THE MAN BEHIND THE MAGIC: WE INTERVIEW JQUERY'S JOHN RESIG

Visual jQuery

The Magazine

Issue 1 ... September 2006

$ TUTORIAL

An Introduction to the World of jQuery—Starting With the jQuery Object Itself

$ JQUERY

Winning on Philosophy: Why jQuery's Approach Works

INSIDE jQuery Plugin Rundown

Visual jQuery Magazine

Meet Dave Cardwell

The creator of jQBrowser and jQMinMax chats with *Visual jQuery Magazine*.

On Page 9

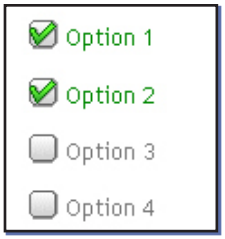
ISSUE 2 SNEAK PEEK

Next month, we have an interview with **Klaus Hartl**, the creator of the popular Tabs plugin, and a developer on Plazes.

Also, a tutorial on jQuery's **AJAX** functionality, a visual effects **plugin roundup**, another article, and **your letters**. All on October 18.

We Want Your Letters

Please submit letters to the editor to editor@visualjquery.com. WE will print letters relating to articles in the Magazine or of general interest to the jQuery community if space permits. We reserve the right to reject any letter.



...From The Editor

As is traditional, I'd like my first editorial to welcome new readers to the **Visual jQuery Magazine**.

Around a year ago, after a fairly large amount of time doing traditional web development, I found myself interested in this newfangled concept called **AJAX**. I attended a workshop by the luminary Thomas Fuchs, the creator of **Scriptaculous** (based on the Prototype library), and while I was quite impressed by the capabilities of both Prototype and Scriptaculous, something tugged at me.

After using them for a bit, and becoming somewhat proficient at using Prototype to build responsive, rich **web applications**, I realized that the learning curve was far too steep. While I was able to do all sorts of powerful things, I found myself constantly reinventing the wheel for simple tasks. While AJAXy Javascript has gotten the term "**DOM Scripting**," I barely felt the influence of the DOM in my day-to-day Prototype development.

Thankfully, I came across **Ruby on Rails** shortly thereafter, which does a darn good job of abstracting out the entire concept of Javascript (and while the now-popular RJS templates were not available yet, I still found Rails' **Prototype helpers** to take the edge off).

A few months later, I came across the **Interface** library in a post about the various cool effects (especially notable was the selectables plugin), and I followed the link home to **jQuery**.

Immediately, I took to the framework, which seemed to think the way I was programming: by centering on **DOM Elements** and tacking bits of functionality on top of them, jQuery made Javascript fun again.

jQuery was far ahead of the pack, even then, with documentation, but I quickly dove in, trying to organize the jQuery **documentation** wiki, and eventually throwing up the first iteration of **Visual jQuery**, the slick visual representation of the jQuery API. When jQuery 1.0 began to include the documentation built-in, I retooled Visual jQuery to take advantage of that.

I suspect many of this magazine's readers took parallel paths to finding jQuery. Still others, initially horrified at the prospect of learning Java, were no doubt pleasantly surprised at just how **easy** it is to learn jQuery. Hopefully this magazine will appeal to both sorts. For the seasoned jQueryist, we bring you advanced techniques, and a **rundown of plugins** that will supercharge your next project.

For the beginner, we'll be featuring **ground-up tutorials** on the framework, helping you to build your first project with a solid grasp of what you're doing, not just copying and pasting code snippets.

With that, I leave you to **the magazine**. Good luck with jQuery!

Illustration By **Jörn Zaefferer**

WINNING ON PHILOSOPHY

Why jQuery's Approach Works

By Yehuda Katz

The approach jQuery takes isn't just "cleaner code" or "chainability." Its fundamental philosophy, centering on collections of DOM Elements, puts it squarely where most Javascript developers program most.

By contrast, other frameworks, like Prototype and Dojo, take a functional approach. Sure, they're fully capable of addressing DOM Elements, just like jQuery, but these other frameworks make entirely different choices.

Prototype, for one, fancies itself a true Object Oriented extension of Javascript's paltry native offerings. In pursuit of true object-orientedness, its developers have put a substantial amount of time into developing object-oriented classes for different types of functionality. A class for forms, a class for elements, a class for events. It goes on and on.

It's perfectly possible to write clean, good-looking Prototype code.

And Prototype can emulate one of the best things about jQuery's focus on DOM Element collections: its chainability. But jQuery conceives of modern Javascript development the way many major players in the Javascript community are starting to see it: as DOM Scripting first and foremost.

For those of us whose Javascript programming focuses primarily on page elements, and I suspect that's most of us, jQuery makes the work dead simple.

jQuery Workflow

Most jQuery methods start with the collection of elements, using the handy support for CSS3, XPath, and a slew of custom expressions (like :visible, which returns only visible elements, and :checked, which returns only checked form fields).

Once you obtain a group of elements, the fun begins. Add .fadeIn("slow") and each of the elements will fade in—slowly.

But we're not done. Without skipping a beat, add .addClass("thisIsDamnFun"). Each element will get the class "thisIsDamnFun." It is, isn't it?

And it can go on from there. Add .click(function(){alert("Hello")}); to throw up an alert box when any of the elements are clicked. Add .append("Hello") and the word hello will be appended to the end of each of the matched elements. Cool, huh?

jQuery Selectors

Now that you've seen the power of jQuery methods, how do we get the element collection in the first place? Happily, we have CSS (1-3) at our disposal, as well as a limited subset of XPath, and some nice custom expressions thrown in for good measure.

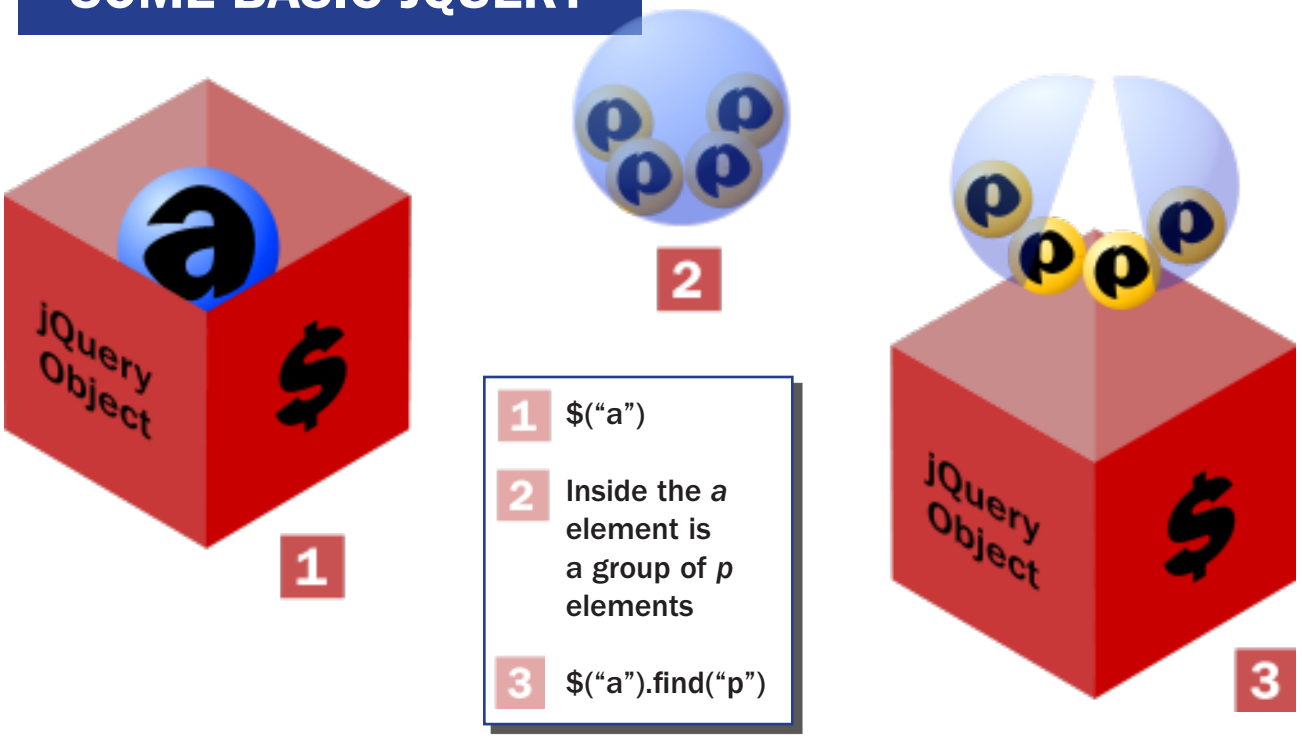
When I say CSS3, I mean it. jQuery supports the ~ selector, :not(expr), attributes via [@attr='whatever'].

XPath support is a bit more limited, but most of the good stuff is here. The / and // operators are available, as are parent and preceding sibling axes. jQuery supports :first, :last, and :eq(n), a slimmed down version of [position() = n].

Finally, jQuery supports testing for contained elements via [tagName]. And because the jQuery parses operates on XHTML, it can be easily co-opted for parsing raw XML from AJAX requests.

And of course, jQuery has the full complement of AJAX methods through $.ajax, and $().load. **Ω**

SOME BASIC JQUERY



THE MAN BEHIND THE MAGIC

JOHN RESIG

This magazine, and jQuery itself, owe their existence to the strong unyielding vision of John Resig. Like many children of the '80s, John grew up as computers grew up.

His first programming language, the ubiquitous QBASIC, underscores John's intellectual curiosity for programming. With the exception of Java, all of his programming skills are self-taught., which makes his upcoming book, *Pro Javascript Techniques*, due to be published this December by APress, even more of an accomplishment.

While he doesn't much care for Java, he believes in watching talented programmers in their element: he keeps tabs on more than 250 web feeds a day. "Seeing an amazing programmer in his 'natural habitat' is always a thing of beauty," Resig said.

Of his many influences, Resig cited Alex Russell of Dojo, and Dean Edwards of IE7 fame as the most consistently interesting. "If there's any Javascript developers that I admire and respect, it's them."

While he has respect for the developers of more established frameworks like Dojo and Prototype, Resig, like David Heinemeier Hansen of Rails fame, has some very strong opinions that drive the framework.

Like Ruby on Rails, jQuery can be seen as a sort of opinionated software, where strong design philosophies give developers a consistent, simple way to approach previously complex problems.

And in addition to a solid design philosophy, Resig doesn't skimp on the community side of things. "I'm frequently discouraged by other project mailing lists, where a simple misdirected question will be answered with anger and malice," Resig said. "For example, having a question come in about Javascript, as opposed to jQuery itself would be easy to dismiss. But by taking the time to answer it, you can win another user."

As part of his drive to ensure the community has plenty of good resources, John focused heavily on documentation in the run-up to the recently released version 1.0 of jQuery.

> 'The major triumph was simply getting the code out the door.'

Before he would be satisfied that the framework was production ready, the code did not need to be bug-free only; everything needed to be documented.

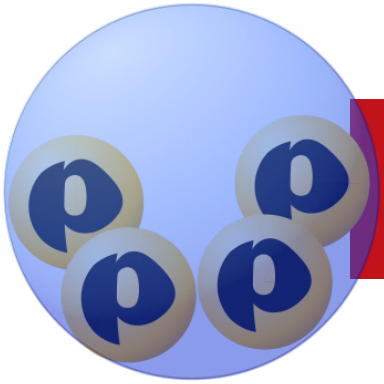
It paid off. As a result of his innovative documentation efforts, version 1.0 of jQuery spawned a dynamically updating Visual jQuery documentation site, which could be updated as often as the codebase itself was updated. In contrast to the old Visual jQuery, which required tedious, manual entry, the new site became a place of first resort for keeping up with the latest jQuery documentation.

Proving that jQuery is more than just a fad, Resig will speak at the Ajax Experience conference next month, a conference organized by the popular Ajaxian.com. In addition to a talk about jQuery itself, he will present a discussion on choosing a Javascript framework.

The run-up to jQuery 1.0 had its share of hassles, but had its rewards as well. "The major triumph was simply getting the code out the door. As any developer will tell you, it's not easy to actually ship," Resig said.

And the work he put into fostering the jQuery community paid off. "There's quite a few developers with [subversion] access who've been helping out, resolving issues as they come to them." While Resig has lately become busy with work and completing his book on Javascript, the community picked up the slack, keeping the post-1.0 momentum going.

That's not to say the release went off without a hitch. "I had to break the API in a couple ways. Many method names were simply not as clear as they should've been and caused many conflicts," Resig said. That said, virtually all users worked out the difficulties, and major plugins, like the Interface visual effects library, released updated versions simultaneously with the official release of 1.0. Ω



JQUERY OBJECT

A BRIEF INTRODUCTION

jQuery, the framework, is fundamentally a way to manipulate collections of DOM Elements. You'll find that's a theme throughout this magazine, but what does a *collection of DOM Elements* really mean? And how does jQuery make it easy to make use of these collections?

DOM Element

A DOM Element is a single **HTML node**, like a *p* or *a*. It can be empty, or it can contain text or other elements inside it.

Think of it this way: every time you open a tag in HTML, you are creating a DOM Element. Everything inside that tag is a **child** of the DOM Element.

An element can be a **parent element**, which means it contains other elements, or a **child element**, which means it has a parent element.

In the example on this page, the *p* is a parent element of both the *span* and *hr*. The *span* is a child of the *p*.

Elements can also be **siblings**. That means they share a parent. The *span* and *hr* in the example on this page are siblings, because they share the *p* as a parent.

Collection of Elements

Traditionally, Javascript programmers have held collections of DOM Elements in standard **Javascript arrays**.

Like any other array, it was possible to check the size of the array, iterate through the array, and get array elements by index. Basic array **functions**.

But there was nothing about an array of DOM Elements that set it apart from any other array.

Say you wanted to append a class to each element in your collection. You'd iterate through each item in the collection with a regular loop, and add the class to each element.

This worked perfectly fine, and some of the more popular frameworks refined the concept to make it easier to do things like add a class.

But underlying all the syntactical sugar, a collection of elements was no different from a collection of strings or a collection of integers.

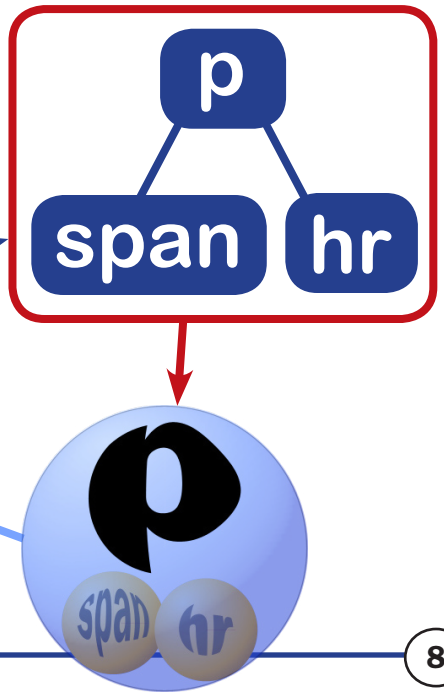
Enter jQuery

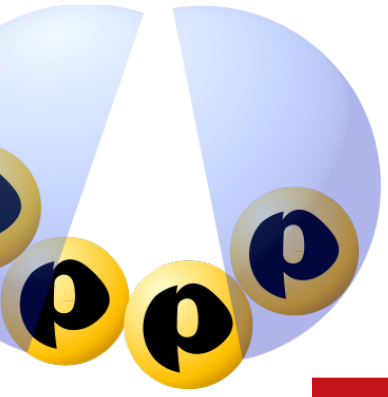
The jQuery framework changes all of that. Instead of seeing a collection of elements as yet another array, jQuery sees the collection as something uniquely DOM-centric.

So in addition to the traditional methods for getting the length of an array and others, collections of elements, held in a **jQuery object**, can do all sorts of interesting things.

First of all, you get elements

```
<p>
  <span>
    Hello
  </span>
  <hr />
</p>
```





‘You can think of a jQuery object like a bucket that holds DOM Elements’

into a jQuery object in a very familiar way: through CSS selectors. For instance, $(“a.fun”) will fill a jQuery object with all *a elements* on the page with the class *fun.*

You can think of a jQuery object like a bucket that holds DOM Elements.

Once we have a jQuery object, we can do any number of things to the elements it holds—all at once.

Remember earlier, when we wanted to add a class to all of the elements in our collection? In jQuery, we can send a message to the object, telling it to add the “hello” class to every element in the bucket.

And the syntax is dead simple: *$(“a.fun”).addClass(“hello”);*

There’s a lot of power packed into that short expression. We’re grabbing all of the elements on the page matching a CSS expression, and adding, to each element matched, the class “hello.”

What’s even more fun is making jQuery chains. We can do $(“a.fun”).addClass(“hello”).hide(). The bucket of elements gets passed to the hide method, which hides them all.

And what makes jQuery special is that all methods that make modifications to a bucket of elements (the jQuery object) return the modified object, so it can be further manipulated.

Event Handling

Another common DOM Scripting need is the ability to bind event handlers to various page elements. For example, you might want to change the class of an element when it’s clicked, to indicate that it’s been selected. Say, for instance, you wanted to add the class *on* to all *p* elements on the page with the class *clickable* when they are clicked. The syntax is classic jQuery:

```
$(“p.clickable”).click(
  function() {
    $(this).addClass(“on”);
  });
```

There are a few Javascript and jQuery idioms here, so let’s go through the code step by step.

First, $(“p.clickable”) gets all *p* elements on the page with the class *clickable* and throws them into a jQuery bucket.

The *.click* means that we are defining an action the browser should take when a click event happens *on an element in the bucket*.

Actions to be taken, or **callbacks**, are defined as anonymous Javascript functions, or Javascript functions without any name. Inside an event handler’s callback, the *this* keyword refers to the specific element that had the event happen to it.

You might wonder why we needed to do $(this).addClass, and not just this.addClass. Remember that you can only run addClass on objects in a jQuery bucket. Inside an event handler, *this* refers only to the element itself. $(this) simply throws the element into a jQuery bucket, which becomes eligible to use the special jQuery methods, like addClass.

Because click is a jQuery method, you can chain additional methods after it. If you remove the semicolon at the end of the command, you can add additional methods on a new line:

```
$(“p.clickable”).click(...)
.append(
  “<span>X</span>”
).fadeTo(“slow”, 0.5);
```

will append a span to the end of the *p* elements with class *clickable* and then slowly fade them out to 50 percent opacity. Ω

KEY POINTS

- Only jQuery objects (buckets of elements) are eligible to use jQuery methods
- Inside event handlers, *this* refers to the element the event happened to
- Write event handlers as anonymous Javascript functions, as: *function() { ... }*

PLUGINS

RICH APPLICATIONS

JQUERY AND PLUGINS

jQuery has a very simple plugin architecture that allows developers to make use of the quasi-magical properties of jQuery functions.

To create a new method that operates on the jQuery object, developers need only create a new function called jQuery.fn.**foo**, which only needs to return a jQuery object itself (to guarantee chainability). To play nice, a jQuery function should iterate through elements automatically, to maintain consistency between the core and plugins.

jTip by Cody Lindley

This page is actually modeled after the jTip plugin (yes, it's that cool). This tooltip solution allows you to set AJAX tooltips with pure XHTML markup—after including the jTip plugin, of course.

Code sample:

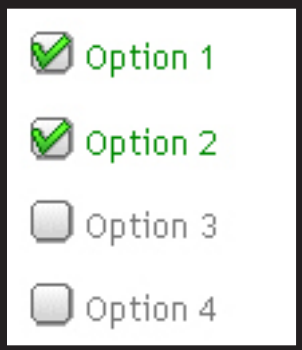
```
<a href="ajax.htm" class="jTip" id="one"
   name="Password must follow these
   rules:">Text
</a>
```

dateSelector by Kelvin Luck

This widget is a sorely needed date selector for jQuery. Notably, it resolved the *select box glitch* in IE, so its calendar control can cover select boxes. dateSelector and the checkbox control prove that jQuery can have a very robust set of widgets. It is also heavily customizable by region.

Code sample:

```
<input type="text" class="date-picker" name="date1"
id="date1" />
```



Checkbox by Kawika K.

The checkbox plugin allows the use of any images instead of the standard checkbox. It allows separate images for off, hovered, and checked. Originally written by editing the raw CSS properties, this new version uses CSS classes for simplicity. The syntax, like the previous plugins, is dead simple.

Code sample:

```
$().cssCheckbox();
```

Meet Dave Cardwell . . .

Dave Cardwell was a programmer well before his release of jQMinMax and jQBrowser brought him fame in the jQuery community.

“I was eight when my parents bought me a green-screen Amstrad computer and it wasn’t long before I was mucking about with BASIC.”

Now, after a year at York University, Cardwell describes himself as ‘a freelance programmer working on his design skills.’

He released his **jQMinMax** plugin in early August 2006.

“At the time there was a lot of buzz about a shift towards liquid and fluid layouts. I wanted to create a plugin that would be of immediate use to people and I saw an opportunity for jQMinMax here.” The plugin simulates max- and min- height and width in Internet Explorer.

As with many diehard jQuery fans, John Resig and his ever-active community were a big part of what drew Cardwell in.

“It has to be the documentation and responsiveness of the community that first endeared me to jQuery. While the syntax and functionality are lovely, there are other libraries available that are comparable in features. It was these peripheral benefits and a real sense of progress in the core library that kept me coming back.”

For Dave, jQuery was a light at the end of the tunnel, where the tunnel was Javascript, and the traffic cops didn’t show up to work.

“I was growing increasingly frustrated trying to track developments in the JavaScript community - new techniques, event handling, document traversal and so on. jQuery brought all these under a single roof for me, at a file size that wouldn’t affect the responsiveness of my sites.”

According to Cardwell, newbies need not be afraid of experimenting with jQuery.

“The **jQuery** and **Visual jQuery** sites are indispensable, and when you’re genuinely stumped I haven’t seen a question to the mailing list that went unanswered.”

Cardwell’s released multiple other plugins since he first began using the jQuery library. More information on those plugins can be found at http://davecardwell.co.uk/geekery/javascript/jquery/.

“I’m always more than happy to hear from like-minded folks - I can be contacted from my website. I can also usually be found lurking in the #jquery IRC channel on freenode.org.”

ABOUT US

Publisher
Wycats Designs

Editor
Yehuda Katz

Contributors
Dave Cardwell
Klaus Hartl
John Resig
Leah Silber
Jörn Zaefferer

CONTRIBUTIONS

Contributions are welcome (and desired). We accept artwork, articles, and interviews. We reserve the right to edit any contribution for clarity and good taste.

Please send contributions to the magazine at editor@visualjquery.com

PLUGINS

If you are interested in having your plugin featured in an issue of the Magazine, please send a link to the plugin, a description, some details about yourself, and, ideally a photo of yourself.

We feature plugins in thematic groupings, so your plugin may appear many issues from now. It may appear grouped with any other plugin we feel is appropriate. The titles, descriptions, and details of plugins are created by the Magazine editors and subject to our discretion.

JAVASCRIPT IS SEXY AGAIN

www.jQuery.com