

AJAX type callbacks using Framework v2.0

by Leslie Drewery

Keynote

We are going to lever the Microsoft .net framework version 2.0 and show you how to make AJAX(Asynchronous JavaScript And XML) call backs using standard framework functionality.

We are going to use the `ICallbackEventHandler` interface. This interface will allow you to implement AJAX (Asynchronise JavaScript and XML) functionality NOW without any major coding and third party controls. This nugget is the core to Microsoft's ATLAS server side controls and it can be used in production applications today. It will require a little more work to implement this at the moment until the final release of Atlas, but the rewards will outweigh the effort.

This is the ideal solution for all you web developers/users that hate full-page posts back to update a simple control. It allows you to implement a lightweight callback to the server and update a specific area of the page, without that much loved blank browser view when the postback occurs.

If you think this is just fancy talk, take a look at start.com, Google suggests, Google maps and even Outlook Web Access: these are a few applications that are full of this technology.

I have been asked why we should bother, with Atlas on the horizon and there being numerous AJAX tools available for download. The main reason is this technology is ready for production applications NOW and will be supported in future releases on the .net framework.

This project source is available from the Downloads page of the Developers Group web site (<http://www.richplum.co.uk/downloads/index.asp>).

Overview

In this example I am going to use Visual Studio 2005 to write a simple web application that uses AJAX. We are going to enter a customer ID and return and display the customer details. The data in the XML file is a direct dump from the Northwind database supplied with SQL Server.

The key goals are.

- Avoid the full-page postbacks, to give the user a more responsive and richer GUI experience.
- Display how simple it is to implement the interface.
- And for those of us that classify ourselves as JavaScript dummies, a trick to avoid excess JavaScript knowledge to display the data.

AJAX

Asynchronous JavaScript And XML, or its acronym **AJAX**, is a web development technique for creating interactive web applications. The intent is to make web pages feel more responsive by exchanging small amounts of data with the server behind the scenes, so that the entire web page does not have to be reloaded each time the user makes a change. This is meant to increase the web page's interactivity, speed, and usability.

ICallbackEventHandler

The `ICallbackEventHandler` Interface can be used with any web control to add XML http callbacks.

The secret to this callback is it executes the standard ASP.net execution model with the exception of the html rendering step, allowing for high-speed lightweight server calls.

To implement this interface a little JavaScript is required in the client and the following three methods, and then we are up and running.

The `ICallbackEventHandler` requires the following three methods to be implemented.

1. `public string GetCallbackResult()`

We use this method to return data to be used by the client side JavaScript to update the required controls directly in the browser.

2. `public void RaiseCallbackEvent(string eventArgument)`

This method is the entry point of the XML HTTP call; the eventArgument holds the values passed from the client browser.

3. `public string GetCallbackEventReference(Control control, string argument, string clientCallBack, string context)`

This method returns a reference to the client-side function which, when invoked, initiates a client callback to a server-side event. The client-side function for this overloaded method includes a specified control, argument, client-side script, and context.

Step By Step implementation of the ICallbackEventHandler

Assumptions

- Visual Studio 2005 will be used.
- A working knowledge of ASP.net using Visual Studio 2005.
- Microsoft .net Framework v2.0 is installed.

Enough formalities: time to get our hands dirty.

Step 1: Create a new web project and save the Default.aspx as CallBack1.aspx.

Step 2: Add a text box and button to the form.

Step 3: Add a div section to the Default WebPage.

```
<div id="UserControlHolder">&nbsp;</div>
```

The data returned from the server call will be displayed in this control's inner HTML area.

Step 4: Create a user control called CustomerDetails to display the requested data.

Create a method LoadCustomerDetails(string customerCode) to populate the usercontrol from the XML file.

In the LoadCustomerDetails we read the XML file, recurse until we find a matching customer ID then update the fields in the control.

```
<%@ Control Language="C#" AutoEventWireup="true" CodeFile="CustomerDetails.ascx.cs"
    Inherits="CustomerDetails" %>

<table class="CustomerTable">
  <tr class="oddRow">
    <td align="right">Customer ID :</td>
    <td>
      <asp:Label ID="lblCustomerID" runat="server" Text=""></asp:Label></td>
    </tr>

  <tr class="evenRow">
    <td align="right">Company Name :</td>
    <td><asp:Label ID="lblCompanyName" runat="server" Text=""></asp:Label></td>
    </tr>

  <tr class="oddRow">
    <td align="right">Contact Name :</td>
    <td><asp:Label ID="lblContactName" runat="server" Text=""></asp:Label></td>
    </tr>

  <tr class="evenRow">
    <td align="right">Contact Title :</td>
    <td><asp:Label ID="lblContactTitle" runat="server" Text=""></asp:Label></td>
    </tr>

  <tr class="oddRow">
    <td align="right">Address :</td>
    <td><asp:Label ID="lblAddress" runat="server" Text=""></asp:Label></td>
    </tr>
```

```

<tr class="evenRow">
<td align="right">City :</td>
<td><asp:Label ID="lblCity" runat="server" Text=""></asp:Label></td>
</tr>

<tr class="oddRow">
<td align="right">PostalCode :</td>
<td><asp:Label ID="lblPostalCode" runat="server" Text=""></asp:Label></td>
</tr>

<tr class="evenRow">
<td align="right">Country :</td>
<td><asp:Label ID="lblCountry" runat="server" Text=""></asp:Label></td>
</tr>

<tr class="oddRow">
<td align="right" >Phone :</td>
<td><asp:Label ID="lblPhone" runat="server" Text=""></asp:Label></td>

</tr>
<tr class="evenRow">
<td align="right">Fax :</td>
<td><asp:Label ID="lblFax" runat="server" Text=""></asp:Label></td>
</tr>
</table>

```

```

using System;
using System.Xml;
using System.Data;
using System.Configuration;
using System.Collections;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

public partial class CustomerDetails : System.Web.UI.UserControl
{
    private XmlDocument _xmlDocument = new XmlDocument();
    protected void Page_Load(object sender, EventArgs e)
    {

    }

    public void LoadCustomerDetails(string customerCode)
    {
        string dataFile = Server.MapPath("~/Data/Customers.XML");
        _xmlDocument.Load(dataFile);
        XmlNodeList nodeList = _xmlDocument.GetElementsByTagName("Customer");
        if (nodeList != null)
        {
            lblCustomerID.Text = customerCode.ToUpper();
            foreach (XmlNode node in nodeList)
            {
                if (node.Attributes["CustomerID"] != null)
                {
                    if (node.Attributes["CustomerID"].Value.ToUpper() ==
customerCode.ToUpper())
                    {
                        if (node.Attributes["CompanyName"] != null)
                            lblCompanyName.Text =
node.Attributes["CompanyName"].Value.ToString();
                        else
                            lblCompanyName.Text = "&nbsp;";

                        if (node.Attributes["ContactName"] != null)
                            lblContactName.Text =
node.Attributes["ContactName"].Value.ToString();
                        else
                            lblContactName.Text = "&nbsp;";

                        if (node.Attributes["ContactTitle"] != null)
                            lblContactTitle.Text =
node.Attributes["ContactTitle"].Value.ToString();
                        else

```

```

        lblContactTitle.Text = "&nbsp;";

        if (node.Attributes["Address"] != null)
            lblAddress.Text = node.Attributes["Address"].Value.ToString();
        else
            lblAddress.Text = "&nbsp;";

        if (node.Attributes["City"] != null)
            lblCity.Text = node.Attributes["City"].Value.ToString();
        else
            lblCity.Text = "&nbsp;";

        if (node.Attributes["PostalCode"] != null)
            lblPostalCode.Text =
node.Attributes["PostalCode"].Value.ToString();
        else
            lblPostalCode.Text = "&nbsp;";

        if (node.Attributes["Country"] != null)
            lblCountry.Text = node.Attributes["Country"].Value.ToString();
        else
            lblCountry.Text = "&nbsp;";

        if (node.Attributes["Phone"] != null)
            lblPhone.Text = node.Attributes["Phone"].Value.ToString();
        else
            lblPhone.Text = "&nbsp;";

        if (node.Attributes["Fax"] != null)
            lblFax.Text = node.Attributes["Fax"].Value.ToString();
        else
            lblFax.Text = "&nbsp;";
        break;
    }
}
}
}
else
{
    //Do Exception Handling.
}
}
protected void Button1_Click(object sender, EventArgs e)
{
    // TextBox1.Text = DateTime.Now.ToLongTimeString();
}
}
}

```

Save the user control for future use.

Step 5 : Create a method to register the client side JavaScript.

```

internal void registerJScript()
{
    StringBuilder jscript = new StringBuilder();
    jscript.Append("<script language=\"javascript\">");
    //If an exception occurs on the server display the error message to the user.
    jscript.Append("function errorCallback(message){");
    jscript.Append("alert('An error occurred on the server:'\n + message);}");

    //Assign the Data returned from the server to the DIV section called
    UserControlHolder
    jscript.Append(" function FetchCustomerDetails(result, context){");
    jscript.Append("document.all['UserControlHolder'].innerHTML = result;}");

    jscript.Append("</script>");

    //Register the the Javascript, this method will prevent the Script from been
    registered more than once,
    //Note : The Key parameter must be unique to each block of JavaScript you wish to
    register.
    Page.RegisterClientScriptBlock("CallbackScript", jscript.ToString());
}

```

We register two JavaScript methods.

- **ErrorCallback**
We use this method to display an error message, if an error occurs on the server.
- **FetchCustomerDetails**
This method handles the data returned from the server; we update the Inner HTML of the UserControlHolder div control we inserted in step 3.

Step 6 : Add and implement ICallbackEventHandler interface to the default page.

```
public partial class Callback1 : System.Web.UI.Page , ICallbackEventHandler

#region ICallbackEventHandler Members
    private string _arguments = null;

    public string GetCallbackResult()
    {
        //Load the UserControl to display the customer information.
        CustomerDetails userControl = (CustomerDetails)LoadControl("~/Customerdetails.ascx");

        /*
         * This little tick helps the JavaScript Challenged have cool webpages.
         * Load the User Control, Fetch the Customer Details using the public method
         * LoadCustomerDetails
         * on the UserControl.
         * We then Render the Control using the RenderControl Method on the UserControl.
         * Tip : This Method is available on all WebControls.
         * We use the StringWriter to return the HTML Rendered from the UserControl.
         */
        StringWriter writer = new StringWriter();
        HtmlTextWriter htmlText = new HtmlTextWriter(writer);
        userControl.LoadCustomerDetails(_arguments);
        userControl.RenderControl(htmlText);

        return writer.ToString();
    }

    public void RaiseCallbackEvent(string eventArgument)
    {
        //Save the Customer ID sent by the client.
        _arguments = eventArgument;
    }

#endregion
```

The thing to note is that the RaiseCallbackEvent is called first, allowing you to initialise any objects. Upon completion the GetCallbackResult method is called to generate the return data.

RaiseCallbackEvent

All I do is save the eventArguments that is sent in the callback to a private variable: in the example the eventArgument will contain the customer ID.

GetCallbackResult

This method is where the magic happens. In the example, we lever off the user control's standard functionality by making it render the html itself. If you so wish you can also pass back the data in any format as long as the JavaScript on the client can process it.

We create an instance of the user control CustomerDetails.

```
CustomerDetails userControl = (CustomerDetails)LoadControl("~/Customerdetails.ascx");
```

We use HtmlTextWriter to render the html and a StringWriter as a buffer to store the rendered html.

```
StringWriter writer = new StringWriter();
HtmlTextWriter htmlText = new HtmlTextWriter(writer);
```

We then populate the user control using the public method LoadCustomerDetails on the user control.

```
userControl.LoadCustomerDetails(_arguments);
```

Now we take the lazy route and make the control render itself: we now have the Html representation of the user control. We pass this control's html data back to the client to where the JavaScript updates the UserControlHolder with the Html.

```
userControl.RenderControl(htmlText);  
return writer.ToString();
```

Step 7 : Wiring up The JavaScript callbacks to the server control,

```
protected void Page_Load(object sender, EventArgs e)  
{  
    //Write JavaScript to the document.  
    registerJScript();  
  
    //Assign the Text Box Value to the Arugments to be posted back.  
    string argument = "document.getElementById(`" + TextBox1.ClientID + `).value";  
  
    //retrieve the javascript method call declaration generated for us  
    string scriptDeclaration = this.ClientScript.GetCallbackEventReference(this,  
        argument, "FetchCustomerDetails", "null", "ErrorCallback", false);  
  
    //Wire up Call back event two the onClick on the button.  
    this.Button1.Attributes.Add("onclick", scriptDeclaration + "; return false;");  
  
}
```

- In the Page Load we register the Client side JavaScript using RegisterJScript() Method.
- We assign the variable to be passed back to the argument variable, in this case the Text1 Text Box.

```
string argument = "document.getElementById(`" + TextBox1.ClientID + `).value";
```

We use the JavaScript getElementById and the unique ClientID for the text box to return the values for the arguments.

- We now assign the Argument, the JavaScript Methods to be called to process the returned data using the ClientSide.GetCallBackReference method which returns a reference to the server side objects.

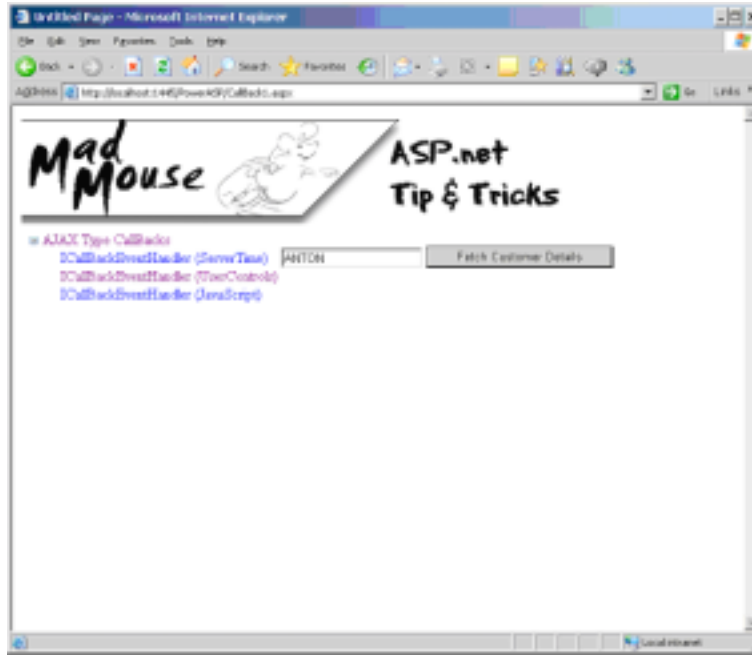
```
string scriptDeclaration = this.ClientScript.GetCallbackEventReference(this, argument,  
    "FetchCustomerDetails", "null", "ErrorCallback", false);
```

- We then assign the Script to the onclick event and prevent postback by returning false.

```
this.Button1.Attributes.Add("onclick", scriptDeclaration + "; return false;");
```

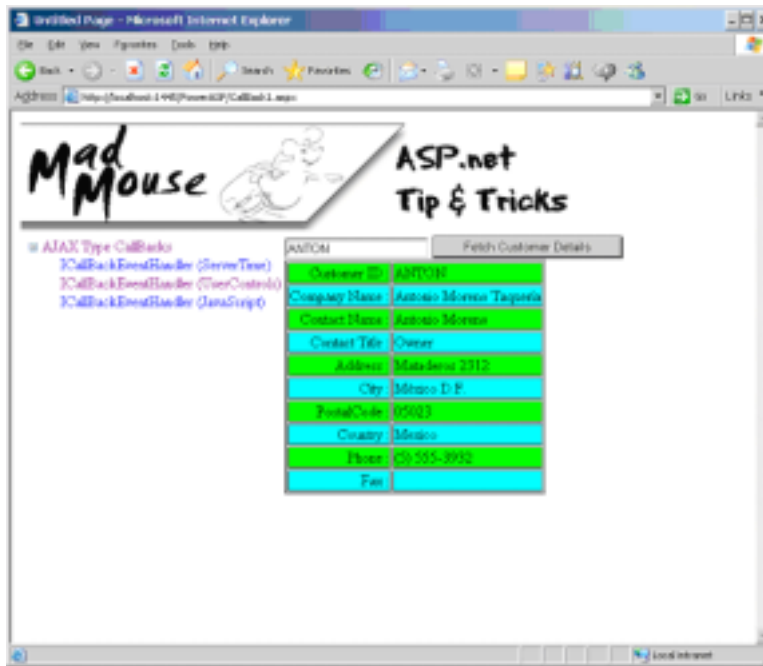
Step 8 : Run the project

Before :



Enter the Client ID in the text box and click on Fetch Customer Details (BERGS,ANTON,HILAA, consult XML File in Data Directory to get more customer details). You will notice the user control appears without a full page postback.

After :



Step 9 : Go brag to colleagues that you have implemented server callbacks into your ASP.net pages.



Leslie Drewery GG (General gopher) is one of the developers for CompuFile Limited, He develops in Delphi and C# covering Win32 through to ASP.net and DirectShow using DirectX ☺. He also has a keen interest in PC based Digital TV systems using .net. He can be reached via his website <http://www.firedtv.com> or email leslie@firedtv.com