

Integrating jQuery in SharePoint 2010

One of the main reasons for SharePoint's success is that, out of the box (OOB), it is highly customizable and adaptable, letting you meet users' functional requirements. However, if you take advantage of jQuery, you can create not only functional but also fashionable implementations. In this article, I will introduce you to jQuery and show you how to include it in your SharePoint applications.

Although a SharePoint implementation can have the best personalized Web parts, the best list filters, and be optimized to obtain excellent performance, it's not uncommon to hear users say comments such as "The style is very SharePoint-like." What the users are trying to express is that many SharePoint implementations are very similar, or at least they have many features in common, not taking into account all the "junk" contained in it. These users probably don't know that attractive and avant-garde websites like [Ferrari](#) and [Starbucks](#) are implemented completely with SharePoint.

SharePoint offers all the tools needed to make a SharePoint portal as personalized as possible, internally and externally, but the developer has to work on this personalization, which is a hard process. For this reason, the elements that SharePoint includes by default are often reused and innovation and design are left aside. But by using jQuery, you can make your SharePoint implementation less "SharePoint-like."

What is jQuery?

You can find many definitions of jQuery on the web, some of them are more accurate than others. One definition that I consider "curious" is:

"To me it's the difference between avoiding JavaScript as much as possible, and embracing it wholeheartedly. jQuery abstracts away the hard stuff like DOM incompatibilities, leaving me free to write fairly basic script to accomplish what I need. And the CSS-like selector syntax is absolutely wonderful. I've already learned that so it's building on what I know. I'm still not convinced I'm much of a scripter, but writing with jQuery makes me feel like I'm actually somewhat in control when it comes to JavaScript. And the joy of seeing my script work as expected first time 'round across the board when testing in various browsers? Undefinable."

This definition is from David Shea, co-author of *The Zen of CSS Design* (Peachpit Press, 2005). The fact that someone like him, toughened by a thousand battles with website designs, offers such a definition makes it even more valuable. Furthermore, it has the same meaning as jQuery's slogan "Write less, do more."

In essence, jQuery is a JavaScript framework (aka library) with a superior abstraction layer that lets developers write scripts very easily and elegantly. Internally, it is JavaScript, but developers perceive it as a group of tools that lets them easily access Document Object Model (DOM) nodes, manipulate events, work with Ajax, and more. With jQuery, developers don't have to worry about whether an application is going to work correctly on different browsers or whether they need to reduce the number of code lines. It is a cross-browser framework that has a lightweight footprint.

In my opinion, jQuery became the framework that it is nowadays when Microsoft (along with Nokia) included it in its Visual Studio development platform in 2008. Microsoft not only supported this open-source project for the first time but also collaborated closely on its development and extension.

This article is not meant to teach jQuery syntax. Undoubtedly, the best information source about jQuery syntax is the [jQuery official documentation](#). In it, you can find thousands of examples, numerous tutorials, and a huge user community in which users support each other. The objective of this article is to show you how to include jQuery in your SharePoint website so you can make that website more attractive to users. For that reason, I am assuming that you have basic knowledge about SharePoint, HTML, Cascading Style Sheets (CSS), and JavaScript.

How Do I Include jQuery in SharePoint?

As I mentioned previously, jQuery is intrinsically JavaScript. As a result, it's not surprising that you need to include a JavaScript file (.js) in the <header> tag of the website document (ASP.NET) like this:

```
<script src="/Style Library/js/Scripts/
jquery-1.5.2.min.js"
type="text/javascript"></script>
```

The latest version of jQuery can be downloaded from the jQuery website and other websites, such as Google Code and Microsoft. It is advisable to use the Content Delivery Network (CDN) version because it is very probable that users already have it in their browser cache, thereby reducing the load time for the site. When using a CDN version, the <header> tag of the website document (ASP.NET) might look like this:

```
<script src="http://ajax.microsoft.com/ajax/
jquery/jquery-1.5.2.min.js"
type="text/javascript"></script>
```

However, you must remember that you are in a SharePoint environment. For that reason, you can use the utilities that it offers. I will show you three different ways to include jQuery in SharePoint: using the Ajax Script Loader, using a delegate control, and using a ScriptLink control with the ScriptSrc Attribute. They all have their advantages and disadvantages, so you have to choose the best one given your requirements.

Using the Ajax Script Loader

A direct consequence of Microsoft's support for JQuery in 2008 is the adaptation of the ASP.NET Ajax library and the inclusion of jQuery in the Ajax Control Toolkit. Because this adaptation needs jQuery to work, the Ajax Script Loader (which is a script named Start.js) has been adapted to load jQuery scripts OOB. (The Script Loader is also used to load other types of scripts, such as ASP.NET Ajax Library scripts.) Therefore, you can use Start.js to load jQuery in all the websites in your SharePoint environment by referencing Start.js from the master page. This is the most convenient way to reference it because you will then have access to all the jQuery framework functions in all the pages. Furthermore, it only loads the first time a user visits due to the fact that browsers use the in-

cache version for future visits. Referencing Start.js from the master page is a three-step process. Let's take a look at those steps.

Step 1: Download Start.js

The first step is to download Start.js. You can download this script directly from the Microsoft Ajax CDN by entering <http://ajax.microsoft.com/ajax/beta/0910/Start.js> in your browser. Alternatively, you can download the development version by entering <http://ajax.microsoft.com/ajax/beta/0910/Start.debug.js>.

In SharePoint community, there is a big debate about which site is the best to host JavaScript, image, CSS, and Extensible Style Language Transformations (XSLT) files. I think the best option is to save JavaScript files in a folder (with an intuitive name as "js") inside the Styles Library of the Site collection for two reasons. First, it is logical. The Styles Library exists OOB, so why not use it? The second and more important reason is that you can use the JavaScript files in a sandboxed solution. If you save those files in the LAYOUTS folder in the SharePoint root on the server file system, you can't use them in a sandboxed solution. Nevertheless, although I recommend saving the JavaScript scripts in the Styles Library, you are free to save the scripts wherever you want.

Step 2: Add the Reference

Next, you need to reference the Ajax Script Loader in the page in which you want to use jQuery. You can do so by including the following <script> tag:

```
<script src="/Style Library/js/Start.debug.js"
type="text/javascript"></script>
```

Or, if you want to load it directly from the Microsoft Ajax CDN, you can use this <script> tag:

```
<script src="http://ajax.microsoft.com/
ajax/beta/0910/Start.debug.js"
type="text/javascript"></script>
```

Step 3: Call Sys.loadScripts

The last step is to reference the jQuery library (or any other library you want to use) using the *Sys.loadScripts method*, which is part of Start.js. As Listing 1 shows, the Sys.loadScripts method takes two arguments: the script to load and a callback function that calls it after it has been loaded.

Listing 1: Code that uses the Sys.loadScripts method

```
<script type="text/javascript">
  Sys.loadScripts("[/Style Library/js/jquery-1.5.2.min.js]", function(){
    alert("jQuery Loaded");
  })
</script>
```

The advantage of using Ajax Script Loader is that you can avoid the situation in which you need to load jQuery many times from different points in the same page, even if Web parts are being used to implement this same code.

Using a Delegate Control

One way of elegantly adding jQuery in each SharePoint page is using a delegate control. The delegate control is a powerful SharePoint utility that lets you define regions inside the master pages, which can be replaced by appropriate code for the purpose of meeting some requirements. The most interesting thing about the delegate control is that it doesn't need to alter the master page because the operations in question are made using *Features*. Furthermore, all the OOB master pages that come with SharePoint (e.g., v4.master, default.master, minimal.master, nightandday.master) include the following code in the <head> tag:

```
<SharePoint:DelegateControl runat="server"
  ControlId="AdditionalPageHead"
  AllowMultipleControls="true"/>
```

As I previously mentioned, it is not necessary to manipulate those master pages. You only have to create a Feature. Let's take a look at how to do so.

Step 1: Create a User Control

To begin, you first need to create a new empty SharePoint project in Visual Studio 2010. Next, you add a user control to that project. For this example, call it *jQueryControl.ascx* and add the following code to it:

```
<script type="text/javascript"
  src="/Style Library/js/jquery-1.5.2.min.js" />
```

Creating and deploying a user control in SharePoint is that simple. You can now use this control in all of your pages.

Step 2: Add a Feature

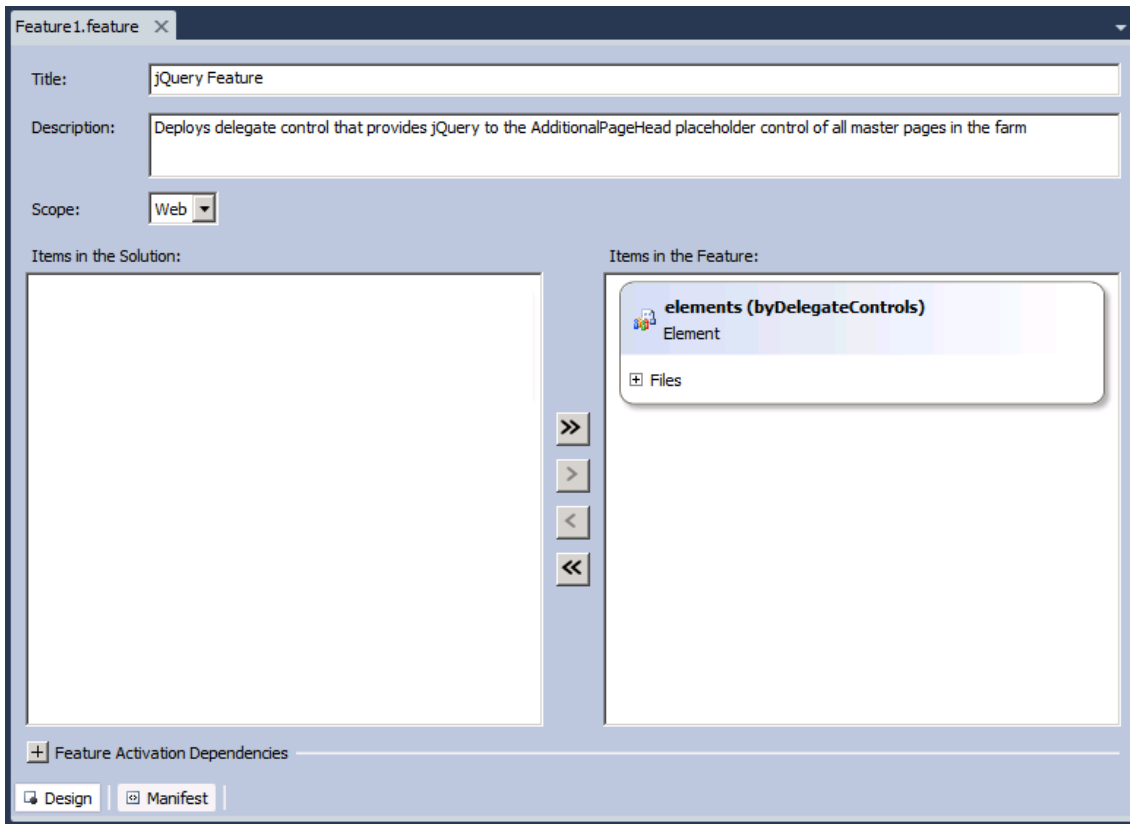
The next step is to add a Feature to the project. The Feature will tell SharePoint where to find the *jQueryControl.ascx* user control. For that reason, you need to add a new *empty element* to your project. Once created, you use the code in Listing 2 to change the content of the *Elements.xml* file. This code registers *jQueryControl.ascx* as a delegate control.

Listing 2: Code to register *jQueryControl.ascx* as a delegate control

```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <Control Id="AdditionalPageHead"
    Sequence="90"
    ControlSrc="~/_CONTROLTEMPLATES/byDelegateControls/jQueryControl.ascx" />
</Elements>
```

Now that you have created the element that is going to be part of your Feature, you need to edit it. To do this, open the Feature Designer and select that element. Give it a title, a description, and most important, a scope, as Figure 1 shows.

Figure 1: Editing the element that is going to be part of the Feature



If you select Web as the scope, those websites in which the Feature is activated will also have jQuery automatically activated. If you want a wider level, select Site (in other words, the Sites Collection). You can also choose the scope of WebApplication or Farm.

Step 3: Activate the Feature

Figure 2 shows the completed jQuery Feature. It provides an easy way to activate or deactivate jQuery in those places where you don't need a library, such as in an intranet, and where the content is more important than the container.

Figure 2: The completed jQuery Feature



The big advantage of using a delegate control is that it is not included by default in many SharePoint application pages, such as SiteManager.aspx. Those pages do have, however, a control named *ScriptLink*, which I will discuss next. The question you need to ask yourself is, "Do I need to alter the compartment of any SharePoint application page using jQuery?" If not, using a delegate control is a very elegant and valid option. If you do need to alter a compartment, using the ScriptLink control is a better option.

Using a ScriptLink Control with the ScriptSrc Attribute

The ScriptLink control is found in all of the OOB master pages in SharePoint. This control allows you to add some resources (in this case, jQuery) when rendering the current page. SharePoint 2010 includes a new attribute for the *CustomAction* element of a Feature: ScriptSrc. With this attribute, you can associate a script with the activation of a Feature. Here is how to join all these pieces together.

Step 1: Create a User Control

You first need to create a new empty SharePoint project to host your Feature, like you did when using a delegate control. Add a user control to that project, then add the following code to it:

```
<script type="text/javascript"
src="/Style Library/js/jquery-1.5.2.min.js" />
```

Step 2: Add a Feature

Next, you need to add a Feature to your project by adding a new *empty element*. Once created, you use the code in Listing 3 to change the content of the Elements.xml file. This code deploys the Feature inside a sandboxed solution. Note that to reference virtual directories (such as the Style Library) inside the ScriptSrc attribute, it is necessary to add the prefix *SiteCollection*. It is a very simple but necessary addition. Finally, you need to edit the Feature in the Feature Designer, giving it a title, a description and, most important, a scope.

Listing 3: Code to deploy the Feature inside a sandboxed solution

```
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <CustomAction
    ScriptSrc="~SiteCollection/Style Library/jquery-1.5.2.min.js"
    Location="ScriptLink"
    Sequence="10">
  </CustomAction>
</Elements>
```

Step 3: Activate the Feature

The last step is to activate the Feature in the target scope. If you then check the header in the HTML source code for the page in question, you will find code like this:

```
// <![CDATA[
...
document.write('<script type="text/javascript"
src="/style%20library/js/jquery.min.js">
</' + 'script>');
...
</script>
```

In other words, the ScriptLink control is present in the master page. It will dynamically add the reference to jQuery. An important disadvantage of this method is that you can't use the CDN versions of jQuery because the ScriptSrc attribute doesn't allow it.

Avoid Conflicts

In jQuery, like in real life, it is very important to avoid conflicts. When you are going to start using jQuery, bear this in mind: SharePoint is full of OOB JavaScript code and this can be a problem. Why? There are two ways to use jQuery. The first approach is to use the \$ character, as in:

```
$("#miDiv");
```

The second approach is to use the reserved word *jQuery*, like this:

```
jQuery("#miDiv");
```

Both lines of code perform the same task (i.e., obtain a div with the property ID of miDiv). In other words, the \$ character is an alias of the reserved word jQuery. The problem is that this alias is also used by other frameworks, such as MooTools, Prototype, and Dojo. In addition, the JavaScript library included in SharePoint uses the \$ character; it is here where conflicts can arise. Fortunately, these conflicts can be easily avoided with this line of code:

```
jQuery.noConflict();
```

By adding this line at the beginning of your jQuery code, you are telling jQuery to not interpret any \$ character as the reserved word jQuery. In other words, jQuery will understand only the reserved word jQuery, thereby avoiding conflicts with SharePoint's JavaScript library. Note that when you call jQuery.noConflict(), you must use the reserved word jQuery instead of the \$ alias in your jQuery code.

A Practical Example

Let's wrap up this article with a practical example that applies what you have learned. Suppose you want to replace an image in the header of a single default.aspx page with a dynamic menu that contains some images. You are going to obtain the images from the library of a SharePoint list. You want to use Ajax to access the Web service that exposes this list and retrieve the necessary fields. The resulting menu is going to look like that in Figure 3. When users hover their mouse over an image, it will get bigger and the rest of the images will get smaller, producing a nice effect.

Figure 3: The new dynamic menu



The first step is to include jQuery in your SharePoint page. For this example, use the ScriptLink control, following the instructions in the “Using a ScriptLink Control with the ScriptSrc Attribute” section. Although the ScriptLink control is being used here, you could use any one of the three ways to include jQuery in SharePoint.

Next, in the <header> tag of the layout you want to modify, you need include a reference to the JavaScript file that contains all the logic for your new menu. In this case, the default.aspx page uses the layout named WelcomeSplash.aspx. To avoid modifying the original file, make a copy of WelcomeSplash.aspx, name it ModifiedWelcomeSplash.aspx, and make your changes to the copy.

In the <header> tag of ModifiedWelcomeSplash.aspx, you need to include the reference to the JavaScript file that contains all the logic for your new menu. This is very important because if you don't do so, you will be calling something that doesn't exist yet.

Another option would be to create a Feature like the one created to include jQuery (e.g., jQuery Feature). In this example, it would not make sense because the new menu will be on one page only. However, if you plan to use the new menu in several sites, you could use the same process to create a Feature for the JavaScript file that contains the new menu's logic. By setting that Feature's scope to Web, you would have a wide range of options on where to apply the menu. You would just need to activate and deactivate the Feature where appropriate.

The last step is to verify that the new menu is working correctly. You can find the code for this example at

<http://www.solidq.com/JournalAssets/Integrating jQuery in SharePoint 2010 Assets.zip>.

About the Author



Cristian M. Zaragoza works for SolidQ in the collaboration area. He was assigned to SolidQ through a practice program at the University of Alicante. After he finished his studies in Computer System Engineering, he became a member of SolidQ. Currently, Cristian works with Microsoft SharePoint Server 2010 technology, mainly performing internal work, such as SolidQ website development and maintenance. He is also taking the "Applications and Web Services Development Master" course at the University of Alicante. Cristian is an MCP in SharePoint 2010 Development and Configuration.



Learn from the best — stand out from the rest.

Training

www.solidq.com