Extending jQuery
By Keith B. Wood

jQuery is the most widely used JavaScript library on the Web today. It offers many functions to make life much easier as a front-end developer. Moreover, the developers of jQuery have recognized that it can’t (and shouldn’t) do everything and has provided various extension points that allow additional functionality to be integrated into the normal jQuery processing. This foresight has contributed to its popularity. This green paper based on Extending jQuery, explains why and how would want to extend jQuery.


You may also be interested in...

Why Extend jQuery?
You can extend various aspects of jQuery to provide greater reuse and easier maintenance of your code. Alongside the standard plugin that operates on a collection of elements on a Web page, you can create custom selectors, utility functions, custom animations, enhanced Ajax processors, custom events, and validation rules.

jQuery is defined on its website as “a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development.”

It is a library of JavaScript functions that allows you to easily access the HTML Document Object Model (DOM) and inspect or update it, enabling you to provide more dynamic web pages and experiences in keeping with the Web 2.0 paradigm. Its main features include:

- Element selection using a CSS-like syntax, with extensions.
- Element traversal.
- Element manipulation, including removal, content updates, and attribute changes.
- Event handling, including custom events.
- Effects and animations.
- Ajax support.
- A framework for extending its functionality.
- Various utility functions.
- Cross-browser support, including hiding differences between the browsers.

jQuery is a freely available, open source library. It is dual-licensed under the MIT License or the GNU General Public License, Version 2.

Origins
jQuery was initially developed by John Resig and was announced in January 2006 at BarCamp NYC. He had come across the Behaviour code written by Ben Nolan and saw the potential of its ideas—using pseudo-CSS style selectors to bind JavaScript functions to various elements in the DOM. However, he was not happy with its
verbosity and lack of hierarchical selectors (http://ejohn.org/blog/selectors-in-javascript/). His suggested syntax and subsequent implementation became the basis for jQuery.

Listing 1 shows an example of Behaviour code for attaching a \texttt{click} event handler that removes the clicked item to all \texttt{li} elements within an element with the ID \texttt{example}:

\begin{verbatim}
Listing 1 Sample Behaviour code

Behaviour.register({
  '#example li': function(e){
    e.onclick = function(){
      this.parentNode.removeChild(this);
    }
  }
});

\end{verbatim}

And now, the familiar corresponding \texttt{jQuery} code:

\begin{verbatim}
And why \texttt{jQuery} as the name? Originally, the library was called \texttt{jSelect} to reflect its ability to select elements within a web page. However, when checking for that name on the web, John found that it was already taken and changed to \texttt{jQuery} instead.

\section*{Growth}

Since its initial announcement, jQuery has been through numerous incremental releases, as shown in table 1. Over that time, it has grown greatly in terms of functionality and size.

\begin{table}[h]
\centering
\caption{jQuery versions}
\begin{tabular}{llll}
\hline
Version & Code date & Size & Notes \\
\hline
1.0 & August 26, 2006 & 44.3K & First stable release \\
1.0.1 & August 31, 2006 & 44.7K & \\
1.0.2 & October 9, 2006 & 48.1K & \\
1.0.3 & October 27, 2006 & 49.9K & \\
1.0.4 & December 12, 2006 & 52.2K & Last 1.0 bug fix \\
1.1 & January 14, 2007 & 55.6K & Selector performance improvements \\
1.1.1 & January 22, 2007 & 56.3K & \\
1.1.2 & February 28, 2007 & 57.9K & \\
1.1.3 & July 1, 2007 & 61.2K & \\
1.1.3.1 & July 5, 2007 & 61.4K & \\
1.1.4 & August 23, 2007 & 65.6K & Any name \text{jQuery} \\
1.2 & September 10, 2007 & 77.4K & \\
1.2.1 & September 16, 2007 & 78.5K & \\
1.2.2 & January 14, 2008 & 93.0K & \\
1.2.3 & February 6, 2008 & 94.4K & \\
1.2.4 & May 18, 2008 & 95.3K & \\
\hline
\end{tabular}
\end{table}

For Source Code, Sample Chapters, the Author Forum and other resources, go to
\url{www.manning.com/wood}
1.2.5  May 20, 2008  97.6K  Fix for bad build of 1.2.4
1.2.6  May 26, 2008  97.8K
1.3  January 13, 2009  114K  Sizzle Selector Engine introduced into core, live events and events overhaul
1.3.1  January 21, 2009  114K
1.3.2  February 19, 2009  117K
1.4  January 13, 2010  154K  Performance improvements, Ajax enhancements
1.4.1  January 25, 2010  156K  height() and width() added, parseJSON() added
1.4.2  February 13, 2010  160K  delegate() added, performance improvements
1.4.3  October 14, 2010  176K  CSS module rewrite, metadata handling
1.4.4  November 11, 2010  178K
1.5  January 31, 2011  207K  Deferred callback management, Ajax module rewrite, traversal performance
1.5.1  February 23, 2011  211K
1.5.2  March 31, 2011  214K
1.6  May 2, 2011  227K  Significant performance improvements to the attr() and val() functions, prop() added
1.6.1  May 12, 2011  229K
1.6.2  June 30, 2011  230K
1.6.3  August 31, 2011  232K
1.6.4  September 12, 2011  232K
1.7  November 3, 2011  243K  New Event APIs: .on() and .off(), event delegation performance
1.7.1  November 21, 2011  242K
1.7.2  March 21, 2012  246K
1.8.0  August 9, 2012  253K  Sizzle re-written, animations re-imagined, more modularity

Although the size of the jQuery library has grown substantially, when the code is minimized (stripping unnecessary comments and whitespace), it is reduced to about a third of its source size. When that minified version is served from the Web in a GZip format it is further reduced to about a third, resulting in a download cost of about 32 K for the latest version. By using one of the Content Delivery Networks (CDNs) available, that file may already be cached on the client, removing the need to download it at all.

jQuery now includes the Sizzle selection engine enabling the fundamental abilities of finding the elements within the DOM that you wish to operate upon. Whenever possible, Sizzle delegates these selectors to the underlying browser implementation but resorts to JavaScript when necessary to ensure a common experience across all the major browsers.

**Today**

jQuery has become the most popular JavaScript library on the Internet and has been adopted by many organizations and individuals for use in their websites. **BuiltWith** reports over 50 percent of the top 10,000 websites
use jQuery and over 35 percent of the top million. W3Techs reports jQuery usage at 45 percent of all websites and 85 percent of those using any JavaScript library.

There is a thriving community of plugin developers, most of whom make their code freely available in the spirit of the underlying jQuery library. Unfortunately, the “official” repository of jQuery plugins has been out of action for some time due to planned upgrades to overcome spam problems. So, you have to resort to searching the Web for appropriate modules. Some plugins are great, with solid code, good documentation, and examples. Others are not so good, being hard to use, buggy, and/or poorly documented. Once you have read this book and apply its principles, your plugins should fall into the former category.

There is also much activity on the jQuery forums with over 220,000 responses to over 100,000 questions. Within the forums there are special sections devoted to using and developing jQuery plugins.

The ongoing development of jQuery is now managed by the jQuery Project. It was formed in September 2009 to look after all the jQuery projects, including jQuery Core, jQuery UI, jQuery Mobile, Sizzle, and QUnit. Contributions and donations by the jQuery community provide the financial basis for this support.

Since jQuery offers so much functionality, why would you want to extend jQuery? To keep the size of the jQuery code manageable, only those functions that are generic and widely used are included in the core code (although there is debate over what is used and/or useful). Basic element accessing and modification, event handling, animation, and Ajax handling are provided as functionality that most users require, while more specialized abilities are left to others to add.

Fortunately, the jQuery team has recognized the fact that core jQuery can’t do everything, so they have provided numerous integration points by which others can extend the functionality of jQuery, while benefitting from its existing infrastructure and abilities.

As well as extending jQuery to provide additional functionality, packaging your extension as a plugin allows you to easily reuse those abilities on other Web pages. Thus, you only have one copy of the code to maintain, and any improvements are immediately applied wherever it is used. You can test your plugin code in isolation and under controlled circumstances to ensure that it works as expected.

What can you extend?

There are many ways to extend jQuery, just as there are many abilities provided by the core library. That’s what we’ll discuss in the following paragraphs.

Selectors and filters

jQuery selectors and filters allow you to identify and collect together the elements from the web page on which you wish to operate. Although standard selectors by node name, ID, and class are built into jQuery, there is scope for adding pseudo-class selectors that allow you to filter a previous selection in a consistent and succinct manner. You can also add set filters that are aware of the entire collection of previously selected elements and each one’s position within that list.

By creating a custom selector, you can consolidate the selection process into one location, making it easier to reuse that code elsewhere, ensuring a consistent implementation across your projects. It is also easier to maintain the selector, with any bug fixes or enhancements being applied to all instances immediately.

Collection plugins

Collection plugins are functions that are applied to collections of elements as retrieved by a selector. These functions are what most people think of when the term “jQuery plugin” is used and make up the largest portion of the available third-party plugins. The new abilities supplied by a collection plugin are only limited by your imagination and can range from simple attribute changes, through behavioral changes from monitoring events on those elements, to the complete replacement of the original component with an alternate implementation.

A key component of writing your plugin is testing its functionality, and using a unit test tool enables you to easily and consistently run tests on your code, proving that it works as expected. Once your code it ready to release, it needs to be packaged for distribution so that others can obtain it easily and integrate it with their own project. You should also provide a web page that demonstrates the capabilities of your plugin to allow prospective users to see how it works and what it can do. And you must supply documentation for every aspect of your plugin to let others get the most out of it.
**Function plugins**

Function plugins are utility functions that don’t directly operate on collections of elements. They offer additional abilities within the jQuery framework and usually make use of jQuery’s own functionality to perform their duties.

Examples of these function plugins include support for sending debugging messages to a console for monitoring code execution or the retrieval and setting of cookie values for a web page. By making these abilities available as a jQuery plugin, you provide the user with a familiar way to invoke the code and reduce possible interference with external code. Several of the guidelines mentioned earlier still apply to these sorts of plugins, as do the steps of testing, packaging, demonstrating, and documenting the plugin.

**jQuery UI widgets**

jQuery UI "provides abstractions for low-level interaction and animation, advanced effects and high-level, themeable widgets, built on top of the jQuery JavaScript Library, that you can use to build highly interactive web applications.” It defines a widget framework that allows you to create plugins that work in a consistent manner and that can take advantage of the numerous themes available for styling the UI.

The jQuery UI widget framework also implements plugin guidelines and provides common functionality to all jQuery UI widgets in a consistent manner. By basing your plugin on this framework, you gain these built-in abilities automatically and can concentrate on delivering the unique functionality of your own widget. If you apply the classes defined in the ThemeRoller styling to your new widget, it will immediately be visually integrated with other jQuery UI components and will change appearance if a new theme is applied.

Several jQuery UI widgets rely on mouse-drag actions to implement their functionality, and the jQuery UI team has recognized the importance of this interaction. By having your widget extend the jQuery UI Mouse module instead of the basic Widget one, you gain support for drag operations, complete with customizable conditions for starting a drag and can again focus on implementing your own widget’s functionality.

**jQuery UI effects**

jQuery UI also provides a set of effects that may be applied to elements within your page. Many of these are used to show or hide an element, such as “blind,” “clip,” “fold,” and “slide.” Some bring your attention to an element, such as "highlight" and “pulsate.” You can define your own effect and apply it to elements just like the standard ones.

**Animation**

jQuery provides an animation framework that can be applied to any element style attribute that has a simple numeric value. It allows you to vary that attribute from one value to another, controlling the duration of the change and the incremental steps along the way. However, if the value you want to animate is not a simple numeric value, you need to implement the functionality yourself. For example, jQuery UI provides a module that allows you to animate from one color to another.

**Ajax**

jQuery’s Ajax functionality is one of its clear benefits, making it incredibly easy to load remote data and then process it. As part of the Ajax call you may identify what type of data is expected by the success callback: plain text, HTML, XML, JSON. A conversion process happens behind the scenes to transform the byte stream received by the remote call into the appropriate format. You can add your own transformations to allow you to produce specialized formats directly by simply identifying which type you want returned.

**Events**

The event handling abilities of jQuery allow you to attach multiple event handlers to elements to respond to user interactions, system events, and custom triggers. jQuery provides several hooks to let you create your own event definitions and trigger points, resulting in code that is consistent with the existing functionality.

**Validation**

The Validation plugin written by Jörn Zaefferer is widely used to validate user entry on the client side before submitting completed values to the server. Although not a part of the core jQuery functionality, it also provides...
extension points to allow you to create custom validation rules and have them applied as part of the existing processing.

**Summary**

jQuery has grown to be the most widely used JavaScript library on the Web today. Although it has a lot of built-in functionality, it concentrates on providing the basic infrastructure and abilities used by many people across many websites. Recognizing the fact that it can’t provide everything for everyone, it includes numerous extension points whereby others can extend its behavior.

You can add functionality to just about every part of jQuery, from custom selectors, through animating non-numeric attribute values and defining new events, to fully blown UI components. The only limits are your imagination.

Creating a plugin for your code lets you more easily re-use it in many web pages. It reduces your testing and maintenance burden as there is only one copy of the script.
Here are some other Manning titles you might be interested in:

- **Rails 3 in Action, Second Edition**
  Ryan Bigg

- **Sass and Compass in Action**
  Wynn Netherland, Nathan Weizenbaum, and Chris Eppstein

- **Secrets of the JavaScript Ninja**
  John Resig and Bear Bibeault

Last updated: September 18, 2012