Microsoft ASP.NET AJAX 1.0: A Background

Microsoft ASP. NET AJAX 1.0 lets developers build Web 2.0 sites using the latest Ajax techniques. ASP.NET AJAX extends ASP.NET 2.0 and makes several new tools and techniques available to help you build applications more quickly:

- **Extensions to JavaScript**. ASP.NET AJAX extends the JavaScript library to bring standard object oriented concepts to JavaScript. It brings a formal type declaration system, with support for inheritance. It also provides a significant number of out of the box types, including types such as Sys.Net.WebRequest for working with web services. Finally, it helps to abstract some cross-browser issues such as XML element traversal. This makes it much easier to create robust JavaScript libraries and frameworks which are commonly needed by rich internet applications.
- **ASP.NET Control Extenders**. Extenders are additional ASP.NET controls which can extend the functionality of existing controls with additional Ajax capabilities. A common example is an extender which allows existing textbox controls to have autocomplete functionality with no modification to the extended control. (The autocomplete extender is included with the ASP.NET AJAX Control Toolkit.)
- **UpdatePanels**. UpdatePanels allow your existing ASP.NET controls and web parts to achieve the fluid, no-postback updates of Ajax-based applications with minimal re-coding of your control or part. Quite simply, controls within the UpdatePanel control which ordinarily would post back to update their data will now be routed through an Ajax-style callback, resulting in a silent update back to the server. This makes your application “postback” much less, making interaction with your control more seamless.

With Microsoft ASP.NET AJAX 1.0, you can build more dynamic applications that come closer to the rich style of interruption-free interaction you may see in standard client applications.

Microsoft ASP.NET AJAX 1.0 and SharePoint

Windows SharePoint Services version 3 builds much more directly on top of ASP.NET 2.0; therefore, many of the capabilities of ASP.NET AJAX work directly with SharePoint. However, in a few cases there are some compatibility issues between ASP.NET AJAX and SharePoint which are anticipated to be addressed in the first service pack of Windows SharePoint Services. Specifically, there are some limitations on usages of the UpdatePanel in your web parts and controls. Some approaches are described below to address these limitations, but these are workarounds and as such may cause other issues in your application.

Here are some common scenarios in SharePoint you should be able to achieve with Microsoft ASP.NET AJAX 1.0:

1. Building a more powerful, re-usable JavaScript libraries you can use in your web controls and parts
2. Enabling your web services to render via JSON, resulting in easier usage in JavaScript/Ajax Applications
3. Building a web part that takes advantage of Extender technology to provide richer interaction styles, such as autocomplete on a textbox.
4. Using an UpdatePanel in your web part or control for more fluid, no postback interaction. (this will require some workarounds, however.)

Adding Microsoft ASP.NET AJAX Technology to SharePoint Pages

To extend your SharePoint site with Microsoft ASP.NET AJAX 1.0, you'll need to perform a few steps.

- First, you will need to download and install ASP.NET AJAX on servers in your farm.
- Second, you need to extend web.config with some settings to enable ASP.NET AJAX technology.
- Third, you will need to add the ASP.NET AJAX Script Manager into your master page to enable scenarios such as Extenders or UpdatePanels.

You must install and configure Microsoft ASP.NET 2.0 AJAX Extensions 1.0 to work in a Windows SharePoint Services 3.0 environment. This topic describes the procedures to install and configure the ASP.NET AJAX 1.0 Extensions. The installation process deploys the ASP.NET AJAX 1.0 assembly (System.Web.Extensions.dll) to the Global Assembly Cache (GAC) and also installs the Microsoft AJAX Library JavaScript files. The configuration process tells Windows SharePoint Services 3.0 how to handle Web Parts that have ASP.NET AJAX functionality. It also registers the server controls as safe for use within Windows SharePoint Services 3.0.

Note that the procedure described below applies to .NET Framework 2.0. If you are using .NET Framework 3.5, change the version number in each of the XML snippets from `Version=1.0.61025.0` to `Version=3.5.0.0`.

1. Download ASP.NET AJAX 1.0.
2. To install the extensions, double-click the .msi file and follow the prompts.
3. Extend the SharePoint web.config file, which is typically found in a directory with the following structure: *drive*`\inetpub\wwwroot\VirtualDirectories\`*port number*.
   a. Add the following <sectionGroup> elements within the <configSections> element.

```
<sectionGroup name="system.web.extensions"
type="System.Web.Configuration.SystemWebExtensionsSectionGroup,
System.Web.Extensions, Version=1.0.61025.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35">
```

```
<sectionGroup name="scripting"
type="System.Web.Configuration.ScriptingSectionGroup,
System.Web.Extensions, Version=1.0.61025.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35">
<section name="scriptResourceHandler"
type="System.Web.Configuration.ScriptingScriptResourceHandlerSection,
System.Web.Extensions, Version=1.0.61025.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" requirePermission="false"
allowDefinition="MachineToApplication"/>
<sectionGroup name="webServices"
type="System.Web.Configuration.ScriptingWebServicesSectionGroup,
System.Web.Extensions, Version=1.0.61025.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35">
<section name="jsonSerialization"
type="System.Web.Configuration.ScriptingJsonSerializationSection,
System.Web.Extensions, Version=1.0.61025.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" requirePermission="false"
allowDefinition="Everywhere" />
<section name="profileService"
type="System.Web.Configuration.ScriptingProfileServiceSection,
System.Web.Extensions, Version=1.0.61025.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" requirePermission="false"
allowDefinition="MachineToApplication" />
<section name="authenticationService"
type="System.Web.Configuration.ScriptingAuthenticationServiceSection,
System.Web.Extensions, Version=1.0.61025.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" requirePermission="false"
allowDefinition="MachineToApplication" />
</sectionGroup>
</sectionGroup>
</sectionGroup>
```

b. Add the following controls declaration within the <pages> element, which is located within the <system.web> element.

```
<controls>
  <add tagPrefix="asp" namespace="System.Web.UI"
  assembly="System.Web.Extensions, Version=1.0.61025.0,
  Culture=neutral, PublicKeyToken=31bf3856ad364e35"/>
</controls>
```

c. Add the following assembly declaration within the <assemblies> element.

```
<add assembly="System.Web.Extensions, Version=1.0.61025.0,
Culture=neutral, PublicKeyToken=31bf3856ad364e35"/>
```

d. Add the following verb handlers within the <httpHandlers> element.

```
<add verb="*" path="*.asmx" validate="false"
type="System.Web.Script.Services.ScriptHandlerFactory,
System.Web.Extensions, Version=1.0.61025.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"/>
<add verb="*" path="*_AppService.axd" validate="false"
type="System.Web.Script.Services.ScriptHandlerFactory,
System.Web.Extensions, Version=1.0.61025.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"/>
<add verb="GET,HEAD" path="ScriptResource.axd"
type="System.Web.Handlers.ScriptResourceHandler,
System.Web.Extensions, Version=1.0.61025.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" validate="false"/>
```

e. Add the following script module handler within the <httpModules> element.

```
<add name="ScriptModule"
type="System.Web.Handlers.ScriptModule, System.Web.Extensions,
Version=1.0.61025.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"/>
```

f. Add the following safe control entry within the <SafeControls> element, which is located within the <SharePoint> element.

```
<SafeControl Assembly="System.Web.Extensions,
Version=1.0.61025.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" Namespace="System.Web.UI"
TypeName="*" Safe="True" />
```

g. Add the following scripting web service handlers within the <configuration> element.

```
<system.web.extensions>
  <scripting>
    <webServices>
    <!-- Uncomment this line to enable the authentication
    service. Include requireSSL="true" if appropriate. -->
    <!--
      <authenticationService enabled="true"
      requireSSL = "true|false"/>
    -->
    <!-- Uncomment these lines to enable the profile service.
    To allow profile properties to be retrieved and modified in
    ASP.NET AJAX applications, you need to add each property
    name to the readAccessProperties and writeAccessProperties attributes. -->
    <!--
      <profileService enabled="true"
      readAccessProperties="propertyname1,propertyname2"
      writeAccessProperties="propertyname1,propertyname2" />
    -->
    </webServices>
    <!--
    <scriptResourceHandler enableCompression="true"
    enableCaching="true" />
    -->
  </scripting>
</system.web.extensions>
<system.webServer>
  <validation validateIntegratedModeConfiguration="false"/>
  <modules>
    <add name="ScriptModule" preCondition="integratedMode"
    type="System.Web.Handlers.ScriptModule,
    System.Web.Extensions, Version=1.0.61025.0,
    Culture=neutral, PublicKeyToken=31bf3856ad364e35"/>
  </modules>
```

```
  <handlers>
    <remove name="WebServiceHandlerFactory-Integrated" />
    <add name="ScriptHandlerFactory" verb="*" path="*.asmx"
    preCondition="integratedMode"
    type="System.Web.Script.Services.ScriptHandlerFactory,
    System.Web.Extensions, Version=1.0.61025.0,
    Culture=neutral, PublicKeyToken=31bf3856ad364e35"/>
    <add name="ScriptHandlerFactoryAppServices" verb="*"
    path="*_AppService.axd" preCondition="integratedMode"
    type="System.Web.Script.Services.ScriptHandlerFactory,
    System.Web.Extensions, Version=1.0.61025.0,
    Culture=neutral, PublicKeyToken=31bf3856ad364e35"/>
    <add name="ScriptResource" preCondition="integratedMode"
    verb="GET,HEAD" path="ScriptResource.axd"
    type="System.Web.Handlers.ScriptResourceHandler,
    System.Web.Extensions, Version=1.0.61025.0,
    Culture=neutral, PublicKeyToken=31bf3856ad364e35"/>
  </handlers>
</system.webServer>
```

Adding a ScriptManager into a SharePoint MasterPage

Many components of ASP.NET AJAX require the inclusion of a .NET ScriptManager control within a page.

Although it may be possible in some cases to dynamically insert a script manager from within a control, in many cases the control may not be able to insert the script manager early enough in the page lifecycle depending on how the control is used, making this tricky to get right. Also, the control implementer will need to ensure that multiple instances of their control (or other controls) do not result in the addition of multiple script managers within the page. For these reasons, dynamic insertion of a ScriptManager control from another control is not recommended.

To statically embed a script manager into a page, it is recommended that you add the ScriptManager into the master page of a site.

To do this, open up the master page for your site. Typically, this will be located at <site url>/_catalogs/ masterpage. You can edit this file by opening it in an editor such as Microsoft SharePoint Designer.

Add the following into the markup of your page. A recommended location is right beneath the WebPartManager registration (search for <WebPartPages:SPWebPartManager id="m" runat="Server" />):

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

Using UpdatePanels within SharePoint

UpdatePanels are a very useful addition to ASP.NET AJAX, and represent the simplest way to convert existing, standard ASP.NET controls and parts to take advantage of Ajax techniques. However, there are some changes within Windows SharePoint Services which may get in the way of working with ASP.NET AJAX.

Windows SharePoint Services JavaScript has a “form onSubmit wrapper” which is used to override the default form action. This work is put in place to ensure that certain types of URLs, which may contain double byte characters, will fully work across most postback and asynchronous callback scenarios. However, if your scenarios do not involve double byte character URLs, you may successful disable this workaround and gain the ability to use ASP.NET AJAX UpdatePanels.

To do this, you may need to register a client startup script which disables this workaround, in addition to resetting the default form action:

```
<script type='text/javascript'>_spOriginalFormAction = document.forms[0].action;
_spSuppressFormOnSubmitWrapper=true;</script>
```

This script may be directly embedded in the page, or could be emitted by a control that uses the UpdatePanel. The following is an example of a very simple ASP.NET Web Part which uses UpdatePanel capabilities:

```
using System;
using System.Runtime.InteropServices;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Serialization;
using Microsoft.SharePoint;
using Microsoft.SharePoint.WebControls;
using Microsoft.SharePoint.WebPartPages;
namespace AjaxEnabledWebPart
{
    [Guid("1eb3b0c9-f742-445b-9ade-32338a75adf5")]
    public class SayHello : System.Web.UI.WebControls.WebParts.WebPart
    {
        public SayHello(){}
        Label lblRadius, lblArea;
        TextBox txtRadius, txtArea;
```

```
 protected override void CreateChildControls()
{
    base.CreateChildControls();
    this.EnsureUpdatePanelFixups();
    UpdatePanel up = new UpdatePanel();
    up.ID = "UpdatePanel1";
    up.ChildrenAsTriggers = true;
    up.UpdateMode = UpdatePanelUpdateMode.Conditional;
    this.Controls.Add(up);
    lblRadius = new Label();
    lblRadius.Text = "Radius";
    up.ContentTemplateContainer.Controls.Add(lblRadius);
    txtRadius = new TextBox();
    up.ContentTemplateContainer.Controls.Add(txtRadius);
    up.ContentTemplateContainer.Controls.Add(new LiteralControl("<br>"));
    lblArea = new Label();
    lblArea.Text = "Area";
    up.ContentTemplateContainer.Controls.Add(lblArea);
    txtArea = new TextBox();
    up.ContentTemplateContainer.Controls.Add(txtArea);
    up.ContentTemplateContainer.Controls.Add(new LiteralControl("<br>"));
    Button button = new Button();
    button.Text = "Calculate Area";
    button.Click += new EventHandler(HandleButtonClick);
    up.ContentTemplateContainer.Controls.Add(button);
}
private void HandleButtonClick(object sender, EventArgs eventArgs)
{
    double radius, area;
    radius = Convert.ToDouble(txtRadius.Text);
    area = Math.PI * radius * radius;
    txtArea.Text = area.ToString();
}
```

```
        private void EnsureUpdatePanelFixups()
        {
            if (this.Page.Form != null)
            {
                string formOnSubmitAtt = this.Page.Form.Attributes["onsubmit"];
                if (formOnSubmitAtt == "return _spFormOnSubmitWrapper();")
                {
                    this.Page.Form.Attributes["onsubmit"] = "_spFormOnSubmitWrapper();";
                }
            }
            ScriptManager.RegisterStartupScript(this, typeof(SayHello), "UpdatePanelFixup",
"_spOriginalFormAction = document.forms[0].action; _spSuppressFormOnSubmitWrapper=true;", true);
        }
    }
}
```