## Contents


About AJAX ..... 1

History of the XMLHttpRequest Object ..... 1

**How *XMLHttpRequest object works?*** ..... 2

About JSON ..... 3

Installing ASP.NET AJAX in VS 2005 ..... 5

Dealing with ASP.NET AJAX Main Items ..... 5

ScriptManager ..... 5

the brains of an Ajax page ..... 5

Understanding the ScriptManager ..... 6

Dynamically Assigning ASP.NET AJAX Script References ..... 6

ASP.Net AJAX Toolkit ..... 7

Installing the Control Toolkit for VS 2005 ..... 7

AJAX with UpdatePanels And Web Services ..... 8

Invoking web service methods from JavaScript ..... 8

Invoke ASPX Page Methods ..... 8

Working with the DOM Using AJAX Client library ..... 9

References and Links ..... 9


Latest days I was interested with the best use for ASP.NET Ajax also covering its helpfully features, after many reads for books and websites (blogs, articles) hope to introduce something that help developers.

# About AJAX

The main concept behind Ajax is to enable web pages to make HTTP requests in the background, or *asynchronously*, without reloading an entire page (or, in ASP.NET terms, without a round trip, or a postback). Ajax also allows more responsive UIs to be constructed by drawing on the power of commonly supported browser functions such as, JavaScript, Document Object Model (DOM), and Cascading Style Sheets (CSS).

Creating Ajax-enabled web pages by programming the browser requires knowledge of JavaScript, DOM, and the XMLHttpRequest object, which handles the requests from the client to the server.

The name of the **XMLHttpRequest** object is somewhat misleading because data can be transferred in the form of XML or other text-based formats. The ASP.NET AJAX framework relies heavily on a format called JavaScript Object Notation (**JSON**) to deliver data to and from the server.

# History of the XMLHttpRequest Object

The first implementation of XMLHttpRequest can be found in the 1999 release of Internet Explorer 5. That release included an ActiveX object called XMLHttpRequest that did just what the name suggests; make an HTTP request and get a message back. (The format of the returned message could be an XML message, but that was not a requirement.)
Originally, Internet Explorer engineers needed this functionality for the web frontend to Outlook (Outlook Web Access [OWA]), so they could makeOWAbehave more like a desktop application. As useful as it was, for some time the addition of the XMLHttpRequest object to Internet Explorer went unnoticed by web programmers. However, competing browser developers later incorporated a compatible version in their own applications. Because only Internet Explorer supports ActiveX controls,other browsers implemented the XMLHttpRequest object natively in their browser.

After Internet Explorer, the first browser to support XMLHttpRequest was the Mozilla 1.0 browser (not to be confused with the code name for early Netscape browsers). Subsequent versions of Mozilla as well as derivatives, such as the Camino browser for Mac OS X and Firefox, implement XMLHttpRequest. Apple then added appropriate support
in the 1.2 version of their Safari browser. Safari is based on the KHTML renderer that is part of Konqueror, the web browser of the KDE desktop environment for Linux. Apple engineers later back-ported support for the XMLHttpRequest object to Konqueroras as well.
Opera 8.0 and later also included XMLHttpRequest support in their browser, as did the rather exotic system, Open Laszlo, from IBM.

# How *XMLHttpRequest object works?*

look for the following simple example:

The fact that there are different implementations of the object based on browsers and their versions requires you to write browser-sensitive code when instantiating it from script. Listing 1.1 uses a technique called *object detection* to determine which XMLHttpRequest object is available.

```
var xmlHttp = null;

if (window.XMLHttpRequest) { // IE7, Mozilla, Safari, Opera, etc.

xmlHttp = new XMLHttpRequest();

} else if (window.ActiveXObject) {

try{

xmlHttp = new ActiveXObject("Microsoft.XMLHTTP"); //IE 5.x, 6

}

catch(e) {}

}
```

Now that the object has been instantiated, you can use it to make an asynchronous request to a server resource. To keeps things simple, you can make a request to another page called Welcome.htm (listing 1.2).

```
<html xmlns="http://www.w3.org/1999/xhtml" >

<head>

<title>Welcome</title>

</head>ting 1.2 Welcome.htm

<body>

<div>Welcome to ASP.NET AJAX!</div>

</body>
```

```
</html>
```

Welcome.htm is pretty minimal and contains some static text welcoming you to the book. You make the asynchronous request with a few more lines of code that you wrap in a function called sendRequest ( listing 1.3).

```
function sendRequest(url) {
if (xmlHttp) {
xmlHttp.open("GET", url, true); // true = async
xmlHttp.onreadystatechange = onCallback;
xmlHttp.setRequestHeader('Content-type',
'application/x-www-form-urlencoded');
xmlHttp.send(null);
}
}
```

The sendRequest method takes as a parameter the URL to which you'll be making an HTTP request. Next, it opens a connection with the asynchronous flag set to true. After the connection is initialized, it assigns the onreadystatechange property of the XMLHttpRequest object to a local function called onCallback. Remember, this will be an asynchronous call, which means you don't know when it will return. A callback function is given so you can be notified when the request is complete or its status has been updated. After specifying the content type in the request header, you call the send method to transmit the HTTP request to the server.

When the status of the request changes and the callback function is invoked, the final step is to check the status and update the user interface with the contentsreturned from Welcome.htm ( listing 1.4).

```
function onCallback() {
if (xmlHttp.readyState == 4) {
if (xmlHttp.status == 200){
var r = document.getElementById('results');
r.innerHTML = xmlHttp.responseText;
}
else {
alert('Error: ' + xmlHttp.status);
}
}
}
```

The status of the request is returned in the readyState property of the XMLHttpRequest object. The value 4 indicates that the request has completed. Next, the response from the server must be checked to confirm that everything was successful. Status code 200 is designated in the HTTP protocol to indicate that a request has succeeded. Finally, the innerHTML of a span element is updated to reflect the contents in the response

# About JSON

In addition to the XMLHttpRequest object and XML, a third major technology often used for Ajax applications is JavaScript Object Notation (JSON, http://www.json.org/). With JSON, JavaScript objects or data can be persisted (serialized) in a short and easily understandable way, without requiring a lot of JavaScript code to either write or read the data (also true for XML). JSON makes use of a previously oftenoverlooked feature of JavaScript, or more accurately, of the ECMAScript language

specification, also known as ECMA-262. JSON is used internally by current versions of ASP.NET AJAX and generally can be used to exchange complex data with a server. This allows JavaScript to understand

it, and it helps avoid the sometimes cumbersome parsing process of XML. The following code uses JSON to define a book object:

```
{"book": {
"title": "Programming ASP.NET AJAX",
"author": "Christian Wenz",
"chapters": {
"chapter": [
{"number": "1", "title": "Introduction"},
{"number": "2", "title": "JavaScript"},
{"number": "3", "title": "Ajax"}
]
}
}}
```

This is the same data that you saw defined using XML earlier in this chapter. The

object with the book property contains title, author, and chapters properties.

The chapters property contains several chapter subelements, each with a number and

a title property. This can be best visualized when looking at it as XML data.

```
<book title="Programming ASP.NET AJAX" author="Christian Wenz">
<chapters>
<chapter number="1" title="Introduction" />
<chapter number="2" title="JavaScript" />
<chapter number="3" title="Ajax" />
</chapters>
</book>
```

*Simple example for Using JSON*

```
<head>
<title>JSON</title>
</head>
<body>
<script language="JavaScript" type="text/javascript">
var json = '{"book": { "title": "Programming ASP.NET AJAX", "author": "Christian
```

```
Wenz","chapters": {"chapter": [ {"number": "1", "title": "Introduction"}, {"number": "2",
"title": "JavaScript"}, {"number": "3", "title": "Ajax"} ]} }}';
var obj = eval("(" + json + ")");
for (var i=0; i < obj.book.chapters.chapter.length; i++) {
document.write(
"<p>" +
obj.book.chapters.chapter[i].number +
": " +
obj.book.chapters.chapter[i].title +
"</p>"
);
}
</script>
</body>
</html>
```

# Installing ASP.NET AJAX in VS 2005

AJAX enabled web application is a new feature added with visual studio 2008 but with VS 2005 you will need to setup ASP.NET AJAX that is integrated directly into the IDE. On the ASP.NET AJAX home page (http://ajax.asp.net), you can finda link to ASP.NET AJAX itself in the form of an MSI installer package named ASPAJAXExtSetup.msi. Look for the Microsoft ASP.NET 2.0 AJAX Extensions 1.0.

# Dealing with ASP.NET AJAX Main Items

## ScriptManager

### the brains of an Ajax page

The ScriptManager control is considered the brains of an Ajax-enabled page and is by far the most important control in the framework. As we move along in this chapter and throughout the book, we'll demonstrate how to leverage the ScriptManager and reveal its intricacies. The important thing to understand at thispoint is that, as the name suggests, this control is responsible for many of the operations that take place during an Ajax application.

Because you want this control to be present on all the pages of the site, you place it in the master page of the web application rather than in the home page (or content page):

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

You place it in the master page so that any content pages that inherit from it receive the same functionality. This is generally a good practice for similar controls that are used across multiple content pages. Furthermore, this invisible control must be declared *before* all other Ajax-enabled server controls in the page hierarchy to

ensure that they're loaded and initialized accordingly.

Even though the ScriptManager control isn't declared in the content page, you can easily retrieve an instance of it by calling its static method GetCurrentand passing in the current Page instance:

ScriptManager scriptManager = ScriptManager.GetCurrent(this.Page);

With this instance, you can manage and configure the way the errors, scripts, and other settings on the page behave. We'll explore some of this in a moment; first, let's see what adding the ScriptManager to the page does to the application.

## Understanding the ScriptManager

The primary responsibility of the ScriptManager is to deliver scripts to the browser. The scripts it deploys can originate from the ASP.NET AJAX library—embedded resources in the System.Web.Extensions.dll, local files on the server, or embedded resources in other assemblies. By default, adding the control to the page, declaratively or programmatically, delivers the required scripts you need for Ajax functionality on the page. To see the evidence, right-click the home page from the browser, and select the View Source option (or select View > Source in IE, or View > Page-Source in Firefox). In the viewed source window, search for an occurrence of ScriptResource.axd

<script src="/04/ScriptResource.axd?d=zQoixCVkx8JK9a1Az_4OOriP7

iw9S-TvBA24ugyHeZ8NSIfT6_bRe7yPttgsOhCr1ud1jBUWNQa9KSAugqepLY7DN4cuXzH5ybztCger

rk1&t=633141075498906250"

type="text/javascript">

</script>

Let's decode what this tag means; this is at the core of how scripts are delivered to

the client.

In ASP.NET 2.0, resources embedded in an assembly are accessed through the WebResource.axd HTTP handler. In the ASP.NET AJAX framework, a new HTTP handler called ScriptResource.axd replaces it with some additional functionality for localization and browser compression. Previous script shows a reference to a script assigned by the ScriptManager that is eventually downloaded by the new handler.

What about the cryptic text? How does the browser decipher it, and what does it mean? A closer look exposes two parameters: d and t. They assist the browser in identifying and caching the resource. The first is the encoded resource key, assigned to the d parameter. The second is the timestamp, t, that signifies the last

modification made to the assembly (for example, t=632962425253593750). When the page is loaded a second time, the browser recognizes the parameters and spares the user the download by using what's in its cache to retrieve the resources.

NOTE Embedding resources in an assembly is a common technique for controls

and libraries that require resources like images and scripts. This

approach simplifies how controls are packaged and deployed.

## Dynamically Assigning ASP.NET AJAX Script References

In most scenarios, the easiest way to add a script file to an ASP.NET page is in markup, as in the following example:

```
<asp:ScriptManager ID="SMgr" runat="server">
  <Scripts>
    <asp:ScriptReference Path="./Script.js" />
  </Scripts>
</asp:ScriptManager>
```

However, it is also possible to add script references dynamically.

1 -

```
ScriptManager Smgr = ScriptManager.GetCurrent(Page);
 if (Smgr == null) throw new Exception("ScriptManager not found.");
ScriptReference SRef = new ScriptReference();
```

2 -

```
// If you know that Smgr.ScriptPath is correct...
SRef.Name = "Script.js";
// Or, to specify an app-relative path...
SRef.Path = "~/Scripts/Script.js";
```

3 -

If the script is part of an assembly, set the Name and Assembly properties of the ScriptReference instance.

```
SRef.Name = "Script.js";
SRef.Assembly = "ScriptAssembly";
```

4 –

Specify whether to run debug or release versions of the script. To set this mode for all scripts on the page, set the ScriptMode property of the ScriptManager control. To set debug mode for an individual script, set the ScriptMode property of the ScriptReference object.

```
// To set ScriptMode for all scripts on the page...
Smgr.ScriptMode = ScriptMode.Release;


//Or, to set the ScriptMode just for the one script...
SRef.ScriptMode = ScriptMode.Debug;


//If they conflict, the setting on the ScriptReference wins.
```

5 –

```
Smgr.Scripts.Add(SRef);
```

# ASP.Net AJAX Toolkit

The Ajax Control Toolkit is an open source project that Microsoft started in the early days of ASP.NET AJAX. It's a collection of extenders, script controls, and client components written with the Microsoft Ajax Library.

## Installing the Control Toolkit for VS 2005

Before you can use ASP.NET AJAX controls, you need to add the Toolkit controls to your development environment. You can download it from the ASP.NET AJAX home page at http://ajax.asp.net/toolkit/default.aspx?tabid=47. Up-to-date documentation can be found at http://ajax.asp.net/ajaxtoolkit. The toolkit is hosted on CodePlex web site (http://www.codeplex.com/AtlasControlToolkit/) and is provided in the form of a ZIP archive. Actually, two archives: one contains the toolkit plus source code; the other, smaller archive, does not come with the sources.

## AJAX with UpdatePanels And Web Services

The UpdatePanel is an Ajax-enabled server control that works closely with the ScriptManager to apply partial-page updates to a page. also it the most obviously control in toolkit and as I think that it most used one by developers, where UpdatePanle still consume server resources then we must use UpdatePanle carefully and in special cases like refreshing  GridView or other data controls, but if you plan to to get small data from server then calling web service by AJAX is better

# Invoking web service methods from JavaScript

The first step in Ajax-enabling a page is to add the ScriptManager control. Remember, the ScriptManager is the brains of an Ajax page because its responsibilities primarily include managing and deploying scripts to the browser. In this case, you want to leverage the ScriptManager so the page can use the web service

proxy you just generated.

1-

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
<Services>
<asp:ServiceReference Path="~/StarbucksService.asmx"
InlineScript="true" />
</Services>
</asp:ScriptManager>
```

2-

```
<div>
<input id="Location" type="text" />
<input id="GetNumLocations" type="button" value="Get Count"
onclick="getLocations()" />
<div id="NumLocations"></div>
</div>
```

3-

```
function getLocations(){
var zip = $get("Location").value;
AspNetAjaxInAction.StarbucksService.GetLocationCount(zip,
onGetLocationSuccess,
onGetLocationFailure,
"<%= DateTime.Now %>");
}
```

# Invoke ASPX Page Methods

An interesting feature in ASP.NET AJAX is the ability to call, from JavaScript, methods that are declared in

the ASP.NET page itself. Because these methods are declared on a page, not from a Web Service, they're appropriately called page methods. To demonstrate how this works, let's add a simple static method called HelloEmployee to the page. This method takes as a parameter an instance of the Employee class you created earlier. The method returns to the caller a formatted greeting:

```
[WebMethod]
public static string HelloEmployee(AspNetAjaxInAction.Employee emp)
{
return string.Format("Hello {0} {1}.", emp.First, emp.Last);
}
```

Notice how the method is decorated with the WebMethod attribute (defined in the System.Web.Services namespace), similar to public methods in a Web Service.

This required attribute must be adorned on any methods you want to expose as a page method. In the .aspx page, you enable support for these types of methods by setting the **EnablePageMethods** property of the ScriptManager to True. By default, this setting isn't enabled, and any static web methods on the page are omitted from the

web service proxy:

```
<asp:ScriptManager ID="ScriptManager1" runat="server"
EnablePageMethods="True">
<Services>
<asp:ServiceReference Path="StarbucksService.asmx"
InlineScript="true" />
</Services>
</asp:ScriptManager>
```

To complete this example, you need to call the method from JavaScript and process the response

# Working with the DOM Using AJAX Client library

The Microsoft Ajax Library lets you access the DOM in a manner independent from the browser that renders the page. The abstraction API consists of the methods exposed by two client classes: Sys.UI.DomElement and Sys.UI.DomEvent. The first one abstracts a DOM element, and the second represents the event data

object that DOM event handlers receive as an argument.

The following is Shortcut Methods used for accessing AJAX library

$get, Sys.UI.DomElement.getElementById Returns a reference to aDOM element

$addHandler, Sys.UI.DomElement.addHandler Adds an event handler toan event exposed by a DOM

element

$removeHandler, Sys.UI.DomElement.removeHandler Removes an event handler added with $addHandler

$addHandlers, Sys.UI.DomElement.addHandlers Adds multiple event handlers to events exposed byDOM elements and wraps the handlers with delegates

$removeHandlers, Sys.UI.DomElement.removeHandlers Removes all the handlers added with $addHandler

and $addHandlers

# References and Links

Books:

ASP.NET AJAX IN ACTION

Programming.ASP.NET.AJAX (OReilly)

URLs:

http://www.asp.net/AJAX/Documentation/Live/default.aspx

http://encosia.com/

***Hisham Mohammad El-Breky***

***Senior Software Developer***