Syncfusion®

Deliver innovation with ease®

Ignore ASP.NET MVC at your own peril

Lessons learned from the trenches

CONTENTS

What is the end goal?

As users, we enjoy the experience offered by certain websites more than others. Some of our favorite sites, considering just the experience perspective, include stackoverflow.com (implemented with ASP.NET MVC), Google maps and basecamphq.com (implemented with Ruby on Rails—RoR).

Consider common attributes of these sites.

- Clean look and feel
- Solid performance
- Great Ajax experience
- Solid cross-browser compatibility – use the browser of your choice and things work as expected

What does it take to provide a user experience similar to the experience provided by these sites? This document examines several aspects that are of interest when attempting such a task using the two web-development frameworks from Microsoft: Web Forms[1] and ASP.NET MVC[2]. Each aspect considered has a section on how Web Forms and ASP.NET MVC fare in that area.

The premise of this document is that ASP.NET MVC makes it much easier to create user experiences such as those implemented by stackoverflow.com. Our position on ASP.NET MVC should be no surprise given the title of this document. Many ASP.NET MVC advocates take a rather timid approach when it comes to recommending ASP.NET MVC over the Web Forms framework. As you will see, we are not at all shy about our preference for ASP.NET MVC. We strongly believe ASP.NET MVC is better, and a better fit, for **every** web-development need. We aim to convince even the most die-hard fan of Web Forms to take a close look at ASP.NET MVC.

Requirements

We assume some familiarity with ASP.NET MVC (and Web Forms). Exposure to one of the numerous getting started tutorials on the Web should be all that is required.

Why should you even consider our opinion?

At Syncfusion we have had extensive experience with both frameworks. We have produced Web Forms controls for several years. We were the first component vendor to offer ASP.NET MVC custom controls. We ran our website (www.syncfusion.com) using the ASP.NET Web Forms framework for several years. When ASP.NET MVC 1 was released, we switched to ASP.NET MVC and have since maintained our website on this platform.

More importantly, we have supported thousands of customers on both platforms for a wide variety of application needs. We have been in the trenches with them and therefore believe that we have a considered opinion on key aspects to be considered for the implementation of successful Web applications.

[1] Whenever we say Web Forms, we refer to the forms framework built over the ASP.NET runtime and not the runtime itself. The runtime itself is excellent and is shared with ASP.NET MVC. Features such as caching, security, managed code use and master pages are built on the same ASP.NET runtime in both frameworks.
[2] Version 2 unless indicated otherwise.

Ajax – the straw that broke the web forms abstraction model

Abstraction of the HTTP protocol is at the heart of Web Forms while shunning such abstraction is at the heart of ASP.NET MVC. Let us consider which approach works better in the real world.

Web Forms

With Web Forms, Microsoft worked very hard to abstract the stateless nature of the HTTP protocol. The idea was to make web development as similar as possible to Rapid Application Development (RAD) rich-client development.

Consider the implementation of server-side events. The Web Forms state model enables server-side events. Without this plumbing, when a server-side control on a web page is initialized and sent to the client, the server no longer has any idea what the value was originally set to. If data is changed on the client-side, the changed value is posted back. The server has no idea that the value changed, making it impossible to trigger server-side events such as TextChanged. Web Forms accomplished this state model by serializing state (ViewState). We have seen instances where the improper use of ViewState cost megabytes in additional traffic, but overall this abstraction worked well for most applications – until Ajax came along.

Ajax and the server-side lifecycle

Ajax allowed for the selective refresh of sections of a Web page. With Web Forms, even when the user interface refresh was selective, the actual cycle of server-side post-back and server lifecycle events was required to be complete since the server-side abstraction model depended on it. UpdatePanels, introduced to Ajax-enabled Web Forms, essentially triggered a full-scale post-back under the covers. Data transferred and processed on the server were pretty much the same as with a full post-back. This was certainly not the most optimal way to implement Ajax and provided for a poor user experience.

ASP.NET MVC

With ASP.NET MVC there is no real abstraction of the details. HTTP is a stateless protocol. ASP.NET MVC forces us to deal with this reality from day one. State can certainly be maintained in sessions, hidden fields and caches. There is however no attempt to make these decisions on our behalf. We choose to serialize the state we need. This is very powerful. Ajax calls are almost trivial to implement. They perform exactly as advertised and can be selectively processed on the server. When something breaks it is much easier to debug. The building blocks do not have a complex abstraction. We can readily observe what is happening and make changes.

Separation of concerns – User Interface code and Business Logic

The Model-View-Controller pattern has been around for over three decades. It has been adapted for use in several user-interface frameworks. It is a very simple pattern that advocates the clear separation of responsibilities between the model (data), controller (logic & orchestration) and the view (presentation). It has been observed to help produce more maintainable, structured applications. Web Forms and ASP.NET MVC differ radically in their adherence to separation of concerns best practices.

Web Forms

A key issue with Web Forms is that the controller and view (and at times even the model) are implemented together in the Web Forms page (or server-side control). This tied key parts of the application's logic into the view, making the application hard to maintain (or test) over time.

It is technically possible to have a clear separation between the view and business logic with Web Forms. However, such a separation requires extra awareness and work on the developer's part[3]. The framework does not mandate it in any way. On the contrary, drag-and-drop rapid application development encourages a designer-centric approach, which makes it very easy to quickly add code that makes things work but with no consideration whatsoever for the separation of concerns. How else can we explain the presence of components such as the SqlDataSource? These components make it trivial to write code that mixes all layers of an application – code that leads to maintenance nightmares.

All in the view

```
<asp:GridView

    ID="GridView1"

    run at="server"

    DataSourceID="SqlDataSource1"

…

</asp:GridView>

<asp:SqlDataSource

    ID="SqlDataSource1"

    runat="server"

    ConnectionString =

     "<%$ ConnectionStrings:ConnectionString" %>"

    SelectCommand = "Select * from Customers" <asp:SqlDataSource>
```

ASP.NET MVC

ASP.NET MVC and other frameworks such as Ruby on Rails (RoR) and Spring take the separation between the View and the Controller very seriously. The convention with these frameworks is that the controller orchestrates interaction between the model and the view. There is no coupling between the view and the model or the view and the controller.

The ASP.NET MVC framework bakes this separation of responsibilities into the framework. Incoming URLs are handled by a routing framework that then instantiates the correct controller instance. The controller then orchestrates data access as applicable and then passes information back to the framework for presentation. The framework then completes the final step by acting on the instructions (display a view, redirect to another action, etc.) provided by the controller. Views in ASP.NET MVC contain little to no logic. Code-behind, while supported, is discouraged (Visual Studio does not even create a code-behind file by default). This system leads to a very clean separation between data, business logic and the presentation, which leads to tangible long-term maintenance benefits. Such a system is also very testable.

[3] The Web Client Software Factory from Microsoft provides guidance in this area – http://msdn.microsoft.com/library/bb264518.aspx.

jQuery & client-side scripting

In the early days of Web development, client-side script was mostly relegated to tasks such as validation. With the advent of Ajax this changed and a lot of complex JavaScript was written to create AJAX-enabled Web applications. This was hard work. Client-side scripting has a reputation for being quirky and hard to get right. IDE support for the JavaScript language was lacking until recently. The language itself, while very capable, was not understood very well. The DOM as implemented by different browsers caused several compatibility issues and generally earned its name as a difficult area to get right.

Over time, frameworks emerged to make this work easier. One such framework that emerged, and has quickly become the de facto standard for authoring client-side functionality, is the jQuery[4] framework. jQuery makes it quite easy to work with client-side script. It abstracts away differences between common browser platforms and offers an elegant CSS-like syntax for targeting elements of the DOM. This, coupled with its fluid interface, has made client-side programming a joy.

The Web Forms framework and the ASP.NET MVC framework differ substantially in their ability to work seamlessly with jQuery.

Web Forms

Web Forms applications can use jQuery but not in a very optimal manner. Web Forms controls act as self-contained units and produce an extensive amount of intricate markup (presentation and behavior code is often inline). Container controls also implement an interface called INamingContainer, which translates into some really fancy client-side element IDs. This makes it very hard to target such elements in a predictable manner from jQuery. Consider the following output from one of our Web Forms grid samples. Notice the complex and hard-to-pin-down nature of the generated DOM element IDs.

```
<div id="ctl00_PagePlaceholder_GridGroupingControl1">

      <input type="hidden" id="ctl00_PagePlaceholder_GridGroupingControl1CurrentRow"
name="ctl00_PagePlaceholder_GridGroupingControl1CurrentRow"
value="ctl00_PagePlaceholder_GridGroupingControl1~TR~0~_TOPGROUP_^48*R0" />
```

ASP.NET MVC

With ASP.NET MVC, developers have complete control over the markup. This makes it the norm to have HTML that is jQuery friendly. Consider the following output from one of our ASP.NET MVC grid samples. This markup can be easily targeted using a jQuery selector (class selector $('.RowCell') for instance).

```
<td class="RowCell">10030</td><td class="RowCell">1</td><td class="RowCell">LILAS</td>
```

[4] www.jquery.com

Unobtrusive JavaScript

Another aspect to client-side programming that is now widely seen as required practice is the use of Unobtrusive JavaScript[5]. Unobtrusive JavaScript helps tackle browser inconsistencies. It allows for scaling functionality to the client being targeted. It also allows for easier management of the view (including styling with CSS).

Web Forms

With Web Forms applications there is little control over the markup produced by an application. Most server-side controls are, as mentioned earlier, self-contained units of content, presentation and behavior. It is certainly possible for Web Forms controls to be written with Unobtrusive JavaScript principles taken into account. Unfortunately, this is not the case with the vast majority of controls available for Web Forms.

ASP.NET MVC

ASP.NET MVC applications and controls adhere to the practice of making JavaScript code unobtrusive. For instance, a DateTimePicker would be rendered as a TextBox by default. If JavaScript is enabled on the client, it would then be wired to behave as a DateTimePicker on the client. If not, it will function as a TextBox, which will still allow the user to enter dates. Controls written for the ASP.NET MVC framework (typically) respect this model and attach behavior on the client-side, keeping the model clearly separate from the content and presentation.

Performance

The ASP.NET MVC Framework allows for very lightweight markup. The application of Unobtrusive JavaScript principles also leads to less markup being produced since it avoids repetitive inline markup such as that used for event subscriptions. Any state maintained by applications is targeted, and usually a small fraction of ViewState on similar Web Forms pages will be targeted.

Performance differences are even more pronounced with Ajax applications. With Web Forms, the entire page lifecycle with all its associated traffic/code is required. This leads to much higher traffic back-and-forth with the server.

Consider the following traffic reports from an HTTP traffic diagnostics utility – Fiddler[6] tracking two similar applications constructed using Web Forms and ASP.NET MVC.

	Web Forms: Paging implemented with Ajax using the Syncfusion Web Forms Grid control	ASP.NET MVC: Paging implemented with Ajax using the Syncfusion ASP.NET MVC grid control
User interface		
Data sent	17.5 kB	1.5 kB

[5] If you are not familiar with Unobtrusive JavaScript, please refer to Appendix A for a quick summary.
[6] http://www.fiddler2.com

Data received	50.1 kB	1.4 kB
Actual data received	**Abbreviated version – complete data is about 15 pages long** 19459|updatePanel|ctl00_PagePlaceholder_GridPanel| <table id="GridTable"> <tr> <td> <div id="ctl00_PagePlaceholder_GridGroupingControl1"> <input type="hidden" id="ctl00_PagePlaceholder_GridGroupingControl1CurrentRow" name="ctl00_PagePlaceholder_GridGroupingControl1CurrentRow" value="ctl00_PagePlaceholder_GridGroupingControl1~TR~0~_TOPGROUP_^48*R0" />	**Complete data received** [{"UniversityCode":10037,"Title":"Enterprise Computing","Duration":90,"CourseFees":1500,"CGPA":9.75},{"UniversityCode":10038,"Title":"Mobile Computing","Duration":45,"CourseFees":1250,"CGPA":9.66},{"UniversityCode":10039,"Title":"WAP and XML","Duration":60,"CourseFees":1000,"CGPA":8.33},{"UniversityCode":10040,"Title":"Design Patterns","Duration":75,"CourseFees":1500,"CGPA":8.66},{"UniversityCode":10041,"Title":"Distributed Component Architecture","Duration":90,"CourseFees":2000,"CGPA":7.52},{"UniversityCode":10042,"Title":"Data Structures","Duration":60,"CourseFees":1000,"CGPA":9.55},{"UniversityCode":10043,"Title":"Neural Networks","Duration":75,"CourseFees":1750,"CGPA":9.03},{"UniversityCode":10044,"Title":"Genetic Algorithms","Duration":90,"CourseFees":2000,"CGPA":8.91},{"UniversityCode":10045,"Title":"Grid Computing","Duration":30,"CourseFees":1000,"CGPA":9.55},{"UniversityCode":10046,"Title":"Cloud Computing","Duration":60,"CourseFees":2500,"CGPA":9.87},{"UniversityCode":10047,"Title":"Enterprise Computing","Duration":90,"CourseFees":1500,"CGPA":9.75},{"UniversityCode":10048,"Title":"Mobile Computing","Duration":45,"CourseFees":1250,"CGPA":9.66}]

The ASP.NET MVC application has no need to produce an elaborate amount of markup to cater to a complicated server-side abstraction. In this case, only JSON (http://www.json.org/) data is transmitted back from the server on paging. The amount of data transmitted back from the server is 50 KB with the Web Forms solution versus 1.5 KB for the ASP.NET MVC solution.

Browser compatibility

Internet Explorer no longer commands the 90+% market share that it did back in 2003. Today, the numbers are more along the lines seen below, making it very important to ensure that Web applications are browser agnostic.

- Internet Explorer – 60%
- Firefox – 25%
- Google Chrome – 7%

Web Forms

It is entirely possible that a future generation of Web Forms controls will be implemented using jQuery or similar frameworks. Controls available on the market today are mostly written using vendor-specific JavaScript libraries that offer varied support across browsers. If browser compatibility is on your list of concerns, Web Forms is not a great solution.

ASP.NET MVC

ASP.NET MVC applications and controls typically implement their client-side functionality using jQuery. This leads to solid browser compatibility. In general, all modern browsers should work well with ASP.NET MVC applications and controls.

Testing

Solid and easy testing of business logic and data access is considered to be critical to the development of maintainable Web applications.

Web Forms

Web Forms applications are very hard to test. This is primarily due to the reasons below.

- It remains hard to automate Web Forms applications without running them inside a complete hosting environment. It is not possible to easily mock a complex hosting environment such as IIS.
- The view often contains business logic and presentation, making it very hard to test the business logic without testing the presentation.

In practice this meant that Web Forms applications were usually tested using elaborate UI automation tools or manual testing – an error-prone and expensive process.

ASP.NET MVC

ASP.NET MVC makes testing easy, even fun. All of the core components of ASP.NET MVC are designed to easily run from any unit testing container. Consider the following test that validates the business logic of a simple calculation engine. The test is trivial to set up and run. It takes no time to execute and verifies that the business logic of the system works as expected. The view does nothing more than display the returned result.

```
[TestMethod]
  public void Calculator_returns_correct_values_with_single_digit_numbers()
  {
     // Arrange
      HomeController controller = new HomeController();
      var vm = new Models.CalculatorViewModel();
      vm.Operands = new Models.Operands();
      vm.Operands.One = 7;
      vm.Operands.Two = 9;
      // Act
```

```
            ActionResult result = controller.Calculate(vm);

            // Assert

            Assert.IsNotNull(result);

            Assert.IsInstanceOfType(result, typeof(RedirectToRouteResult));

            Assert.IsNotNull(controller.TempData["vm"]);

            Assert.IsInstanceOfType(controller.TempData["vm"],
typeof(Models.CalculatorViewModel));

            var vmResult = controller.TempData["vm"] as Models.CalculatorViewModel;

            Assert.AreEqual(vmResult.Result, 16);

        }
```

Search engine optimization (SEO)

Wikipedia defines SEO, in part, as the process of improving the volume or quality of traffic to a Web site. There are many, many aspects to this science (or art if you prefer). There is much room for discussion on what works and what does not. One aspect that everyone agrees on has to do with friendly URLs. URLs such as this http://www.mysite.com/products/toys/fancy-water-cannon are much friendlier than, for instance, http://www.mysite.com/products.aspx?pid=201. Friendly URLs fare well with search engines since these URLs allow the search engine to correlate terms that appear in the URL with other aspects being considered to determine ranking.

It is also generally agreed that URLs that are "hackable" – URLs where the user is able to remove or add parts and thereby navigate the site – are preferred. For instance, removing "fancy-water-cannon" from the above URL should lead to a page that lists all available toys.

Web Forms

It is possible to formulate friendly URLs with Web Forms. This is not, however, the norm. The norm is to deal with ASP.NET pages as physical resources with URLs reflecting the path to these resources within the server.

ASP.NET MVC

ASP.NET MVC treats action methods as page content and completely hides physical page details. The routing system is very elegant and tightly woven into the framework. It allows for the easy creation of friendly and "hackable" URLs.

The creation of rich user interfaces

One of the major criticisms of ASP.NET MVC that we have seen has to do with the availability of rich prepackaged user interface elements. Surely, this is one area where Web Forms excels, right? Read on and you may be surprised.

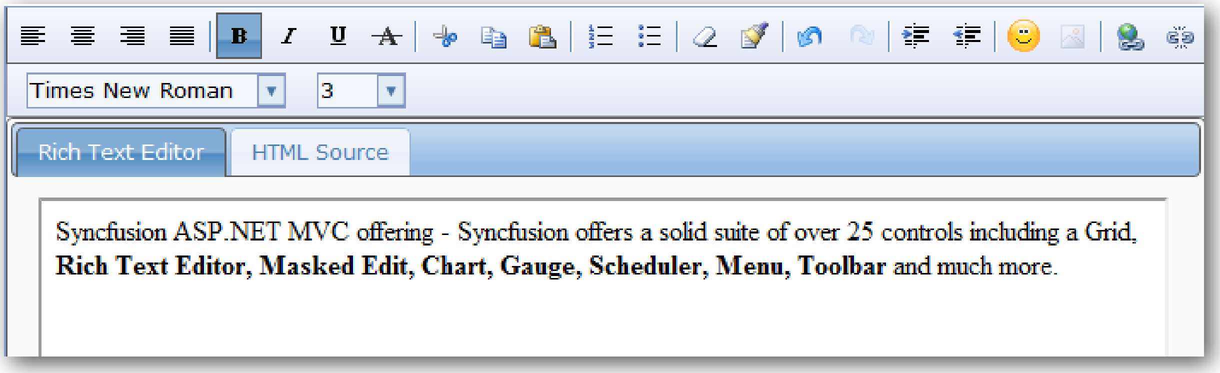
Web Forms

Web Forms has a wide variety of controls available from Microsoft and numerous third party vendors. They make it easy to add a rich and polished user interface to your applications.

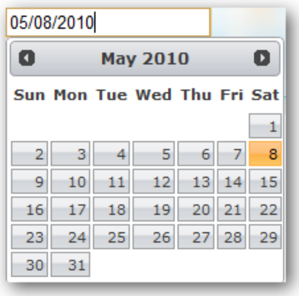
ASP.NET MVC

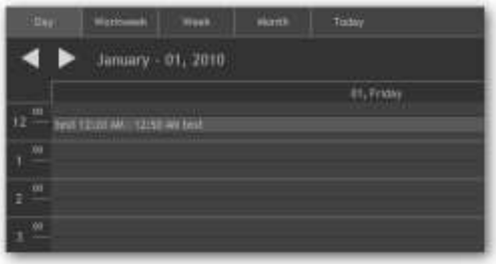
It is a common misconception that ASP.NET MVC does not offer a rich set of user-interface elements. Microsoft provides a limited set of helpers out of the box, but a solid third-party control market is taking root. In the past year, a number of vendors have introduced rich user-interface elements for this market. Every major component vendor has a currently shipping suite of controls or has one in the works.

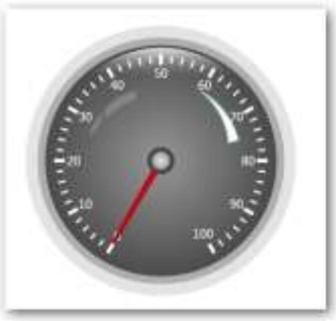
At Syncfusion, we have been shipping a complete set of controls. We offer rich menus, editors, grids, charts, an Outlook-like scheduler, gauges and much more. A few representative control screenshots are displayed below.



Syncfusion ASP.NET MVC offering - Syncfusion offers a solid suite of over 25 controls including a Grid, **Rich Text Editor, Masked Edit, Chart, Gauge, Scheduler, Menu, Toolbar** and much more.







In all, Syncfusion offers over 25 controls for the ASP.NET MVC platform with many more shipping in the coming weeks. These controls are easy to consume in ASP.NET MVC and provide for a very polished UI while retaining the benefits of the ASP.NET MVC platform.

With Visual Studio 2010, ASP.NET MVC is tightly integrated into the IDE. Visual Studio's excellent scaffolding support for ASP.NET MVC allows for a very productive experience. It takes only a few clicks to create a fully functional, strongly typed view for an action method. ASP.NET MVC's convention-over-configuration approach makes it very easy to get started. You can certainly build great looking user interfaces with minimal effort when working with ASP.NET MVC.

Conclusion

We hope that the topics considered in this document have been of interest. With ASP.NET MVC you can truly venture to build websites that offer extremely compelling experiences such as those offered by stackoverflow.com.

Action items

- Consider the ASP.NET MVC 2 platform for your Web development needs
- Download a suite of ASP.NET controls from your favorite vendor and put them to the test
- Beat your competition to the punch and be the first in your industry to switch to ASP.NET MVC!

There is simply no excuse to be messing with ViewState anymore.

About Syncfusion

Syncfusion, Inc. is a leading provider of enterprise-class software components and tools for the Microsoft .NET platform. With Syncfusion, developers can move beyond simply coding applications to delivering business innovation—the elegant user interfaces, business intelligence dashboards, and sophisticated reporting that today's business users need, in the formats they demand. Syncfusion's award-winning .NET components and controls are designed to grow with you, whether you're using Windows Forms, WPF, ASP.NET, ASP.NET MVC, or Silverlight. For questions about products, pricing, and licensing, please contact sales@syncfusion.com or call 1.888.9DOTNET (US only) or +1 919.481.1974.

Syncfusion’s ASP.NET MVC offering: a solid suite of over 25 controls including grids, rich text editors, masked edits, charts, gauges, schedulers, menus, toolbars and much more.

Online Demo: http://samples.syncfusion.com/sfmvcsamplebrowser/8.2.0.18/MVC/tools_MVC/samples/3.5

Product Page: http://www.syncfusion.com/products/user-interface-edition/aspnet-mvc/Tools

APPENDIX A

Obtrusive & Unobtrusive JavaScript

This involves the separation of client-side-scripting based functionality from a Web page's structure, content (HTML) and presentation (CSS). For instance, consider the two examples below.

Obtrusive JavaScript – event subscriptions are embedded in the markup

```
<script type="text/javascript">

  function sayHello(){

        alert('Hello!');

}

</script>

</head>

<body>

<input name="btn" id="btn" type="button" value="Hello!" onclick="sayHello();" />

</body>

</html>
```

Unobtrusive JavaScript (uses jQuery) – event subscriptions are made at run time

```
  $(document).ready(function() {

      $("#btn").click(function(){

                alert('Hello from jQuery!');}

);

// button has no even wire up code

<body>

<input name="btn" id="btn" type="button" value="Hello from jQuery!" />

</body>
```

Unobtrusive JavaScript helps tackle browser inconsistencies. It allows for scaling functionality to the client being targeted. It also allows for easier management of the view (including styling with CSS).