 See This in MSDN Library

Using JavaScript Along with ASP.NET

Bill Evjen
Reuters

February 2004

Applies to:
- Microsoft® ASP.NET
- Microsoft Visual Basic® .NET
- Microsoft Visual C#® .NET

Page Options

Summary: Learn how to apply JavaScript in ASP.NET applications. Also learn ways of doing control focus, button rollovers, alerts and placing text in the browser's status bar. (17 printed pages)

Contents

Introduction

Web developers have been working around the limitations of the browser for a long time by using a combination of server-side processing and client-side processing. Working logic and application processes on the client-side allows browser based applications to seem more responsive and to have more "snappiness" to them. For client-side development in the past, most developers turned to JavaScript and intermingled this programming language into their classic Microsoft® Active Server Pages along with their server-side Microsoft Visual Basic® Scripting Edition or Microsoft JScript® code. Now with Microsoft ASP.NET and the new models that it provides, Web developers often wonder how to properly work with their JavaScript functions in their ASP.NET pages.

This article will take a look at some of the ways past JavaScript functions can now be used. There is more than one way to accomplish this task and this article will take a look at many different possibilities. This article will also take a look at some of the more common uses of JavaScript in ASP.NET pages with some short examples.

Adding JavaScript to a Server Control

It is quite easy to add JavaScript to a specific server control that resides on an ASP.NET page. Let's take a look at the button server control as an example. If you drag and drop a Button HTML server control (`HtmlInputButton` Class) onto a page using either Microsoft Visual Studio® .NET or the ASP.NET Web Matrix and run it as a server control, it should have the following construction:

```
<INPUT type="button" value="Button" runat="server" id="Button1">
```

This is a normal button that can be programmatically manipulated in the code-behind or server-side script of an ASP.NET page. For example, to assign the button text when the page is generated, simply use the **value** property of the button after this element is turned into an HTML server control (right-click on the control and select **Run As Server Control**).

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
     Button1.Value = DateTime.Now.ToString()
End Sub
```

Visual C# .NET

```
void Page_Load(object sender, EventArgs e) {
   Button1.Value = DateTime.Now.ToString();
}
```

This simply provides a button on the page that shows a date and time as the text of the button.

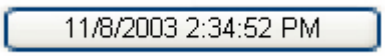

Figure 1. Showing the date and time on a button

It is important to note that the ASP.NET page here gets the time from the server that generated the page. So if the Web server sits somewhere in the Central Time Zone of the United States (CST -6 GMT), then everyone who requests this page will get the same time no matter where they reside in the world.

What if you wanted the button to show the time of the person *viewing* the page? The easiest way of accomplishing this task would be to do this using JavaScript on the client-side.

For an example of this, we will place the end user's (the viewer of the web page) computer time on a button Web server control. The following code shows how to accomplish this task:

Visual Basic .NET

```
<%@ Page Language="VB" %>
<script runat="server">

    Sub Button1_Click(sender As Object, e As EventArgs)
       Response.Write("Postback!")
    End Sub

</script>
<html>
<head>
</head>
<body onload="javascript:document.forms[0]['Button1'].value=Date();">
    <form runat="server">
        <p>
            <asp:Button id="Button1" onclick="Button1_Click"
             runat="server" Font-Bold="True" Font-Names="Verdana"
             Font-Size="Larger"></asp:Button>
        </p>
    </form>
</body>
</html>
```

Visual C# .NET

```
<%@ Page Language="C#" %>
<script runat="server">

   void Button1_Click(object sender, EventArgs e) {
      Response.Write("Postback!");
   }
```

```
</script>
<html>
<head>
</head>
<body onload="javascript:document.forms[0]['Button1'].value=Date();">
    <form runat="server">
        <p>
            <asp:Button id="Button1" onclick="Button1_Click"
             runat="server" Font-Bold="True" Font-Names="Verdana"
             Font-Size="Larger"></asp:Button>
        </p>
    </form>
</body>
</html>
```

In this bit of code, notice how some of the button's attributes are assigned server side before being sent down to the client's browser. In this case, the font of the text on the button is changed to Verdana as well as to a bold font-type of a specific size. Once the button's HTML code is received on the client, the client-side JavaScript changes the text of the button to the current time on the end user's computer. The HTML code generated for the entire page will then appear as such:

```
<html>
<head></head>
<body onload="javascript:document.forms[0]['Button1'].value=Date();">
    <form name="_ctl0" method="post" action="NewFile.aspx" id="_ctl0">
       <input type="hidden" name="__VIEWSTATE"
        value="dDwtNTMwNzcxMzI0Ozs+fGKi5Pun0h+xthnqTZtIR9yEzL4=" />
       <p>
       <input type="submit" name="Button1" value="" id="Button1"
        style="font-family:Verdana;font-size:Larger;font-weight:bold;" />
       </p>
    </form>
</body>
</html>
```

Clicking on the button will still give you a postback (observed through the `Response.Write` command) and a new time on the button control as the page is re-rendered. The result is shown in Figure 2.

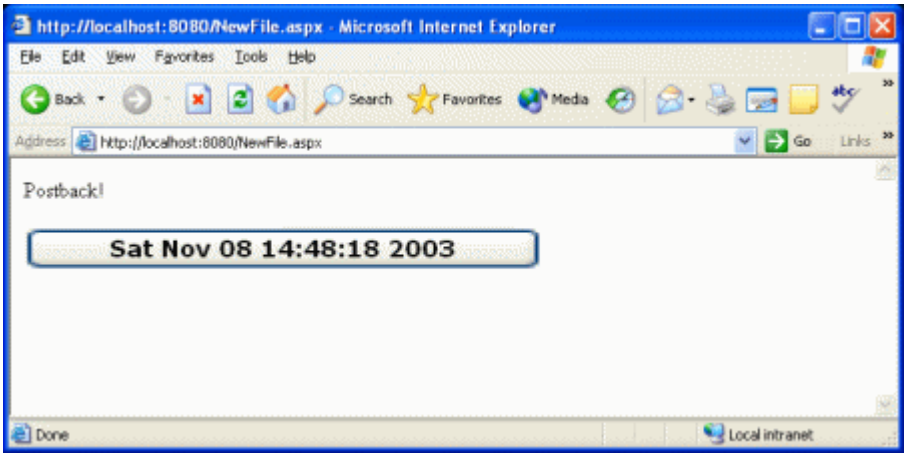

Figure 2. Clicking on the date button

In this case, we placed some JavaScript directly in the <body> element of the page using the **onload** attribute. For the value of the **onload** attribute, we specifically pointed to the HTML element with the name `Button1` that is in the first `<form>` section (as it is possible to have multiple forms in HTML).

This was an easy way to add some JavaScript to work with an ASP.NET Web server control. Though, we could

have also just as easily added a JavaScript command to the button itself as shown here in the following partial code example:

Visual Basic .NET

```
<%@ Page Language="VB" %>
<script runat="server">
    Sub Page_Load(sender As Object, e As EventArgs)
        Button1.Attributes.Add("onclick", _
           "javascript:alert('ALERT ALERT!!!')")
    End Sub
</script>
<html>
<head>
</head>
<body>
    <form runat="server">
        <asp:Button id="Button1" runat="server" Font-Bold="True"
         Font-Names="Verdana" Font-Size="Larger"
         Text="Click Me!"></asp:Button>
    </form>
</body>
</html>
```

Visual C# .NET

```
<%@ Page Language="C#" %>
<script runat="server">
   void Page_Load(object sender, EventArgs e) {
      Button1.Attributes.Add("onclick",
           "javascript:alert('ALERT ALERT!!!')");
   }
</script>
<html>
<head>
</head>
<body>
    <form runat="server">
        <asp:Button id="Button1" runat="server" Font-Bold="True"
         Font-Names="Verdana" Font-Size="Larger"
         Text="Click Me!"></asp:Button>
    </form>
</body>
</html>
```

Using a server control's **attribute** property is a great way to add additional JavaScript to a control that is control specific. In this case, the JavaScript was added using the **Attribute.Add** property along with the key of the script as well as the script itself (both represented as `string` values).

Performing a Simple Button-rollover

When it comes to buttons on a Web page, one of the more common functionalities that Web developers want to give their buttons is a rollover effect. The rollover effect experience is when the end user hovers their mouse over a button on a Web page (without clicking the button) and the button itself changes color or shape. This can be especially useful for Web pages that have multiple buttons, and it would be beneficial from a usability standpoint to notify the end user of the button they would be clicking prior to clicking it.

This was fairly easy to do before server controls came along and it isn't that difficult now with server controls. The code for performing such an operation is as follows:

Visual Basic .NET

```
<%@ Page Language="VB" %>
<script runat="server">
   Sub ImageButton1_Click(sender As Object, e As ImageClickEventArgs)
      Label1.Text = "Postback!"
   End Sub
</script>
<html>
<head></head>
<body>
    <form runat="server">
        <p>
            <asp:ImageButton id="ImageButton1"
             onmouseover="this.src='button2.jpg'"
             onclick="ImageButton1_Click"
             onmouseout="this.src='button1.jpg'" runat="server"
             ImageUrl="button1.jpg"></asp:ImageButton>
        </p>
        <p>
            <asp:Label id="Label1" runat="server" />
        </p>
    </form>
</body>
</html>
```

Visual C# .NET

```
<%@ Page Language="C#" %>
<script runat="server">
    void ImageButton1_Click(object sender, ImageClickEventArgs e) {
       Label1.Text = "Postback!";
    }
</script>
<html>
<head></head>
<body>
    <form runat="server">
        <p>
            <asp:ImageButton id="ImageButton1"
             onmouseover="this.src='button2.jpg'"
             onclick="ImageButton1_Click"
             onmouseout="this.src='button1.jpg'" runat="server"
             ImageUrl="button1.jpg"></asp:ImageButton>
        </p>
        <p>
            <asp:Label id="Label1" runat="server" />
        </p>
    </form>
</body>
</html>
```

Instead of assigning the JavaScript to a server control through the <body> element, this time we used the **onmouseover** and **onmouseout** events of the control. For each of these events, we assigned a JavaScript value. The **onmouseover** event is when the end user hovers their mouse over the control and the **onmouseout** is for actions when the end user removes their mouse from hovering over the control. In our case, we want to show one image while the mouse hovers over the button and then show the original image from when the page was loaded when the mouse moves away from the button.

If you are working directly in the control such as this, instead of specifying the control in the `form` as we did when working with JavaScript in the <body> element, you can use the **this** keyword followed by the property you are trying to change.

Setting Control Focus

In the construction of Web Forms, notice that no property is enabled to set the focus of a Web server control (this will be an added feature in ASP.NET 2.0 Whidbey). Therefore when using the .NET Framework 1.0 or 1.1, you need to employ different methods to accomplish this task. You can do it just as you did before ASP.NET came along—using JavaScript.

For example, if your ASP.NET page has multiple text boxes on it, focus can be set to the first TextBox control when the page is loaded by employing the following code in the <body> tag of the page.

```
<body onload="document.forms[0]['TextBox1'].focus();">
```

Using this construct, when the page is loaded, the element that contains the ID TextBox1 will employ the focus, and this enables the end user to start entering text directly without the need to use the mouse to position the focus.

Changing the Control Focus Dynamically

To change the focus on the ASP.NET page dynamically, turn the <body> tag into a HTML server control.

```
<body id="Body1" runat="server">
```

Next, you can change the focus by placing the following code constructs in your ASP.NET server-side events.

Visual Basic .NET

```
Body1.Attributes("onload") = "document.forms[0]['TextBox2'].focus();"
```

Visual C# .NET

```
Body1.Attributes["onload"] = "document.forms[0]['TextBox2'].focus();";
```

Using this technique, you can assign the focus to other elements in the form. You might find this useful when using the **OnTextChanged** event if there are multiple elements in the form. If you don't do this, the end user must refocus on the form again using their mouse after an **OnTextChanged** event is fired due to the page postback.

Using Larger JavaScript Functions

Now that we can place pieces of JavaScript within HTML elements and even work with JavaScript and Web server controls in a dynamic fashion, how do you go about putting entire JavaScript functions in your code?

There are a couple of ways to accomplish this task and we will take a look at some of the more common methods that you can employ in your ASP.NET code. For this article, we will look at the **RegisterStartupScript** and the **RegisterClientScriptBlock** methods.

> **Note** If you are a Visual Basic .NET developer using Visual Studio .NET 2002 or 2003, then you will have to go into your options to turn on the ability to see these advanced methods. To do this, when in Visual Studio .NET, in the menu bar, click on **Tools > Options**. Then in the Text Editor folder, open the **Basic** folder. Within the Basic folder, uncheck the **Hide advanced members** check box. This allows the methods we will be discussing in this article to be shown with Microsoft Intellisense®. See Figure 3.

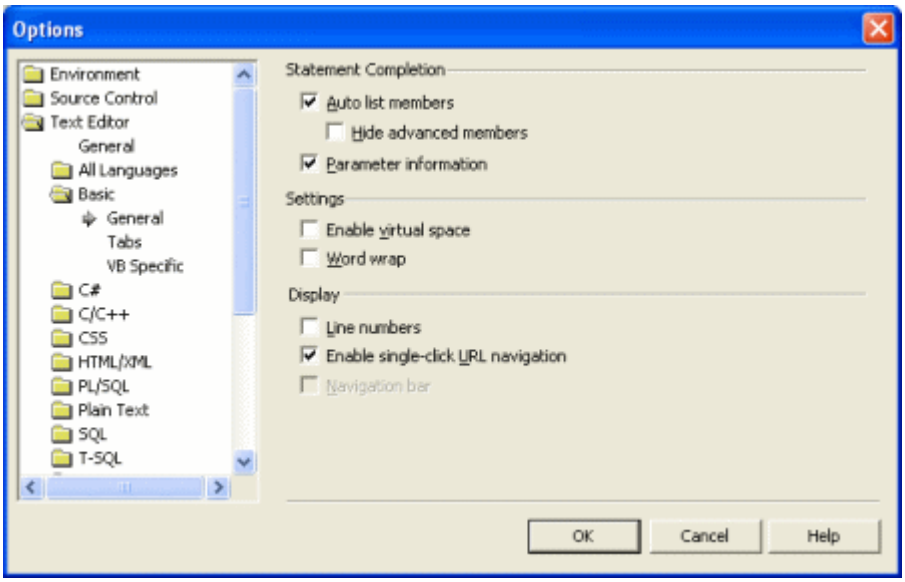

Figure 3. Showing advanced members in Visual Studio .NET

RegisterStartupScript Method

One of the first options available is to register script blocks using one of the .NET classes for this purpose. The first is the **RegisterStartupScript** method. This class would be best used when you have a JavaScript function that you want to initiate when the page is loaded. For an example of this, create an ASP.NET page in Visual Studio .NET that contains two buttons. `Button1` and `Button2` should be the IDs of the two buttons. Then place the following code within the **Page_Load** event.

Visual Basic .NET

```
Page.RegisterStartupScript("MyScript", _
   "<script language=javascript>" & _
   "function AlertHello() { alert('Hello ASP.NET'); }</script>")

Button1.Attributes("onclick") = "AlertHello()"
Button2.Attributes("onclick") = "AlertHello()"
```

Visual C# .NET

```
Page.RegisterStartupScript("MyScript",
   "<script language=javascript>" +
   "function AlertHello() { alert('Hello ASP.NET'); }</script>");

Button1.Attributes["onclick"] = "AlertHello()";
Button2.Attributes["onclick"] = "AlertHello()";
```

Using this code in the **Page_Load** event will produce the following HTML code in the browser (some HTML code removed for clarity):

```
<HTML>
   <HEAD>
      <title>WebForm1</title>
   </HEAD>
   <body>
      <form name="Form1" method="post" action="WebForm1.aspx" id="Form1">
         <P>
            <input type="submit" name="Button1" value="Button"
             id="Button1" onclick="AlertHello()" />
```

```
            <input type="submit" name="Button2" value="Button"
             id="Button2" onclick="AlertHello()" /></P>
            <script language=javascript>function AlertHello() {
             alert('Hello'); }</script>
      </form>
   </body>
</HTML>
```

Working with this ASP.NET page, notice that there is one JavaScript function that was placed at the bottom of the page before the close of the form (`</form>`).

When working with the **RegisterStartupScript** method, the constructor asks for two parameters—the first being the *key* of the script and the second being the *script* itself (represented as a `string`).

```
Page.RegisterStartupScript("MyScript", _
   "<script language=javascript>" & _
   "function AlertHello() { alert('Hello ASP.NET'); }</script>")
```

It is important that all the JavaScripts on the page have unique keys given to them. If there is more than one JavaScript with the same key name, only the first one will be placed on the page.

RegisterClientScriptBlock Method

Now let's create a better version of the button rollover example by using the **RegisterClientScriptBlock** method. The problem with the rollover button example from earlier is that when the end user's mouse hovered over the button image, the rollover image had to be retrieved from the server in a separate request. A better rollover button situation would be where the rollover image of the button is already downloaded and stored in the browser's cache so that when the end user hovers over the button, it is instantaneously displayed. To accomplish this we must build a JavaScript function. The following example shows the JavaScript function as well as the use of the **RegisterClientScriptBlock** method to get the function onto the page. For this example, the code-behind only needs a **Page_Load** event and a **button-click** event for an ImageButton server control.

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles MyBase.Load
        Page.RegisterClientScriptBlock("MyScript", _
            "<script language=javascript>" & _
            "if (document.images) {" & _
            "MyButton = new Image;" & _
            "MyButtonShaded = new Image;" & _
            "MyButton.src = 'button1.jpg';" & _
            "MyButtonShaded.src = 'button2.jpg';" & _
            "}" & _
            "else {" & _
            "MyButton = '';" & _
            "MyButtonShaded = '';" & _
            "}" & _
            "</script>")

        ImageButton1.Attributes.Add("onmouseover", _
           "this.src = MyButtonShaded.src;" & _
           "window.status='Oh Yes! Click here!';")
        ImageButton1.Attributes.Add("onmouseout", _
           "this.src = MyButton.src;" & _
           "window.status='';")
End Sub

Private Sub ImageButton1_Click(ByVal sender As System.Object, _
  ByVal e As System.Web.UI.ImageClickEventArgs) Handles ImageButton1.Click
        Label1.Text = "Postback!"
End Sub
```

Visual C# .NET

```
<%@ Page Language="C#" %>
<script runat="server">
    void Page_Load(object sender, EventArgs e) {
       Page.RegisterClientScriptBlock("MyScript",
           "<script language=javascript>" +
           "if (document.images) {" +
           "MyButton = new Image;" +
           "MyButtonShaded = new Image;" +
           "MyButton.src = 'button1.jpg';" +
           "MyButtonShaded.src = 'button2.jpg';" +
           "}" +
           "else {" +
           "MyButton = '';" +
           "MyButtonShaded = '';" +
           "}" +
           "</script>");

       ImageButton1.Attributes.Add("onmouseover",
          "this.src = MyButtonShaded.src;" +
          "window.status='Oh Yes! Click here!';");
       ImageButton1.Attributes.Add("onmouseout",
          "this.src = MyButton.src;" +
          "window.status='';");
    }

    void ImageButton1_Click(object sender, ImageClickEventArgs e) {
       Label1.Text = "Postback!";
    }
</script>
<html>
<head></head>
<body>
    <form runat="server">
        <p>
            <asp:ImageButton id="ImageButton1"
             onmouseover="this.src='button2.jpg'"
             onclick="ImageButton1_Click"
             onmouseout="this.src='button1.jpg'" runat="server"
             ImageUrl="button1.jpg"></asp:ImageButton>
        </p>
        <p>
            <asp:Label id="Label1" runat="server" />
        </p>
    </form>
</body>
</html>
```

Using this code, the HTML output to the browser will appear as follows:

```
<HTML>
   <HEAD>
      <title>WebForm1</title>
   </HEAD>
   <body>
      <form name="Form1" method="post" action="WebForm1.aspx" id="Form1">
          <script language=javascript>if (document.images) {MyButton = new
          Image;MyButtonShaded = new Image;MyButton.src =
          'button1.jpg';MyButtonShaded.src = 'button2.jpg';}else {MyButton
          = '';MyButtonShaded = '';}</script>
         <P>
            <input type="image" name="ImageButton1" id="ImageButton1"
             onmouseover="this.src = MyButtonShaded.src;window.status='Oh
             Yes! Click here!';" onmouseout="this.src =
             MyButton.src;window.status='';" src="button1.jpg" border="0"
            /></P>
         <P>
```

```
            <span id="Label1"></span></P>
        </form>
    </body>
</HTML>
```

With this output, notice that by using the **RegisterClientScriptBlock**, that the JavaScript function appeared directly after the opening <form> element in the HTML code. In addition to adding a JavaScript function using the **RegisterClientScriptBlock** method, we also added some additional JavaScript (just for fun) so that text will appear in the browser's status bar when the end user hovers over the mouse. This is shown in the Figure 4.

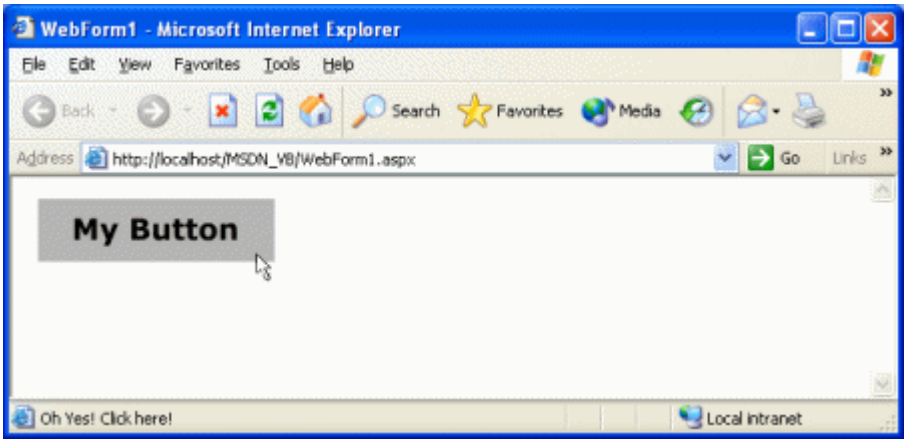

Figure 4. Rollover button in action

The nice thing with all this JavaScript is that the normal postback to server-side events works just fine. Clicking on the ImageButton in this example causes a postback where the Label server control's **text** property is changed.

The Difference Between RegisterStartupScript and RegisterClientScriptBlock

So we have shown two different methods for placing JavaScript functions on an ASP.NET page—so what is the difference? The main difference is that the **RegisterStartupScript** method places the JavaScript at the bottom of the ASP.NET page right before the closing </form> element. The **RegisterClientScriptBlock** method places the JavaScript directly after the opening <form> element in the page. So what difference does this make? It can make quite a bit of difference as we will see.

For an example of this, here is a way to put focus on a text box on a page when the page is loaded into the browser—with Visual Basic .NET using the **RegisterStartupScript** method:

```
Page.RegisterStartupScript("Testing", "<script
  language=javascript>document.forms[0]['TextBox1'].focus();</script>")
```

This works all well and fine because the textbox on the page is generated and placed on the page by the time the browser gets down to the bottom of the page and gets to this little bit of JavaScript. But if instead it was written like this (using the **RegisterClientScriptBlock** method):

```
Page.RegisterClientScriptBlock("Testing", "<script
  language=javascript>document.forms[0]['TextBox1'].focus();</script>")
```

Focus will not get to the textbox control and a JavaScript error will be generated on the page (shown in the Figure 5).

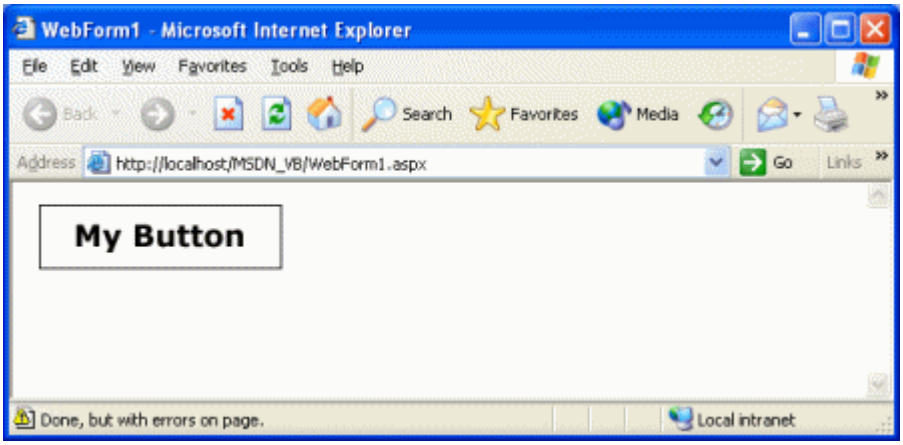

Figure 5. Error executing JavaScript

The reason for this is that the browser will encounter the JavaScript *before* the text box is on the page. Therefore, the JavaScript will not be able to find a `TextBox1`.

Keeping JavaScript in a Separate File (.js)

Keeping JavaScript functions in a separate file (a `.js` file) is highly recommended. Once they are in a separate file and part of a project, the file can be imported into a page using some of the methods already described.

For instance, a `.js` file can be included in an ASP.NET page using the following code:

Visual Basic .NET

```
Page.RegisterClientScriptBlock("MyScript", _
   "<script language=javascript src='MyJavaScriptFile.js'>")
```

Visual C# .NET

```
Page.RegisterClientScriptBlock("MyScript",
   "<script language=javascript src='MyJavaScriptFile.js'>");
```

Once the `.js` file is imported into the ASP.NET page, any of the JavaScript functions can be called as before. This is a great way to manage JavaScript functions and keep them separate from the other logic of ASP.NET pages. It is also an easy way to use the same JavaScript functions on multiple ASP.NET pages.

Summary

This paper took a quick look at some common ways of working with JavaScript in ASP.NET pages and some of the more common uses of JavaScript. Some of the more important tips to take away are to keep JavaScript in a separate `.js` file and to make use of the **RegisterStartupScript** and **RegisterClientScriptBlock** methods to get the JavaScript into pages. It is also quite easy to use the capabilities of the **HtmlGenericControl** to deploy control-specific JavaScript to ASP.NET pages.

About the Author

Bill Evjen is an active proponent of .NET technologies and community-based learning initiatives for .NET. He is a technical director for Reuters, the international news and financial services company based in St. Louis, Missouri. Bill is the founder and executive director of the International .NET Association (INETA), which represents more than 100,000 members worldwide. Bill is also an author and speaker and has written such books as *ASP.NET Professional Secrets*, *XML Web Services for ASP.NET*, *Web Services Enhancements*, and the *Visual Basic .NET*

Bible (all from Wiley).

Top of Page