

Paper 201-2007

AJAX and SAS®: Smooth Web Applications

Alan Churchill, Savian, Colorado Springs, CO

ABSTRACT

AJAX (Asynchronous Javascript and XML) is an approach to designing websites that emphasizes ease of use for end users. Using one of the new AJAX frameworks simplifies the creation of an AJAX enabled website. Additionally, using web services as a backend provider for SAS, an AJAX site can be created that allows users to request SAS information from a server and display it on a web site in a seamless manner. Step by step instructions for creating a AJAX-enabled, web service backend system working with SAS are provided.

INTRODUCTION

AJAX is fast becoming the approach for developing web applications. Creating an AJAX website that exposes SAS can be done in a few short steps. While a ground-up AJAX site involves a lot of JavaScript code, this paper will demonstrate how to build an AJAX site using an AJAX framework in a few easy steps. Most of the actual work involved will be drag and drop inside of a WYSIWIG editor.

AJAX

What is AJAX? AJAX is an approach to developing web sites and is not a specific technology. It encompasses a number of technical approaches but essentially it is a way to create web sites that do not involve as many round-trips from the client to the server. Also, while XML is in the AJAX name, AJAX can work with any form of data. The only requirement is that the AJAX engine can understand and interpret the data.¹

The term was originally discussed in context of how to make web applications run more seamlessly. As end users became more accustomed to the web, more interactivity was demanded. Soon, web application development had become an effort in frustration for everyone involved from the developer to the end user. The end user would press a button in the browser, wait, and the entire screen would refresh or 'blink'. The developer had a number of tricks that they could use to try and prevent a complete refresh but the developer had to write a lot of code and manage the transfer of information to and from the server. Also, the process was synchronous so nothing could happen until the transfer was complete.

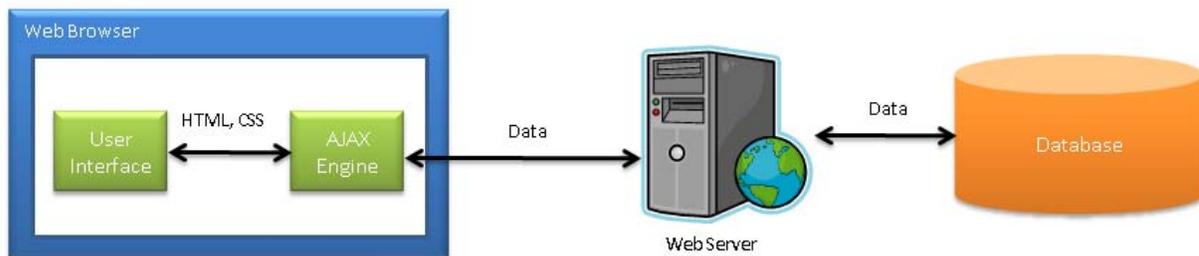
In 2001, Microsoft snuck in a little known ActiveX object into IE5. Known as XmlHttpRequest object, this allowed a web page to use a middle component (XmlHttpRequest) to do the communication to and from the server. In this new model, the developer communicated with the local engine and that engine would handle all of the communication to the server. Moreover, this was an asynchronous process. This functionality was put in at the request of the Microsoft Outlook team who, at the time, were working on the Outlook Web Access (OWA) client. This was really the first AJAX use as it is popularly known now. With the introduction of Google Maps a few years later, the approach of what would become AJAX gained steam. In 2005, Jesse James Garrett of Adaptive Path described the approach and first used the term AJAX. The world suddenly caught on fire and AJAX became very popular.

AJAX is built using JavaScript and HTML. AJAX, by itself, can be very complex to code as will be demonstrated.

Traditional Web Application Model



AJAX Web Application Model



2

Figure 1 AJAX Diagram

¹ (Zacas, McPeak, Fawcett, 2006), p. 5

² (Zacas, McPeak, Fawcett, 2006), p. 5

APPROACHES

AJAX will typically use one of two techniques: hidden iframes or XMLHttpRequest. A hidden iframe is used in a number of applications and was a default choice prior to the advent of XMLHttpRequest. Google Maps, for example, uses hidden iframes to achieve its look and feel. However, the use of iframes to achieve AJAX functionality should wane as the XMLHttpRequest approach becomes more popular and matures. For this reason (and others), this paper will focus on XMLHttpRequest. However, there are pros and cons to each approach. Readers are directed to the following website for more information:

http://ajaxpatterns.org/XMLHttpRequest_Call

READY, AIM, FIRE!

Let's look behind the scenes at some sample AJAX code (partial code shown):

```
<div id="results" style="behavior:url(webservice.htc)"></div>
<input type="image" style="behavior:url(webservice.htc);" onclick='main()' />
function main()
{
    results.useService("http://demo.savian.net/SasWebServicesDemo/Service.asmx?WSDL","ws1");
    iCallID = results.ws1.callService(handleResult,"GetSasDataSetAsHtmlNoMatter", library.value,dataset.value);
    deliveryType = "web services" ;
}

function handleResult(res)
{
    if (!res.error)
    {
        wsReturn = res.value;
        xmlhttpPost() ;
    }
    else
    {
        alert("Unsuccessful call. Error is " + res.errorDetail.string);
    }
}

function xmlhttpPost()
{
    var xmlhttpReq = false;
    var self = this;
    self.xmlhttpReq = new ActiveXObject("Microsoft.XMLHTTP");
    self.xmlhttpReq.open('POST', "http://demo.savian.net/Ajax/CallWebService.htm" , true);
    self.xmlhttpReq.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
    updatePage();
}

function updatePage()
{
    document.getElementById("dt").innerHTML = "<p>Data retrieved using " + deliveryType + " at " + Date() + "</p>";
    document.getElementById("results").innerHTML = wsReturn ;
}
```

Code 1 Raw AJAX JavaScript and HTML

WHAT IS HAPPENING IN THIS AJAX CODE

1. A button is declared that, when pushed, fires the main() function.
2. The main() function then tells the results div tag to call a web service., passing in 3 parameters: a program name of 'GetSasDataSetAsHtmlNoMatter', a library name found in a label, and a dataset name from a dropdown box
3. The web service (a C# program) sends the information to SAS which then executes the code.
4. The web service gets the dataset from SAS and returns the data to the web page.
5. The function handleResult is run which gets the data
6. xmlhttpPost function is then called which does the AJAX 'magic' and passes the data to the web page without refreshing the whole page.
7. Finally, the updatePage function is called which locates the dt div tag and writes that data was retrieved using "web services" delivery type and also updates the results div tag with the SAS data.

The above isn't even a fraction of the code needed to do AJAX cross-platform and robustly. It gets very complex, very quickly. Every browser has slightly different calls to the XMLHttpRequest object so a true cross-browser solution has to account for all of those variants.

I hear the collective groan... "Uggghhhh! I don't want to have to code that for every page. Where's the simplicity of my SAS put statements!?!?"

ENTER THE AJAX TOOLKITS

To simplify working with AJAX, a number of people have created AJAX toolkits to hide most of the complexity. ASP.NET AJAX is one such kit newly released by Microsoft and is what is used in all of the remaining examples.

Here are some equivalent steps to do this in ASP.NET AJAX:

1. Create a Script Manager

```
<asp:ScriptManager ID="ScriptManager1" runat="server" EnablePartialRendering="true" />
```

2. Create a button

```
<asp:Button ID="btnSubmit" runat="server" OnClick="btnSubmit_Click" Text="Submit" /></td>
```

3. Create an update panel with a datagrid control:

```
<asp:UpdatePanel ID="updPanel" runat="server">
  <Triggers>
    <asp:AsyncPostBackTrigger ControlId="btnSubmit" EventName="Click"/>
  </Triggers>
  <ContentTemplate>
    <asp:Panel ID="SasData" runat="server">
      <asp:DataGrid ID="SasGrid" runat="server" ...style info.../>
    </asp:DataGrid>
    </asp:Panel>
  </ContentTemplate>
</asp:UpdatePanel>
```

That is mostly it. There is some 'code behind'³ that is shown below but the above is a vast simplification from the JavaScript version.

Here is what is happening in the ASP.NET tags:

1. The script manager handles the communication to the XmlHttpEngines
2. A button called btnSubmit is created. Inside of the UpdatePanel is a trigger that is fired when the btnSubmit is clicked.
3. An UpdatePanel can have any valid control or HTML inside of it. The UpdatePanel handles all of the plumbing needed to make the AJAX side work.

Here is the code behind for the page:

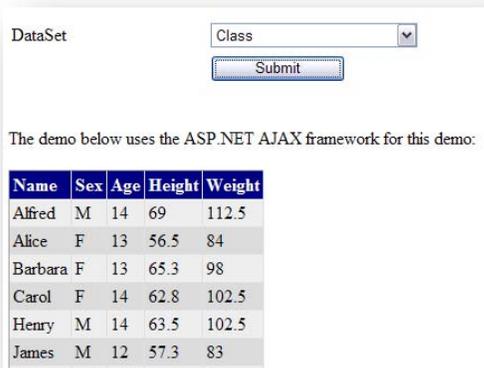
```
protected void btnSubmit_Click(object sender, EventArgs e)
{
    Demo.Service svc = new Demo.Service();
    DataSet ds = svc.LoadSasDataSetIntoDataTable(ddDataSet.Text);
    SasGrid.DataSource = ds.Tables[0];
    SasGrid.DataBind();
}
```

Code 2 ASP.NET Code Behind

In the code behind, a new instance of the web service is created. The service LoadSasDataSetIntoDataTable is called and a .NET dataset is returned. The .NET dataset is then bound to the datagrid and the grid shows up on the screen.

³ Code behind is a term used in ASP.NET. It refers to the code that is used to render and work with the page.

Here is a screenshot of the demo application (with formatting applied):



The screenshot shows a web application interface. At the top, there is a label "DataSet" followed by a dropdown menu labeled "Class" and a "Submit" button. Below this, a text label reads "The demo below uses the ASP.NET AJAX framework for this demo:". Underneath is a table with five columns: Name, Sex, Age, Height, and Weight. The table contains six rows of data.

Name	Sex	Age	Height	Weight
Alfred	M	14	69	112.5
Alice	F	13	56.5	84
Barbara	F	13	65.3	98
Carol	F	14	62.8	102.5
Henry	M	14	63.5	102.5
James	M	12	57.3	83

Figure 2 Demo Screen

The ASP.NET AJAX also handles all cross-platform issues automatically and has a drag-and drop editor so it writes a lot of the code for you. Most of the code above was automatically created for me, the developer, by simply dragging items out and placing them on the designer.

This app can be found at the following URL:

<http://sgfdemo1.savian.net/>

NOTE: The Microsoft ASP.NET AJAX toolkit is in its first iteration. Make sure that if you decide to use this toolkit to get the latest updates and search the web if you encounter any issues. While my experience has been that it is very stable, issues have cropped up in certain situations.

THE MAGIC BEHIND THE SCENES

WEB SERVICES

In the AJAX examples shown so far, all of the data is being transported across the web using web services. Web services allow the application to call SAS or to read SAS data on the same or another server. Their ability to abstract where information comes from and expose it in a common way makes them very attractive and why they were chosen. While this paper is not designed to demonstrate web services, a little explanation is in order.

A web service is exposed to the web through a web server. Think of it as a website to service applications rather than people. For SAS programmers, imagine it is a macro call across the web. Unlike a macro call though, a web service can be consumed by any application that supports web services. Web services are cross-platform and application agnostic. In SAS 9.2, SAS programmers will be able to expose SAS code out as a web service. Web services have become a de facto standard for application interoperability.

Here is a diagram showing how the AJAX application shown here uses web services for reading SAS datasets:

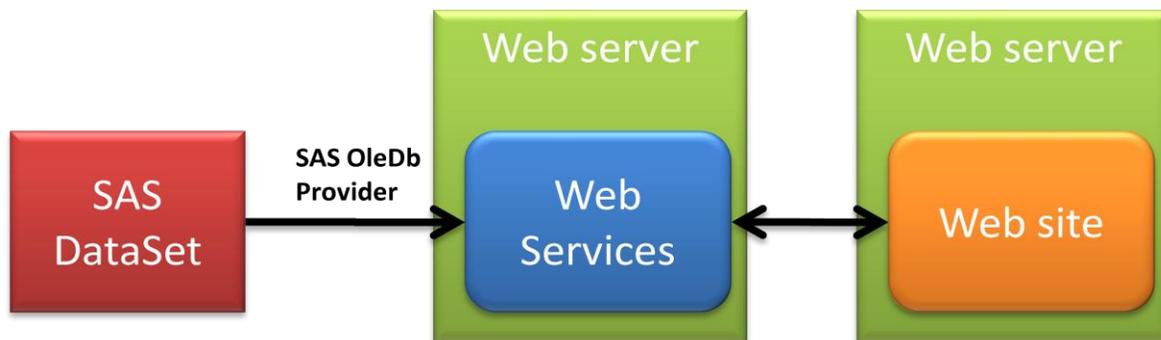


Figure 3 Web service call to SAS

Currently, in order to expose SAS data or functionality out as a web service, enabling technologies are used. In the demos shown here, a C# wrapper was created to make this happen. While this may sound intimidating to a person unfamiliar with C#, numerous examples exist on the web for how to make this happen. Microsoft even provides a guide⁴ for how to create your first web service using C# and even provides a free development environment to help out⁵.

⁴ The guide can be found in the Visual Studio help files by looking under XML Web Services in the index

⁵ <http://msdn.microsoft.com/vstudio/express/>

Let's look at the last web service that was used (**LoadSasDataSetIntoDataTable**):

```
[WebMethod]
public System.Data.DataSet LoadSasDataSetIntoDataTable(string sasDataSet)
{
    string conn = @"c:\temp";
    System.Data.DataSet ds = new System.Data.DataSet();
    OleDbConnection sas = new OleDbConnection("Provider=sas.LocalProvider; " +
        "Data Source=" + conn);

    sas.Open();
    OleDbCommand sasCommand = sas.CreateCommand();
    sasCommand.CommandType = CommandType.TableDirect;
    sasCommand.CommandText = sasDataSet;
    OleDbDataReader sasRead = sasCommand.ExecuteReader();
    ds.Tables.Add("OUTDATA");
    ds.Tables[0].Load(sasRead);
    return ds;
}
```

Figure 4 - Web service call to SAS

This web service does the following:

1. Sets the location of the SAS sample data directory
2. Creates an instance of a .NET dataset called ds. Think of a .NET dataset as a SAS library but in memory.
3. Opens an OleDb connection to SAS using the SAS OleDb provider available, for free, on the SAS website
4. Opens the connection
5. Opens the dataset specified when the service is called
6. Reads the data and adds it to a new table called OUTDATA
7. Returns the .NET dataset

PUTTING IT ALL TOGETHER

Ok, you should now have a basic understanding of AJAX. However, setting it all up along with a web service may be a bit confusing. So here is how I make all of this happen, step-by-step. (Note: some of these steps can be modified to achieve other functionality or for other platforms):

1. Install Visual Studio 2005 and the ASP.NET AJAX framework. Also install the framework on a web server running IIS. In the following example, the web server and the development platform are the same for simplicity. You may be able to use Visual Studio Express as well.
2. Fire up Visual Studio 2005 → File → New → Web Site → ASP.NET Web Service
3. At the top of the code that is displayed, add in the following 2 using statements (this will simplify the amount of typing):

```
using System.Data
using System.Data.OleDb
```

4. Paste in the code shown in Figure 4 - Web service call to SAS
5. Press F5 to test the website.
6. Build → Publish web site → Specify a directory
7. Make sure you copy your SAS datasets to the directory that you specified in the conn string
8. Go to Windows Explorer → right-click the directory that you just created → Sharing and Security → Web sharing → Share this folder → Name it "SAS" for now → Ok → Ok
9. Set up security to allow a web service to read data from that directory.
 - a. Right-click the directory in Windows Explorer → Properties → Security
 - b. Add in the user ASPNET

Wham! You now have a web service that can read SAS datasets running on your localhost. Time to create a web site that reads it in:

1. Fire up Visual Studio 2005 → File → New → Web Site → ASP.NET AJAX enabled web site
2. At the bottom lower left of the screen, switch from Source to Design
3. Delete the existing ScriptManager1 (this appears to be a bug in this framework version)
4. Drag out a new ScriptManager.
5. Drag out a button from the left onto the designer
6. Drag out an Update Panel. Drag out a GridView (it is under the Data section) and place it in the update panel. Auto-format it if desired.
7. Click on the UpdatePanel and on the right-hand side of Visual Studios, look for the Properties docking control. You should see the properties for the UpdatePanel. Click on the triggers section and then click the 3 dots that appear.
8. New trigger → Trigger when a control raises an event → In the lower frame → click on control → Button1 → Click → Ok → Ok

Interesting. No code yet but the AJAX framework is in place and you never wrote a single line of JavaScript or HTML. Let's hook in the web service now so that we can get that SAS data:

1. In the Solution Explorer (upper right control), right-click the directory → Add Web Reference → <http://localhost/SAS/Service.asmx> (you can also search for it but it may take some time). For the web reference name, type in SASServices. Click Add Reference.

Wow! That wasn't bad. Just a bunch of dragging and dropping, a little typing, and click, click, click...Now for the tiny bit of code to make it all work.

1. Go back to the Default.aspx in the Design mode. Double-click the button. This will wire up the click event and show you the code behind the website. The code you will now write will execute whenever the user clicks a button. (if you ever need to get to this code behind, do the following: In the Solution Explorer, right-click Default.aspx → View Code)
2. Place the following code in the new Button1_Click event (it is cool to type this in rather than cut and paste if you have never experienced Intellisense before):

```
SASServices.Service svc = new SASServices.Service();  
svc.Credentials = System.Net.CredentialCache.DefaultCredentials;  
DataSet ds = svc.LoadSasDataSetIntoDataTable("Class");  
GridView1.DataSource = ds.Tables[0];  
GridView1.DataBind();
```

3. Press F5 (Build) → Ok
4. Click the button and your SAS data should appear. If not, and you receive an authentication error, try switching the DefaultCredentials to DefaultNetworkCredentials. If that doesn't work... Google and good luck: security is the #1 time killer in these applications if things don't go smoothly.

CONCLUSION

AJAX is a welcome change in the web world and helps make the user experience much more palatable. Using an AJAX framework, it is possible to create AJAX enabled sites very quickly. Web services are a new paradigm for how remote communications operate. Since the two technologies work well together and web services should be considered for every project, web services were employed here.

While the initial project may seem intimidating, it is much easier due to the AJAX toolkits and the ease of setting up web services using Microsoft's .NET framework. With a little training time, the world of SAS will open up to your web users AND be enjoyable.

REFERENCES

Ajax. (n.d.). Retrieved February 7, 2007, from Wikipedia: [http://en.wikipedia.org/wiki/Ajax_\(programming\)](http://en.wikipedia.org/wiki/Ajax_(programming))
Web Service. (n.d.). Retrieved February 7, 2007, from Wikipedia: http://en.wikipedia.org/wiki/Web_services
XmlHttpRequest. (n.d.). Retrieved February 7, 2007, from AjaxPatterns: http://ajaxpatterns.org/XMLHttpRequest_Call
Zacas, McPeak, Fawcett. (2006). *Professional Ajax*. Indianapolis: Wiley Publishing, Inc.

ACKNOWLEDGMENTS

Thanks to Don Henderson of Henderson Consulting Services (www.hcsbi.com) for the mentoring, encouragement, and help. Thanks also to Vince DelGobbo of SAS for the encouragement and advice.

RECOMMENDED READING

Professional Ajax, WROX

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Alan Churchill
Savian
68 W Cheyenne Mtn. Blvd
Colorado Springs, CO 80906

Work Phone: 719-687-5954
E-mail: alan.churchill@savian.net
Web: www.savian.net

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. © indicates USA registration.
Other brand and product names are trademarks of their respective companies.