

jQuery Crash Course

By Nathan Smith

As developers, we have more and more JavaScript libraries to choose from and, of course, the option not to use any at all. Over time, we each tend to favor one method of coding over another. For those who'd like to learn more about jQuery, one of the more popular libraries, here's a crash course written with code-savvy web designers in mind.

Pre-flight

One of the aspects I appreciate the most is that the way elements are selected is similar—often identical—to CSS.

One of the benefits of server-side programming is, for the most part, it all takes place in a stable and predictable environment. Arguments about speed performance aside, coding in PHP, Python, Ruby or .NET is solid because you can count on the platform itself being relatively reliable as you use it. This is a luxury not afforded by default in the world of client-side JavaScript. Browsers have various discrepancies in how they handle the language, not to mention the CSS rendering quirks that can affect the scripts you write. Add to that scenario various legal accessibility implications, alongside the possibility of JavaScript being disabled altogether, and we can have a recipe for disaster on our hands.

In recent years, programmers have risen to the challenge of remedying this problem, and have created JavaScript libraries in an attempt to level the playing field. While this helps remove some of the unpredictability in the way that browsers handle JavaScript, far too often developers either knowingly or ignorantly add hundreds of kilobytes of additional download weight to their pages, often for negligible benefits. Some server-side frameworks even *encourage* the use of obtrusive in-line styles or scripts, creating the propensity for gluttonously bad coding habits.

There are some libraries, such as YUI and Mootools, which allow a buffet-style choice of which aspects of functionality are needed. This helps cut down on unnecessary file-size bloat, as the browser downloads only that which comes into play. You could think of it as a football team allowing all of its red-shirts to suit-up for home games, but leaving them behind on away trips.

Another philosophy is to provide a singular lightweight solution, one which covers basic cross-browser issues and allows the developer to write the essential parts of the functionality by hand.

It is this minimalist school of thought from which jQuery stems. John Resig, the creator of jQuery, began work on the library while a college student at the Rochester Institute of Technology, and now works as a JavaScript evangelist for the Mozilla Corporation, best known for the popular web browser Firefox. jQuery has come a long way since its early beginnings, growing in popularity into a widely used solution for writing elegant code. Some high-profile sites use jQuery, including: BBC, Digg, Intel, MSNBC and Technorati. Even the popular video game company Ensemble Studios uses jQuery for their Age of Empires community website.

I tend to think of jQuery as a universal translator of sorts. You know what I mean—the magic lapel pin that miraculously translates every alien dialect on Star Trek into intelligible English. jQuery is not unlike such a device. It is small, lightweight, and facilitates a common dialog among modern browsers. As long as you can logically formulate what you want to say, jQuery

jQuery Crash Course

By Nathan Smith

understands your intentions and will interpret for you regardless of which user agent is listening in.

Gaining altitude

So, now that we know what jQuery is, let us continue our journey into what the syntax looks like. One of the aspects I appreciate the most is that the way elements are selected is similar—often identical—to CSS. Consider the following examples, the first in CSS, secondly in jQuery, and thirdly in longhand Document Object Model (DOM) syntax. Suppose that I want to get all the links that reside within an element with the id of container.

CSS:

```
#container a { ... }
```

jQuery:

```
$('#container a');
```

DOM:

```
document.getElementById('container').getElementsByTagName('a');
```

Easy enough, but let's take it a step farther. If you are familiar with DOM scripting, you know that retrieving an ID is easy enough, and grabbing a tag from within that element is trivial as well. Given the relative simplicity by which ID and tag names are retrieved in CSS and DOM scripting, one might think that surely a `getElementsByClassName()` method must exist, but unfortunately you would be incorrect. This is a point of much discussion in JavaScript circles, with each developer eventually constructing his or her own way of dealing with classes. (One such individual is Robert Nyman, who has written what he calls the ultimate `getElementsByClassName` script in conjunction with Jonathan Snook.)

In jQuery, this common headache is abstracted away from you as a developer so that you can write your element selectors as simply as you would in CSS. Therefore, if I wanted to make the aforementioned container scenario a bit more specific, and grab all links with the class of popup, all I would have to do is add one more selector and get on with my day. Here again are two examples.

CSS:

```
#container a.popup { ... }
```

jQuery:

```
$('#container a.popup');
```

For me, the ease of using CSS style descendant selectors is worth the price of admission, so to speak. This is where jQuery really shines, making it incredibly friendly to standards-savvy designers who already have a firm grasp of HTML and CSS. Another area in which it excels is in assigning event handlers to various elements. It also allows for chainable events, chainability being one of the key tenets of object oriented programming languages (such as Java). With those fundamentals squared away, it is time to dig into a working demo.

jQuery Crash Course

By Nathan Smith

Flying the Friendly Skies

For this demo, I have created a passenger management interface for charter flights, which will be used by our hypothetical airline corporation, *jQuery Air*. In keeping with jQuery's philosophy of efficiency, the motto of my fictional company shall be, "Traversing the world, with less code." Incidentally, jQuery Air has adopted the savvy business practices of Southwest Airlines, choosing to operate using only one model of aircraft—the Boeing 737. This enables lower costs on parts, maintenance and training than some of the larger airline competitors, as well as requiring only one web interface to be coded by lazy web developers such as myself. (As you work through this article I recommend keeping the demo page open in another window.)

If you take a look at the 'global.js' file in the demo, you will see that it begins with this line of code:

```
$(document).ready(function() {  
    ...  
});
```

All jQuery files contain this snippet, with all subsequent code being residing inside the brackets. This tells the browser that all of the code contained therein is able to be used as soon as the document is ready, without necessarily having to wait for all the accompanying imagery to be fully downloaded. In global.js, we have three main functions with a few presets being called at the beginning of the file.

Before we dive into the JavaScript side of things, let's take a brief look at the HTML structure of our interface:

- The airplane, containing twenty-five rows of seats, is an unordered list () with id="airplane", and each seat is a list-item () inside that list.
- Each seat has a class of seat_XX N, where XX is the row number and N is the position of the seat (A through F).
- To select each row, there is a list () with id="tabs". Each link in this list points to the fragment identifier of its corresponding row (e.g. href="#row_25").
- The passenger information table (id="passenger_details") is made up of a tbody for each row of seats, containing a table row (<tr>) for each individual seat. The tbody and tr elements have ids/classes that correspond to the links used in the airplane and tabs lists.
- Each table row contains input and select elements for entering the passenger details.

Right, back to 'global.js'. The first line makes sure no input or select are usable until after the corresponding seat is clicked.

```
// Disable various aspects of passenger details table  
$('#passenger_details input, #passenger_details select').attr('disabled', 'disabled');
```

Revised October 8, 2007

Page 3 of 8

jQuery Crash Course

By Nathan Smith

This results in input and select being given the attribute/value of disabled="disabled" throughout the table with id="passenger_details". The attr() method is jQuery's way of saying "add an attribute," and the two words contained within the parenthesis are the attribute and its value. In this case, since we are using XHTML, the attribute disabled needs to also have the redundant value of disabled for validity.

The next two lines add class="selected" to the first link in the tabs list, as well as the first tbody in the passenger_details table.

```
// Add class="selected" to tab + tbody  
$('#tabs a:first, #passenger_details tbody:first').addClass('selected');
```

You will notice the :first pseudo-class being used here. While this aspect of CSS3 is not yet widely supported by browsers, jQuery makes it available so that we can start using the more advanced parts of CSS today. This tells the browser to find the first link and first tbody respectively, and creates the initial selected class. For the tab, this makes the background dark blue as a visual indicator. It also causes the first tbody to be visible, since we are hiding all tbody elements on page load.

The first main function involves adding an event-listener to check for clicks on links within the tabs list. When any of these links is clicked:

- The selected class is removed from all tab links
- class="selected" is added to the one that was clicked
- The clicked link's href is assigned to variable thisTarget
- All tbody elements are stripped of the selected class
- class="selected" is added to the tbody that matches thisTarget

At first glance, it might seem strange that this works, but we rely on the fact that fragment identifiers (e.g. href="#row_12") are exactly the right syntax to use in a CSS—or jQuery—rule. The blur() method is applied, as well as return false, to keep the link from being followed and to get rid of the dotted :focus border. This is a fairly commonplace coding convention, and is reused throughout this script.

```
$('#tabs a').click(function () {  
  // Switch class="selected" for tabs  
  $('#tabs a').removeClass('selected');  
  $(this).addClass('selected');  
  
  // Assign value of the link target  
  var thisTarget = $(this).attr('href');  
  
  // Show target tbody and hide others  
  $('#passenger_details tbody').removeClass('selected');  
  $(thisTarget).addClass('selected');  
  this.blur();
```

jQuery Crash Course

By Nathan Smith

```
return false;
});
```

In this function, we have introduced a very special variable in jQuery. By using `$(this)`, we are grabbing a reference to the element being acted upon—in this case, the link that has been clicked.

The next function involves adding an event listener to all links within the unordered list that have `id="airplane"`. Much like in the previous function, the seat's link is assigned to the variable `thisTarget`. You might be wondering, "Wait, won't the previously defined `thisTarget` variable interfere with this one?" Good question. The answer is *no*. We can safely reuse the same terminology because these are local, not global variables. Outside the scope of each function, no other functions are able to access them, thus we can safely re-define them as needed. Again, similar to the previous function, we:

- Add the selected class to the targeted tbody.
- Add `class="selected"` to the seat that was clicked.

This time, we are also going to match against class names for each table row. Since classes can be used multiple times per document, we have assigned the same classes to both:

- The list items inside the `` with `id #airplane`, and
- The `<tr>` in the `#passenger_details` table.

For handling the adding/removal of the selected class for the clicked link, we use the handy jQuery function `toggleClass()`.

In order to match against the class name, we iterate through the listing of links in `#tabs`. If one of those links point to the same ID as the links in `#airplane`, then we add the selected class to the tab link as well. Likewise, the class name of each `<tr>` is checked, and if it matches then `class="selected"` is added. However, if it already has the class selected, then it is removed.

We also grab all input and select tags within the newly selected row, and remove the disabled attribute via `removeAttr()`, so that the elements become interactive rather than disabled.

If the seat is being deactivated though, we:

- Reassign the attribute and its disabled value via `attr()`, and
- Clear its value using `val("")`.

```
// Add click listener to seats
$('#airplane a').click(function () {
  // Assign value of the link target
  var thisTarget = $(this).attr('href');

  // Show target tbody and hide others
  $('#passenger_details tbody').removeClass('selected');
  $(thisTarget).addClass('selected');
```

jQuery Crash Course

By Nathan Smith

```
// Swap out class="selected" for tab
$('#tabs a').removeClass('selected');
$('#tabs a[@href='+thisTarget+']").addClass('selected');

// Assign the value of the parent li class
var thisSeat = $(this).parent('li').attr('class');

// Compare parent <li class=""> against
// <tr> in <table id="passenger_details">
var thisRow = $('#passenger_details tr');
for (var i = 0, j = thisRow.length; i < j; i++) {
  if (thisSeat == thisRow[i].className) {
    // Add class="selected" to row
    $(thisRow[i]).addClass('selected');

    // Enable inputs and selects so that they can be used
    $(thisRow[i]).children('td').children('input, select').removeAttr('disabled');
  }
  else if (thisSeat + ' selected' == thisRow[i].className) {
    // Remove class="selected" from row
    $(thisRow[i]).removeClass('selected');

    // Disable inputs and selects that aren't being used
    $(thisRow[i]).children('td').children('input').attr('disabled', 'disabled').val("");
    $(thisRow[i]).children('td').children('select').each(function () {
      this.disabled = true;
      this.selectedIndex = 0;
    });
  }
}

// Toggle selected class on/off
$(this).toggleClass('selected');
this.blur();
return false;
});
```

You'll notice that I broke out into DOM scripting longhand a bit, using a *for loop* to handle some of the more complex iterations. I did this to match the seat being clicked against its corresponding table row. For instances such as this, plain-vanilla DOM scripting does the job just fine. Additionally, I am zeroing out all drop-downs via `this.selectedIndex = 0`; by assigning a function to each select via `each()`.

The third function pertains to the master checkbox with `id="check_all"`. We add the event-listener, so that when it is clicked, if it is checked, the selected class is added to *all* #airplane seats, #passenger_details table rows, and all input and select are made usable. However, if it is not checked, this goes through does the opposite, deselecting everything and zeroing out all possible values.

jQuery Crash Course

By Nathan Smith

```
// Assign function to master checkbox
$('#check_all').click(function () {
  if (this.checked === true) {
    // Add class="selected" to seat
    $('#airplane a, #passenger_details tbody tr').addClass('selected');
    $('#passenger_details input, #passenger_details select').removeAttr('disabled');
    this.blur();
  } else {
    // Remove class="selected" from seat
    $('#airplane a, #passenger_details tbody tr').removeClass('selected');
    $('#passenger_details input').attr('disabled', 'disabled').val("");
    $('#passenger_details select').each(function () {
      this.disabled = true;
      this.selectedIndex = 0;
    });
    this.blur();
  }
});
```

The very last function in the file is for demo purposes only. Our web interface does not actually do anything, so we would not want the form to submit any data, since it would not be saved anyway. In this case, we add an event-listener to capture the form submission. Instead of sending it off to be processed, we simply display an alert box with the text: “This is only a test.”

By the way, this is the beginning principle of *Hijax*—grabbing form interaction and handling it with JavaScript. Hijax is beyond the scope of this article, but for more on that, I would recommend you read Jeremy Keith’s latest book *Bulletproof AJAX*.

```
// Disable the form submission
$('#form').submit(function()
{
  alert('This is only a test.');
```

```
  return false;
});
```

And that wraps up our crash course on using the jQuery library. You can download a zip of all the files used in this demo to try it on your own site. I hope that you found it informative, and will consider jQuery for any upcoming projects which might require the use of unobtrusive DOM scripting.

Resources

For other helpful info on jQuery, be sure to check out the official documentation, as well as the tutorial blog [Learning jQuery](#).

jQuery Crash Course

By Nathan Smith

The originator of that blog, Karl Swedberg, has recently co-written a book on jQuery with Jonathan Chaffer, one of the lead developers of the Drupal CMS. Their book, aptly named *Learning jQuery* is available via Packt Publishing.

Nathan Smith is a goofy guy who enjoys practicing and preaching web standards. He works as a senior front-end developer at Viewzi. He writes semi-regularly at SonSpring and Godbit. He has been described by family and friends as mildly amusing, but is really quite dull.