CUTTING EDGE

Explore Rich Client Scripting With jQuery, Part 1

Dino Esposito

Contents

You know how it is. The more functionality you want to squeeze out of the Web browser, the more you have to write in JavaScript. Introduced around 1995 to add more action to HTML pages, the JavaScript language was not expressly designed for developers. It was actually designed to manipulate HTML elements, set styles, and react to user input. It has also been widely used for client-side input validation and other lightweight operations.

Seeing the somewhat trivial uses JavaScript has been applied to might lead you to believe that it is simple to write. But as Ray Djajadinata pointed out in "Create Advanced Web Applications with Object-Oriented Techniques" in the May 2007 issue of *MSDN Magazine*, writing good JavaScript code is not trivial but if you understand it well, you can indeed squeeze some rather advanced functionality out of it.

One of the drawbacks of JavaScript, though, is that it's an interpreted (not compiled) language and thus is subject to the browser's, well, interpretation. Not all browsers process the same JavaScript code in the same way. But libraries like jQuery can make your JavaScript code more predictable across browsers.

Microsoft now fully supports jQuery and distributes it with the ASP.NET Model-View Controller (MVC) framework. Furthermore, extensions have been developed to fully integrate jQuery IntelliSense in Visual Studio 2008 SP1.

In this month's installment, I'll discuss the main characteristics of the jQuery library and focus in particular on CSS selectors, function chaining, and wrapped sets. In a future installment, I'll look at more specific topics such as using jQuery to handle events and effects, browser-side caching and AJAX requests.

jQuery at a Glance

jQuery makes JavaScript code easier and quicker to write. The library provides helper functions that dramatically increase your productivity while decreasing your frustration. The resulting code is easier to read and more robust because the higher level of abstraction hides a number of checks and error handling procedures.

The library, written by John Resig, consists of a single .js file you can download from docs.jquery.com/Downloading_jQuery. The latest version is 1.2.6 which was released in the Spring of 2008. The download site offers three versions of the library: uncompressed, packed and minimized.

The uncompressed version is close to 100KB and is fully human readable and commented. This version is definitely the one to pick up for perusal and debugging.

The minimized version is about 50KB. All extra characters that are not strictly required to implement the functionality have been removed from the source code. The code is impractical if not impossible to read for humans, but it works just fine for computers.

Finally, the packed version is barely 30KB in size, but it requires more initialization time on the client. The jQuery official site recommends you consider getting the minimized version instead of the packed version for production environments. In addition, you should consider that GZip compression over the wire is standard practice and is handled by all modern Web servers and browsers. GZip compression brings the size down to about 20 KB. If you use GZip, the packed version is pretty much useless.

In an ASP.NET project, you also need a jquery-1.2.6-vsdoc.js in order to enable IntelliSense and a Visual Studio 2008 patch (see "KB958502-JScript Editor support for '-vsdoc.js' IntelliSense documentation files") in order to fully support jQuery IntelliSense.

In ASP.NET, you can either use a plain <script> tag to link the library or you can list it in the Scripts section of the ScriptManager control, like so:

Copy Code
```
<asp:ScriptManager id="ScriptManager1" runat="server">
    <Scripts>
        <asp:ScriptReference path="Scripts/jquery-1.2.6.min.js" />
    </Scripts>
</asp:ScriptManager>
```
Note that this approach is not recommended with the current version of ASP.NET unless you also want the Microsoft AJAX library embedded in the page. In ASP.NET 4.0, it will be possible to disable the inclusion of the Microsoft AJAX client framework files, thus making this approach a good one.
The whole set of jQuery functionality can be divided into four main areas: DOM query and manipulation, effects and animation, AJAX, and core functions for working with arrays, filtering data, and detecting browser capabilities.

The jQuery Object

The word "query" in the library's name says it all. It refers to running queries over the DOM of the page, which is where jQuery gets its power.
The library supplies a powerful interface for selecting DOM elements that goes far beyond the simple search for elements that match a given ID. For example, you can easily select all elements that share a given CSS class, have certain attributes, appear in a given position in the tree, or have a relationship to other elements. More importantly, you can add filter conditions and you can chain all these query features together, much like you can query data in SQL.
The root of the jQuery library is the jQuery function, defined as follows:
Copy Code
```
var jQuery = window.jQuery = window.$ = function( selector, context )
{
   return new jQuery.fn.init( selector, context );
};
```
The $ function is an alias for the jQuery function. When you create a jQuery object, you pass a selector and a context. The selector indicates the query expression; the context indicates the portion of the DOM on which to run the query. If no context is specified, the jQuery function looks for DOM elements within the entire page DOM.
The jQuery function (as well as its $ alias) performs some work on the provided arguments, runs the query, and then returns a new jQuery object that contains the results. The newly created jQuery object can, in turn, be further queried, or filtered, in a new statement as well as in a chain of statements.
The root jQuery object supports the following signatures:
Copy Code
```
jQuery( expression, [context] )
jQuery( html, [ownerDocument] )
jQuery( elements )
jQuery( callback )
```
The first takes a CSS selector and returns a wrapped array of HTML elements, the so-called wrapped set. The second accepts an HTML string, creates the related subtree, and appends it to the specified owner documents, if any. The third overload picks up the specified DOM element or elements. Finally, the fourth just takes a callback function and runs it on the entire document, as soon as the page's document is fully loaded.
In addition, the root jQuery object also features a few helper methods, as listed in **Figure 1**. Of particular interest to developers is the each method, which you can use as a shorthand for a manual iteration over the content of a jQuery object—typically the DOM elements selected via a CSS selector. Here's a code snippet that shows the each method in action. The loop processes all <input> tags in a form:
Copy Code
```
$("form input").each(
    function(i) {
            this.title = "Input #" + i;
}
);
```

Figure 1 Edge Server roles and reverse proxy for OCS 2007

Methods	Description
each(callback)	Loops over the content of the wrapped set and executes the specified callback function.
length	Property that returns the number of elements in the wrapped set.
eq(position)	Reduces the wrapped set to the single element at the specified position.
get()	Returns the content of the wrapped set as an array of DOM elements.
get(index)	Returns the DOM elements at the specified position in the wrapped set.
index(element)	Returns the 0-based index in the wrapped set of the specified DOM element, if any.

The difference between each () and a manual JavaScript loop is that each() automatically maps the "this" object to the element in the collection being processed. The callback function, however, receives an optional integer parameter that is the (0-based) index of the iteration. Let's learn more about jQuery selectors and their CSS-based syntax.

Here's the simplest use of the $ function:

Copy Code

```
var elem = $("#grid");
```

The $ function in the code snippet retrieves all DOM element(s) whose ID property matches the specified expression. The # symbol doesn't belong to the ID string, but is just a prefix for the $ function to disambiguate ID strings, CSS classes and HTML tag names. (The # symbol is part of standard CSS syntax for ID selection.) The preceding code snippet is functionally equivalent to the following DOM statement:

Copy Code

```
var elem = document.getElementById("grid");
```

It is worth noting that in the HTML DOM, unlike in ASP.NET, multiple elements can share the same ID. If an array of elements match the ID, then method getElementById would only return the first matching element; getElementsByName, on the other hand, would return the whole collection.

The similarity between classic DOM methods and the $ function ends there; the power of $ goes far beyond. Through $, you select DOM elements and then apply a function to each.

The selector expression is driven by the CSS 3.0 syntax and can reach a nontrivial level of complexity. **Figure 2** shows the supported selectors. The list doesn't include filters, which I'll talk about in a moment. It is key to note that in a hierarchy of selectors, the ancestor, the parent, or the previous element can be any valid selector, not just an HTML element. **Figure 3** shows a few sample queries.

Figure 2 Supported jQuery Selectors

Selector	Description
#id	Returns the first element, if any, in the DOM with a matching ID attribute.
element	Returns all elements with a matching tag name.
.class	Returns all elements with a matching CSS class.
*	Returns all elements in the page.
selector1, ..., selectorN	Applies all given basic selectors and returns the combined results.
ancestor descendant	Given an ancestor selector, returns the collection of all descendant elements that match the descendant selector. For example, "div p" returns all <p> elements within a .
parent > child	Given a selector, returns the collection of all child elements that match the child selector.
prev + next	Given a selector, returns the collection of all sibling elements that match the next selector and are located next to the prev selector.
prev ~ sibling	Given a selector, returns the collection of all sibling elements that match the sibling selector and follows the prev selector.

Figure 3 Sample jQuery Selectors in Action

Sample Selector	Effect
form input	Returns all input fields within any <form> tags in the page.
#Form1 input	Returns all input fields within the form labeled Form1.

h2 + p	Returns all <p> tags that are next to a <h2> while child of the same parent.
input.textBox	Returns all <input> tags whose CSS class is "textBox."
div span.myClass	Returns all tags whose CSS class is "myClass" located within a .

Selectors can be further refined by applying filters on attributes, content, position, and visibility. **Figure 4** lists some of the most popular filters in jQuery. The full reference is at docs.jquery.com/Selectors.
Filters such as first and last find DOM elements at a given position in the returned collection. You can also use an index-based syntax to filter elements by using the eq, gt, and lt filters. The filter eq picks up the element whose index matches the given index, whereas gt and lt pick up elements greater than, or less than, a given index.
Attribute filters are powerful tools that select HTML elements where a given attribute is in a given relationship with a value. In **Figure 4**, I only listed the most commonly used attribute filters. Other filters exist to select elements where a given attribute begins with, ends with, or contains a given value. Here's the required syntax:

Copy Code

```
[attribute^=value]  // begins with value
[attribute$=value]  // ends with value
[attribute*=value]  // contains value
```

Figure 4 jQuery Filters

Positional Filters	Description
:first	Returns the first element of the selected collection of elements.
:last	Returns the last element of the selected collection of elements.
:not(selector)	Filters out all elements matching the specified selector.
:even	Returns all even elements in the selected collection.
:odd	Returns all odd elements in the selected collection.
Child filters	Description
:nth-child(expr)	Returns all child elements of any parent that match the given expression. The expression can be an index or a math sequence (for example, 3n+1), including standard sequences such as odd and even.
:first:child	Returns all elements that are the first child of their parent.
:last-child	Returns all elements that are the last child of their parent.
:only-child	Returns all elements that are the only child of their parent.
Content filters	Description
:contains(text)	Returns all elements which contain the specified text.
:empty	Returns all elements with no children. (Text is considered a child node.)
:has(selector)	Returns all elements that contain at least one element that matches the given selector.
:parent	Returns all elements that have at least one child. (Text is considered a child node.)
Visibility filters	Description
:hidden	Returns all elements that are currently hidden from view. Input elements of type "hidden" are added to the list.
:visible	Returns all elements that are currently visible.
Attribute filters	Description
[attribute]	Returns all elements that have the specified attribute.
[attribute = value]	Returns all elements that have the specified attribute set to the specified value.
[attribute != value]	Returns all elements whose specified attribute (if present) has a value different from the given one.

Attribute filters can also be chained by simply placing two or more of them side by side, like so:

Copy Code

```
[align=right][valign=top]
```

A particularly powerful filter is nth-child. It supports a number of different input expressions, as shown here:

Copy Code

```
:nth-child(index)
:nth-child(even)
:nth-child(odd)
:nth-child(sequence)
```

The first format selects the nth child of the HTML elements in the source selector where n refers to the provided index. All elements placed at any odd or even position (0-based) are returned if you specify the odd or even filter. Finally, you can pass nth-child as the root expression of a mathematical sequence such as 3n to indicate multiples of 3.

As an example, the following selector picks up all rows in a table (labeled DataGrid1) that are on the positions determined by the sequence 3n+1, (1, 4, 7 and so forth, remembering that it is a zero-based index):

Copy Code

```
#DataGrid1 tr:nth-child(3n+1)
```

The next one is a much more complex expression, and it demonstrates the incredible power and flexibility of jQuery selectors:

Copy Code

```
#DataGrid1 tr:nth-child(3n+1):has(td[align=right]) td:odd
```

It reads as follows: of all the table rows selected at the previous step (position 1, 4, 7, and so on), now you only take those which have a cell (a <td> tag) where the attribute align equals the value of "right." Furthermore, of the remaining rows, you only take the cells on columns with an odd index. Let's consider the HTML table in **Figure 5**. In **Figure 6**, the cell with a yellow background is the result of the query.

Figure 5 An HTML Table

Copy Code

```
<table id="DataGrid1" border="1">
    <tr>
       <td>Column1</td>
       <td>Column2</td></tr>
    <tr>
       <td>Val1</td>
       <td align="right">Num1</td></tr>
    <tr>
       <td>Val2</td>
       <td align="right">Num2</td></tr>
    <tr>
       <td>Val3</td>
       <td align="right">Num3</td></tr>
    <tr>
       <td>Val4</td>
       <td>Num4</td></tr>
    <tr>
       <td>Val5</td>
       <td>Num5</td></tr>
    <tr>
       <td>Val6</td>
       <td>Num6</td></tr>
    <tr>
       <td>Val7</td>
       <td>Num7</td></tr>
</table>
```

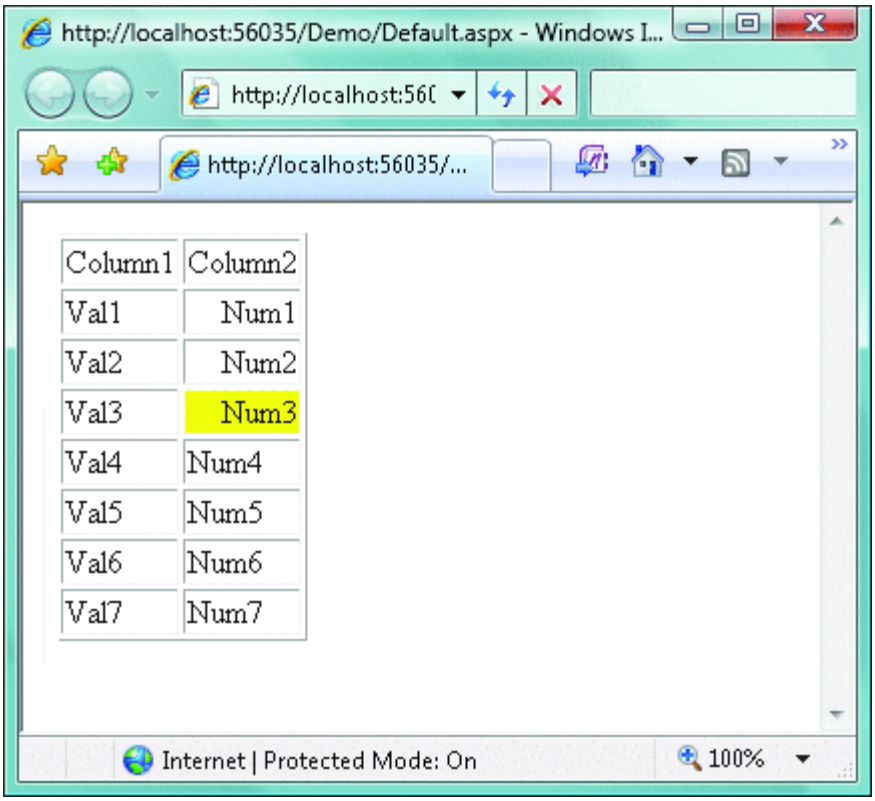

Figure 6 **Selecting a Certain Cell on a Table**

Form Filters

As mentioned, the overall syntax of jQuery selectors is close to the syntax of CSS 3.0 selectors, just extended with some extra pseudo elements such as those listed in **Figure 7**.

The :input filter, in particular, refers to all logical input elements you may find on a page and is not limited to the <input> tags. In fact, it includes <textarea> elements and <select> elements used to display listboxes and dropdown lists. Selectors in **Figure 7** do not match CSS selectors, but provide handy shortcuts to pick up homogeneous elements such as all input tags of a given type. As an example, the selector :text is functionally equivalent to the following:

Copy Code

```
form input[type=text]
```

Figure 7 Form Filters

Selector	Description
:input	Returns all elements that have a role in collecting input data, including textarea and drop-down lists.
:text	Returns all input elements whose type attribute is text.
:password	Returns all input elements whose type attribute is password.
:checkbox	Returns all input elements whose type attribute is checkbox.
:radio	Returns all input elements whose type attribute is radio.
:submit	Returns all input elements whose type attribute is submit.
:reset	Returns all input elements whose type attribute is reset.
:image	Returns all input elements whose type attribute is image.
:button	Returns all input elements whose type attribute is button.
:file	Returns all input elements whose type attribute is file.
:hidden	Returns all input elements whose type attribute is hidden.
:enabled	Returns all input elements that are currently enabled.

:disabled Returns all input elements that are currently disabled.

:checked Returns all checkbox or radio elements that are currently checked.

:selected Returns all list elements that are currently selected.

Other nice helpers are available to grab all input elements in a page that are enabled or disabled and all checkboxes and radio buttons checked as well as list items currently selected.

Operations on Wrapped Sets

HTML elements that match a selector are returned packed in a new instance of the jQuery object. This object incorporates a JavaScript array with all DOM references. The results are often referred to as a wrapped set. A wrapped set is never null, even if no matching elements have been found. You check this situation by looking at the length property of the jQuery object, as shown here:

Copy Code

```
// All TDs with a child IMG
var w_set = new jQuery("#DataGrid1 td:has(img))");
if (w_set.length == 0)
    alert("No match found.");
else
    alert(w_set.length)
```

The high level of flexibility shown by jQuery is a great feature to have, but then once you have all the elements you were looking for, you need to process them. The simplest possible approach entails setting up a loop and running a function over each item in the collection, like this:

Copy Code

```
var ws = $("#DataGrid1 tr:nth-child(3n+1)");
for(i = 0; i<ws.length; i++)
{
   processElement(ws[i]);
}
function processElement(elem)
{
   ...
}
```

In such a manual iteration, you access DOM elements directly, just as in classic JavaScript programming. The jQuery library offers a couple of alternate routes that are functionally equivalent to manual iteration. Nicely enough, jQuery iterations result in more compact and even more readable code. The first approach is based on the each function.

As mentioned, in jQuery the each function executes a user-defined callback on any element associated with the wrapped set. A fairly large number of operational methods, however, exist to make it even quicker and easier for you to execute common operations on the wrapped set. For example, you can use the css function to apply CSS styles to the wrapped set. Following is an example that sets background color and border style of all input elements in a form:

Copy Code

```
$("form input").css(
   {'background-color' : 'yellow',
    'border-style' : 'dashed'}
);
```

Admittedly, this short example is a bit misleading as it may suggest that jQuery should be used for things that CSS itself does well. The css function is great when it works on dynamically applied styles that depend on user interaction or some other run time condition.

Likewise, you can add, remove and even toggle a CSS class on the elements in the wrapped set. You do this via the addClass, removeClass, and toggleClass functions. The attr function allows you set one or multiple attributes on all elements. For example, here's how to disable all input elements:

Copy Code

```
$("form input").attr(
   {'disabled' : 'disabled'}
);
```

The html function sets the HTML content of a tag. It uses the innerHTML property internally. To set the inner text of a tag, instead, you use the text function passing the text to set as the argument. This is a good way to understand the benefits of a cross-browser library like jQuery. The property innerHTML can be considered a de-facto standard property supported by all browsers in the same way.
The same can't be said for the analogous property that only sets the text. This property is innerText in Internet Explorer and textElement in Firefox. The jQuery text function hides differences and provides the same functionality across all browsers.

jQuery Chainability

One of the best features of jQuery is its chainability, which is possible because the jQuery object itself, as well as most of the functions and filters, return a jQuery object. The returned object contains the original wrapped set as modified by the function itself. For example, if you apply the css function to a wrapped set, the returned jQuery object contains the same set of elements along with a modified set of CSS styles. If you apply a filter, you'll get back a smaller set as filtered by the function. It is worth noting that you also have available a not function to exclude all elements that match the specified query.
Concatenating jQuery functions couldn't be easier. All you need to do is attach a new function call to the jQuery object returned by the previous call. Here's an example:

Copy Code

```
$("form input")
.not("[type!=text]")
.css(
   {'background-color' : 'yellow',
    'border-style' : 'dashed'})
.attr("value", "test");
```

Here the sample expression selects all input fields where the type attribute equals text and sets a few CSS styles and the value attribute to fixed values. (Note that in the example, I used the not function just to add a third link to the chain. The result of the not condition could be integrated into the $ query resulting in even clearer code.)
It goes without saying that method of chaining produces more compact code; but it doesn't necessarily result in faster code. Chaining is a technique that you, as a developer, may or may not feel comfortable with. If you experience troubles with one of these compact expressions, you might want to break it into pieces to simplify debugging.
Finally, note that not all jQuery methods return a modified jQuery object. Methods like html or text, for instance, just return a string that respectively points to the HTML or text content of the first element of the source wrapped set. For the exact syntax of jQuery methods see docs.jquery.com/Main_Page.

Know Your HTML

The more you explore the depths of jQuery, the more you understand the importance of knowing in detail the HTML you're working with. ASP.NET server controls tend to hide the structure of the HTML they output. On the other hand, server controls were just introduced to let developers focus on declarative attributes rather than HTML details. Years of experience proved this was not always the right approach.
Today, you do need to keep your HTML under total control for accessibility, styling, and conformance to XHTML. Incidentally, this fact establishes a subtle link between jQuery and the ASP.NET MVC framework. And it is no coincidence that the ASP.NET MVC framework includes the latest version of jQuery in the package.
That said, it is also worth noting that tying logic to markup using jQuery selectors might create an unwanted coupling between logic and the shape of the DOM, which may result in difficult-to-maintain applications.
Finally, if you're looking for more ideas for using JavaScript in interesting ways, check out the list of JavaScript articles at on MSDN.
Send your questions and comments for Dino to cutting@microsoft.com.

Dino Esposito is an architect at IDesign and the co-author of *Microsoft .NET: Architecting Applications for the Enterprise* (Microsoft Press, 2008). Based in Italy, Dino is a frequent speaker at industry events worldwide. You can join his blog at weblogs.asp.net/despos.