AJAX Guidance

Priyanka Poman, Atul Gupta

UX Group, Product Competency Center
Microsoft Technology Centre,
Infosys Technologies Ltd.

Date	Version	Authorized By
31st December 2007	v 1.1	Atul Gupta

Overview

Web development has undergone major changes in recent years. Having started as plain HTML, to web applications using classic ASP, to CGI to ASP.NET to AJAX to Silverlight, it isn't the same anymore. See Figure 1 below to see how the technologies have evolved. Gone are the days when customers were told that a web application will have fewer UI features as compared to thick client applications. Customers these days are demanding much more and the technologies are scaling up to provide the necessary framework, platform and toolset to build such Rich Internet Applications (**RIAs**).
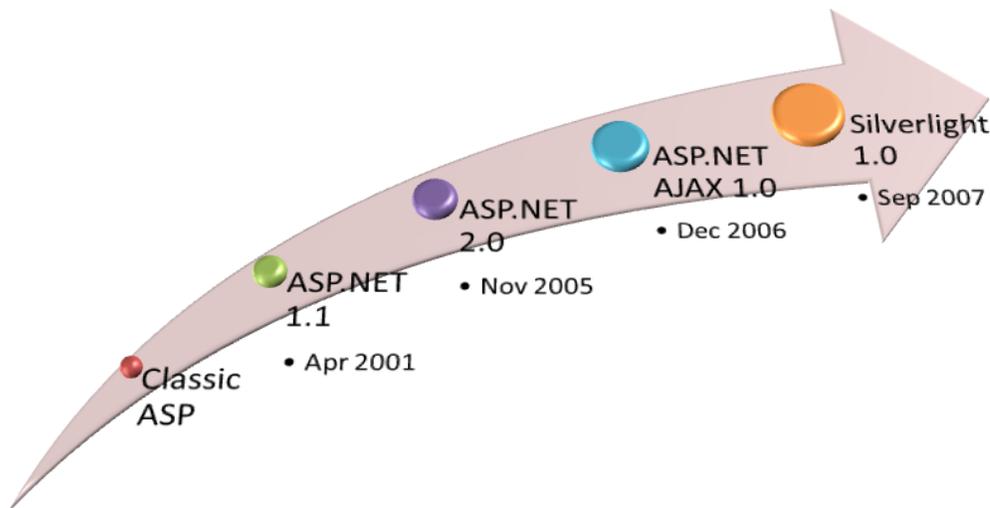

Figure 1 Evolution of UI Technologies on MS Platform

While these technologies have provided immense capabilities for developers to explore and provide **RIAs**, there is also the other side of the story. Without clearly understanding how these technologies impact the performance of a web application, what is planned to be a major overhaul of your application to improve on the UI, could well fall flat with degraded performance.

This whitepaper compares the performance of ASP.NET enabled site with an ASP.NET AJAX enabled site. The idea of this paper is to highlight to the reader the possible performance implications about selecting one approach over another. The paper by no means undermines the capabilities, but only tries to act as a guide for the developers to conduct their own experiments and make well informed decisions to select technology for their solution.

Objective

ASP.NET AJAX gives us the advantage of enhanced user experience (using script manager, various controls and update panel that helps in partial page rendering). This significantly improves the website's user experience, since it provides flicker free pages and faster page updates.

The rich control library of ASP.NET AJAX makes website-development very easy. Developers can drag-and-drop the required controls in the UpdatePanel container and AJAX control, and run the application with partial page rendering. All the necessary plumbing and wiring is managed in the background by the Script manager control provided by AJAX.

However, does it really reduce the application's bandwidth utilization? Does it really help boost the performance of the application or is it notional? We decided to dig deeper into this and conducted an experiment to determine the expected increase or decrease in application performance when using these controls.

Tools and Software Used

Visual Studio 2005 Professional Edition
Development platform for building the ASP.NET AJAX based application.

ASP.NET AJAX Control Toolkit
The control toolkit contains host of AJAX controls that can be used in an ASP.NET application to bring in AJAX functionality. We used the ASP.NET AJAX version 1.0 control toolkit. You can download the toolkit from ASP.NET AJAX site (http://www.asp.net/ajax/downloads/). We rely mainly on two important components of AJAX namely ScriptManager and UpdatePanel.

ScriptManager
The ScriptManager manages client scripts for AJAX pages. It also supports features like partial-page rendering and Web-service calls. By default, it registers the script for the Microsoft AJAX Library with the page.

UpdatePanel
The UpdatePanel is one of the most important controls of AJAX library. It enables you to refresh partial page in an ASP.NET application instead of full page post back.

ASP.NET AJAX Futures
Enhanced capabilities added to AJAX available out-of-box with Visual Studio 2008 and as an add-on to Visual Studio 2005. The December 2006 CTP can be downloaded from MS Download site (http://www.microsoft.com/downloads/details.aspx?FamilyId=29B6B62B-A25A-498D-AA14-90B68CF2E392&displaylang=en)

Not only can we invoke server side web service, but we can also as easily invoke server side windows communication foundation (WCF) service. ASP.NET 2.0 supports callbacks using which you can achieve similar functionality, without having to write a separate web service. However, using the AJAX Script Manager, the functionality can be built and integrated lot more easily.

Visual Studio 2008 also provides enhanced support of client side debugging and display of generated java script.

HTTP Fiddler

Fiddler (http://www.fiddler2.com/Fiddler2/version.asp) is a tool that intercepts the HTTP traffic between your computer and the Internet. Fiddler allows you to inspect all HTTP Traffic, set breakpoints, and "fiddle" with incoming or outgoing data. Fiddler includes a powerful event-based scripting subsystem, and can be extended using any .NET language. You can use the Session Inspector tab and then view either the raw or formatted text (via Text view) tabs to see what information is flowing over the wire when a web page is invoked.

Note: Fiddler 2.0 requires version.NET Framework 2.0. When using Fiddler to intercept traffic on local machine, ensure that the web site isn't invoked using "localhost", but with the machine name i.e. as http://<machinename>/<appname>/<page>.

Test Cases

The following sub-sections describe the details of the scenario and the tools we used to conduct the experiment. Since our intention was primarily to observe the HTTP traffic and not really focus much on actual response times, we ran the tests on a single machine. We decided to compare three different ways to achieve this same functionality

1. Do a regular page post back to get the necessary data and display on the page
2. Use ASP.NET AJAX UpdatePanel to minimize on the flicker and page post back
3. Use ASP.NET AJAX capability to invoke a server side web service to get the necessary data

The third option is a functionality offered by ASP.NET AJAX Futures. This is described a bit more in detail below.

Scenario 1

This scenario included testing a page with the below listed controls. The view state on all of the following controls was enabled as per their default settings:

1. A regular **DataGrid** with basic formatting for alternative rows that is populated on **Pageload** event. It displays details from the **Employee** table in **Pubs** database. This is to primarily create some content on the page, similar to real-time scenarios.
2. A textbox to display server results (server date time) and a button that caused full page post back
3. An UpdatePanel with textbox to display server results (server date time) and a button to submit the contents of UpdatePanel
4. A textbox to display server results and a button to invoke server side web service that returns server date time.

   In all three button invocations, the result obtained from the server was the server date and time

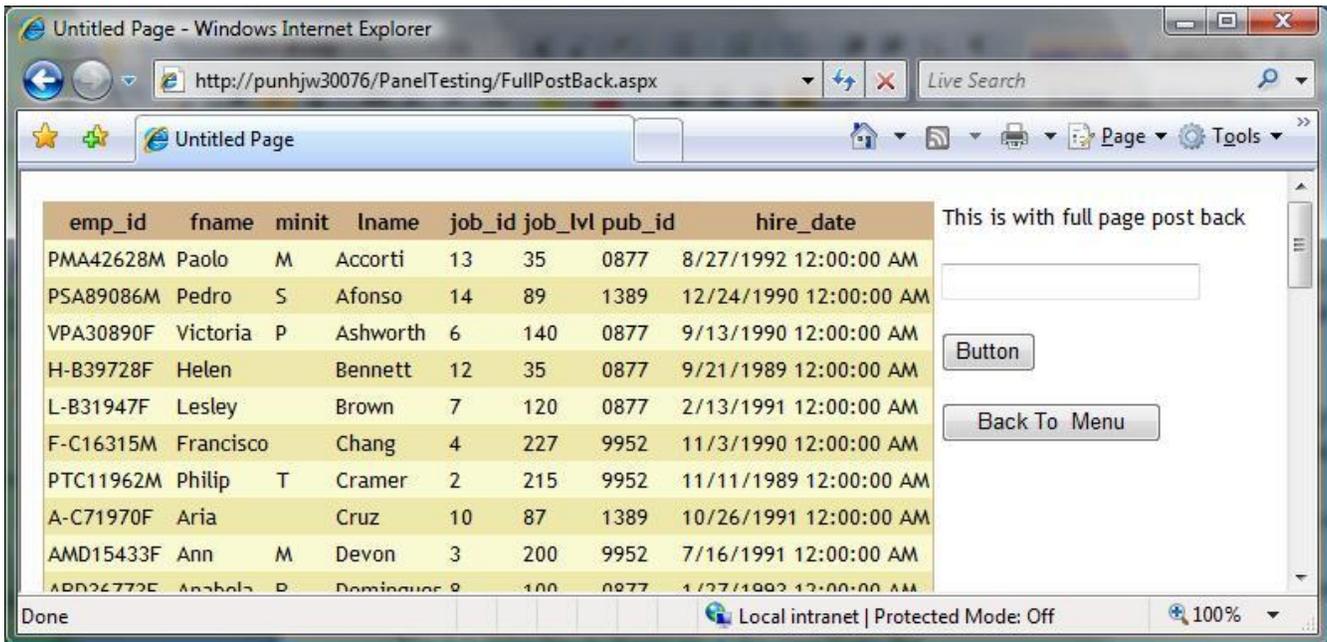The following figure shows what the page looks like.

Figure 2 Full Page Post back scenario with fetching Server time

Scenario 2

This scenario is similar to **Scenario 1** with only view state of the various controls disabled. We set the page view state to false rather than trying to disable view state for individual controls.

Scenario 3

This scenario included testing a page with the below listed controls. The view state on all of the following controls was enabled as per their default settings:

1. A textbox to enter search string and a button to submit the page for server processing. The search results were displayed in a **DataGrid** (with basic formatting)
2. An UpdatePanel providing same functionality as full **Pagepost**. The textbox, button and the **DataGrid**, however, were inside the UpdatePanel.
3. A textbox and button combination to invoke web service. The results obtained were displayed by dynamically creating a table using client-side JavaScript code.

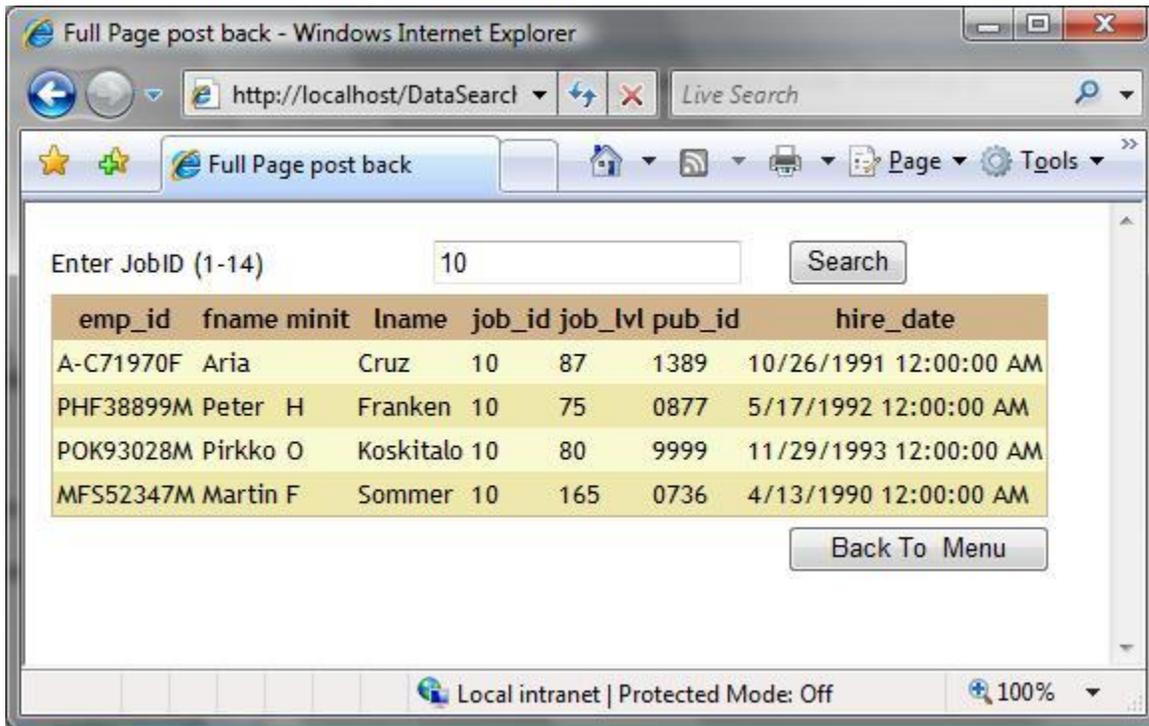The following figure shows what the page looks like.

Figure 3 Full Page Post back Scenario with getting search results

Scenario 4

This scenario was essentially the same as **Scenario 3** with only view state of the various controls disabled. We set the page view state to false rather than trying to disable view state for individual controls.

Note: The code snippets are provided in the **APPENDIX - Code Snippets**. *Do, however, note that the code is provided as is and without any warranty. The code is not a demonstration of best coding practices, but written only to explore the HTTP traffic.*

Results

For each search request, we tracked the bytes being sent to the server from the client browser and the bytes received back in response.

Note: The values listed below are based on the scenarios used, and depend on the controls and their view state settings. They will hence differ from scenario to scenario. The results are only meant to indicate the differences between the three approaches.

Scenario 1 Results

Sr.	Method	Sent Bytes (client to server)	% Saving	Received Bytes (server to client)	% Saving
1	Full Page Post	10,621		18,656	

| 2 | Using UpdatePanel | 10,455 | 01.56 % | 10,612 | 43.12 % |
| 3 | Using Web Service | 548 | 94.85 % | 253 | 98.64 % |

Scenario 2 Results

Sr.	Method	Sent Bytes (client to server)	% Saving	Received Bytes (server to client)	% Saving
1	Full Page Post	1,042		9,087	
2	Using UpdatePanel	900	13.62 %	1,044	88.51 %
3	Using Web Service	548	47.41 %	253	97.22 %

Scenario 3 Results

Sr.	Method	Sent Bytes (client to server)	% Saving	Received Bytes (server to client)	% Saving
1	Full Page Post	1,016		4,565	
2	Using UpdatePanel	874	13.98 %	4,359	04.51 %
3	Using Web Service	502	50.59 %	2,148	52.95 %

Results from Scenario 4

Sr.	Method	Sent Bytes (client to server)	% Saving	Received Bytes (server to client)	% Saving
1	Full Page Post	998		2,921	
2	Using UpdatePanel	832	16.63 %	2,697	07.67 %
3	Using Web Service	502	49.70 %	2,148	26.46 %

Interpretation of Results

The results are in line with the expectations. The number of bytes sent or received is maximum in case of full page post back and least in case of web service calls. The values are much larger for full page post back and UpdatePanel if the controls' view state is enabled. The presence or absence of view state does not impact the data sent in case of invoking the web service.

We observed that when using UpdatePanel, the data flowing from client to server is only marginally lesser than a full page post back. This is interesting since the server-side page processing is similar to a regular post back, while using the **Pageload** and other events. The bytes sent by the server in this case essentially represent the HTML required to update the UpdatePanel, hence depends on the UpdatePanel contents. Since the data returned is specific to contents of UpdatePanel, the savings will depend on the amount of data in the UpdatePanel vis-à-vis the rest of the page.

The Web Service option gives varied percentage savings, which is entirely based on the scenario. It does, however, clearly indicate that when using data that can be easily bound to controls on the client-side, it is better to use the Web Service option. Using web service increases the development effort since the data obtained back

needs to be programmatically assigned to the controls for display. In case the UI requirement is complex, UpdatePanel is still useful (though with higher data over the wire).

Note: We will reiterate here that the numbers and percentages here are only indicative. You will have to conduct tests for your scenario to get the appropriate numbers.

Recommendations

Based on the above results, it is pretty obvious that invoking services using ASP.NET AJAX gives the least amount bandwidth usage and hence should give maximum performance. Performance will definitely depend on the server side logic, network bandwidth, number of users using the application etc, but these other factors would remain consistent across any of the implementation options.

UpdatePanel is a simple, easy, and quick way to convert an existing ASP.NET application to more responsive and flicker-free page rendering. The saving you get on number of bytes on using UpdatePanel is proportional to the difference in data on the entire page versus the data within UpdatePanel. As an extreme case, you can also include the entire page content within an UpdatePanel to give user a flicker-free experience, but in such cases you will not save on the performance of page rendering.

If you are developing a new application you may definitely want to explore the option of invoking web services from client code, to ensure better performance, for your application. However, you have to write more code to extract the service response values and update the page content from client-side **Javascript**. There is no extra effort involved if you were using UpdatePanel.

Additionally, in case of a web service the call is not returned to the same page (similar to UpdatePanel). You cannot use this approach in scenarios where you need to execute code on the server based on the same page, since you may say be accessing some page-specific variables. For such cases UpdatePanel is not very useful. You can instead use ASP.NET Callback. Session variables can be accessed in the web service code by editing the web service method appropriately - [WebMethod(EnableSession=true)].

References

1. UpdatePanel tips and tricks from Wicked Code column in MSDN Magazine, June 2007 (http://msdn.microsoft.com/msdnmag/issues/07/06/WickedCode/)

2. ScriptManager Control Overview, ASP.NET AJAX Documentation (http://www.asp.net/ajax/documentation/live/overview/ScriptManagerOverview.aspx)

3. UpdatePanel Control Overview, ASP.NET AJAX Documentation (http://asp.net/ajax/documentation/live/overview/UpdatePanelOverview.aspx)

4. Script callback in ASP.NET from Cutting Edge column in MSDN Magazine, August 2004 (http://msdn.microsoft.com/msdnmag/issues/04/08/cuttingedge/)

5. Custom Script callbacks in ASP.NET from Cutting Edge column in MSDN Magazine, January 2005 (http://msdn.microsoft.com/msdnmag/issues/05/01/CuttingEdge/).

APPENDIX – Code Snippets

The code used for the scenarios is zipped and available for reference at the following location.

Note that code for Scenario 1 and 2 is the same except for the view state setting in the ASPX pages (EnableViewState="false"). You need to keep these as true/false as per needs.

http://mtc/MTCPCC/UITech/Shared%20Documents/UpdatePanelGuidanceScenario1-2.zip

Similarly for the Scenario 3 and 4, the only difference is the view state setting that you can alter in the @Page tag of the ASPX pages.

http://mtc/MTCPCC/UITech/Shared%20Documents/UpdatePanelGuidanceScenario3-4.zip