mouseenter and mouseout, so it takes two event handlers. But hover() is also smart enough only to fire when moving outside of the specified element and not when moving into child containers like the native mouseout event. This is just another example of how jQuery simplifies a common scenario in an easy-to-understand event handler function.

A click() handler is also set up to navigate to a new location based on an ID value set in the grid row.

While all of this may look a little weird if this is your first time with jQuery, these few lines of code provide a lot of functionality that would be much more complex to do with manual DOM code, while still being rather descriptive. Although the code is compact, you can still easily see what this code does.

jQuery as an API for the DOM

In a way, jQuery feels like a great equalizer-a leveling of the browser DOM programming model into a higher level API that provides more uniform access to the DOM in a more consistent way across browsers. This is something akin to what the HTML DOM was always striving to be, but because of browser vendor bickering probably never will achieve natively. A library like jQuery can bring some sanity back into DOM development by providing the abstraction needed to deal with browser differences.

To be clear, jQuery isn't the first library that has provided browser abstraction, selectors with operations on it (tools like Prototype, Dojo, Ext etc., do similar browser abstraction), but I think jQuery has done so in a very clean and approachable way that is easy to understand and work with, which is key for adoption.

And jQuery has become wildly popular because of it. It strikes the right balance between functionality, size, and usability so it never

feels overwhelming and yet provides important and practical functionality. jQuery neither reduces the feature set to a useless subset, nor provides so much functionality that you get lost in the myriad of choices. There's just enough so you can print it all onto a single sheet and understand all of the functionality just at a glance.

In so many ways jQuery feels "just right" and from watching the code base grow (you can watch the SVN repository directly if you like to keep track) I think the jQuery team is very much striving for that very premise of "no more functionality than you need, but no less."

Plug-ins to Provide Everything Else

It's important to understand though that jQuery is a core library-it provides DOM manipulation and AJAX functionality, plus basic effects out of the box, plus a host of useful utility functions. But it's not the do all, end all library for JavaScript that you will ever need. jQuery is focused very tightly to a very specific feature set and while very functional it's very likely that you'll end up resorting to plug-ins for specific UI components or additional functionality.

To address the "not enough functionality" aspect, jQuery includes a very simple plug-in API model that is easy to implement. It provides a *jQuery.fn* object that you can extend with methods, each of which gets passed an instance of the active jQuery wrapped set through the *this* pointer. What this means is that you can write a jQuery plug-in with just a few lines of code and participate in the wrapped set processing and chainability just as native functions do.

As a result, countless plug-ins have sprung up for just about any kind of extensibility features you might need for your applications. Although the quality of plug-ins varies, most come with good online, runnable examples that let you view source and documentation, following the guidelines that the jQuery site has set for its own documentation and plug-in requirements.

In addition the jQuery team has now put out a jQuery UI (ui.jquery.com) library that provides a host of additional useful and very rich display features including draggables, droppables, sortables, resizables, selectables and a host of controls like Datepicker, Accordion, Slider, and Dialog. These features are ridiculously easy to use for the complexity of functionality provided and it's great to see this sort of UI extensibility is now easily available in a combined UI plug-in library. As I write this, the library is in Beta 2 but it's already very solid and I've used a number of components (mostly the drag and drop components and the Sortables plug-in) in my applications for great functionality enhancements.

jQuery, AJAX, and ASP.NET

jQuery is a client library and as such has no direct support for ASP.NET or any other server framework. But you can definitely call back to the server by simply using one of a number of AJAX methods available both high level and low level. The high-level functions allow you to make REST-style callbacks using POST or Querystring parameters, which is very straightforward.

There are a number of options available on how to do this. You can choose an HTML-centric approach and return HTML content by calling back to the same page and rendering either HTML manually through code, rending individual controls on a page, or going out to separate full ASPX "template" pages that render specific content for callbacks.

You'll need to do a little extra work in managing callback routing, but it's really quite trivial to do by looking at query string or POST parameters and checking for a specific parameter. I have several examples on how to do this from sessions I recently presented at DevConnections that you can download from here: http://shrinkster.com/y5x

Alternatively you can take advantage of ASP.NET ASMX, Ajax

PageMethods or WCF 3.5 Web services, which I also cover in the slides and samples. Web services use JSON (JavaScript Object Notation) messaging and in order to use Web services, you must use an additional JSON parsing library to turn client-side values and objects into JSON to pass to ASP.NET. I use JSON2.js from Douglas Crockford, although it needs to be modified to support Microsoft AJAX-style date formatting, which is specific to the Microsoft implementation. Due to lack of a date literal in the WC3 JavaScript specification, Microsoft uses a string-based date format that requires encoding and decoding logic in the JSON encoding and parsing JavaScript code. I've also provided the patched json2.js library in the above samples. With this JSON library in place and a small wrapper function, it becomes very easy to call standard ASP.NET Web services both in ASMX and WCF flavors directly from jQuery without requiring the ASP.NET AJAX client libraries.

You can find out more about using jQuery for WCF callbacks on my blog at this link: http://www.west-wind.com/weblog/posts/324917.aspx.

If you prefer not to use ASP.NET Web services on the server there are also several other server-side frameworks like my own West Wind Ajax Toolkit and JayRock that also provide server-side JSON services that are self-contained and can be called from jQuery.

jQuery and ASP.NET Controls

Since jQuery is a client-side library, most extensibility and usage scenarios involve client-side hookups through JavaScript. This means that you write JavaScript code to take advantage of jQuery functionality. In most cases, this is fully appropriate, but as an ASP.NET developer you may feel the urge to get a more *ASP.NET-like* feel for some jQuery functionality. There's nothing stopping you from integrating with jQuery from your server-side ASP.NET code.

One of the simplest things you can do is to use the ClientScript object to inject JavaScript into the page and so automate some operations against jQuery. For example, it might be quite useful to have any ClientScript.RegisterStartupCode() you generate use the highly useful $(document).ready() function to ensure that the DOM has completed loading by the time your startup code fires. Or, you may want to automate setting certain options on various plug-ins from the server side when the page loads. I have a simple example of this in the downloadable samples, which simply calls a jQuery implemented function from a server method via client script.

But sometimes you may also want to map a more complex control to an ASP.NET server control. For example, some time ago I wrapped Marc Garbanskis ui.Datepicker control into an ASP.NET control. Since I use this control a lot, I get much better reusability and a self-contained control that knows what it needs to render itself and its dependencies including support CSS, images, and even the jQuery library if it hasn't been loaded yet. In addition, the control handles postbacks in all of its varying display modes.

This approach is probably overkill for most plug-ins, but for some more control-oriented plug-ins like the datepicker that require potentially lots of settings, it actually makes good sense to wrap the behavior into a server control, especially if you end up using that control a lot in your code. I use the date picker for just about all of my date entry now so using the server control is quicker and easier than manually making the appropriate plug-in initialization calls via JavaScript manually especially if you have several controls on the page. Most of the interoperability is handled via the ClientScript object (or rather a ClientScriptProxy object that also detects ScriptManager and uses the right one) which injects the JavaScript into the generated HTML page output.

jQuery and IntelliSense in Visual Studio

When you use jQuery with Visual Studio you may be disappointed to find that IntelliSense doesn't work very well for it out of the box with Visual Studio 2008 RTM. However, recently Microsoft released

a Hotfix rollup for Visual Studio 2008 that provides better parsing support for various JavaScript libraries including jQuery. With it you get basic support for jQuery's wrapped set operations, but only on direct access of the jQuery function. Any chaining currently loses the jQuery object and so only the first level shows IntelliSense. With a few quick modifications of adding ASP.NET AJAX-style comments into the jQuery script file you can get additional IntelliSense for key functions-it's a little work but if you pick a few of the functions you use a lot you get IntelliSense on most wrapped set operations, which helps a lot to get up to speed with jQuery. You can find out more about how to get better jQuery IntelliSense with the Hotfix at http://snurl.com/2efko.

Microsoft promises even better support for jQuery in the forthcoming release of Visual Studio 2008 SP1, which is supposed to provide full script and parsing support (a beta of SP1 is available today) so manual fix up of the jQuery script file shouldn't be necessary any more.

jQuery.UI

At this point I've talked purely about the core jQuery library, but you may find that if you want to create slick UIs with rich visual effects, the jQuery core library doesn't quite get you there. However, the jQuery team recently released a new set of plug-ins for jQuery.ui. Although version 1.0 was a horrible release of this library, version 1.5, which is currently still in late beta, looks like it squarely hits the mark to provide common and practical rich user interface features. jQuery.ui combines a number of previous plug-ins into a single library, which has been reworked to provide a consistent API. The library is also managed by the jQuery development team and provides many of the same practical and extremely easy-to-use features for building interesting and very slick UIs with little code.

Most of the jQuery UI features are extenders that are applied against the matched set, so things like draggables, droppables, sortables, resizables and selectables are applied against any elements matched. These core UI components are then further abstracted by a few controls. I've used drag and drop extensively with jQuery and it's easy to implement this rich UI functionality into your own apps. I also use the Sortables plug-in extensively. Sortables lets you sort lists and div tags easily using rich user interface effects that use transparency, and smooth drag and drop effect. It's amazing how much of a usability improvement you can get from what seems to be very small changes in script code.

What's Not to Like?

If all of this sounds like a glowing commercial of a tool, you can be assured that I'm just a happy user who's stumbled onto this library some time ago and fell in love with it the first time I used it. I tend to be very critical of the tools I use, but I have yet to have any serious complaints about jQuery. This library just works for me and it works the way I like to work.

But the library is not "the perfect tool" and doesn't solve every JavaScript and DOM problem for you, but in my time of nearly a year with jQuery I haven't run into a showstopper problem. You may still need a set of a few helper functions to help with non-DOM related functionality. For example, I still use my old JavaScript library for a number of UI and JavaScript utility functionality like date and number formatting, providing windowing support, and a host of other features. Bottom line: You'll probably still have a utility library with helper functions to provide a few custom routines especially when it comes to formatting and manipulating the standard JavaScript types which jQuery doesn't cover.

The biggest concern that you might run into is versioning or possible interference with other libraries. Although jQuery supports a simple mechanism for isolating its own jQuery namespace (either via closure wrapping or using the $.noConflict() function), you may still run into issues with other libraries that also expose the $() function. There's also the issue of versioning of the jQuery library

itself which is revised frequently. If you use the library names as downloaded (which include the version numbers) you'll end up renaming files and references frequently as the library is updated. If you use a fixed name you may run into version issues.

But all of this is minor and even expected. To be honest I'm grasping to find fault with jQuery for balance here because my editor mentioned I should mention downsides. A quick search around the Web seems to confirm my feelings though-it's really difficult to dig up dirt on jQuery. But you can judge for yourself by trying it out and my bet is you'll dig it because it's quick to pick up and highly practical.

Is jQuery for You?

jQuery has been a watershed tool for me. I've been struggling with JavaScript for years and have always dreaded it, but when I started using jQuery things changed for me drastically. Many little tedious things that are painful using plain DOM code magically drop away when using this little library. The variety of plug-ins and tools and even the basic effects or the more sophisticated ones available in plug-in libraries make this tool compelling. It's by no means perfect and you still have to fight the DOM at times with different renderings between browsers, but for the most part jQuery alleviates many of the problems faced in raw DOM development. Heck, jQuery provides what the DOM should provide NATIVELY.

As an ASP.NET developer you may be hesitant to embrace a non-Microsoft tool, especially when Microsoft has ASP.NET AJAX. But consider that Microsoft ASP.NET AJAX offers almost nothing useful to the application-level client developer. You can add jQuery to ASP.NET AJAX and not risk feature overlap. ASP.NET AJAX simply doesn't address the same functional domain that jQuery does and the two libraries can seamlessly interact without stepping on each other.

If you're like me and you don't use ASP.NET AJAX, you get some of the key features like AJAX functionality and the ability to call ASMX or WCF Web services with only a little bit of extra work and file payload as described earlier without losing anything in the process. In return, using jQuery you gain a smaller library size and no dependency on **System.Web.Extensions** or the large Microsoft AJAX Library.

Ultimately whether you decide to go with jQuery or ASP.NET AJAX for callbacks is up to you to decide. If you're already using ASP.NET AJAX in your application it's a no-brainer to use the native Web service and page callback support since it's definitely easiest, but even then jQuery can still help you extensively to do something useful with the data returned from the server. If you're not using ASP.NET AJAX and you are already using jQuery, it makes sense to try and do your AJAX callbacks through this library as well and cut out the requirement for the Microsoft AJAX Library.

jQuery is a win-win solution and it's easy to get started, so if you haven't tried it out before, by all means do yourself a favor and check it out and see how much fun client-script coding can actually be...