



Technical Notes

Leveraging ASP.NET MVC for Easy Mobile Versions of a Website

Version 1.0

February 12, 2009

Jason Doucette
Technology Director, LGG Media
Jason@lggmedia.com

Contents

Terms of Use	3
Revision History.....	3
Introduction	4
The Code	4
Basic Setup	4
Changing the View.....	6
Demo	8
Next Steps/Other Uses	8
Feedback.....	8

Terms of Use

The concepts and source code provided in this document and any applicable example files may be freely used for whatever purpose your little heart desires. Except for crime. Don't use our stuff for evil.

The document text itself may be redistributed freely in its original unmodified form for noncommercial purposes. For other forms of publication, please contact the author.

LGG Media Inc and this paper's author will not be held responsible or liable for anything that happens as a result of using the information contained herein. If millions die as a result of a (mis)application of these concepts, it's on you. If you use this information and get a zillion dollar raise, that's all you too (but be cool about it and send us a beer, eh.)

We really don't know everything, and as such, feedback, questions, and suggestions are encouraged and can be directed at the author, Jason Doucette, at Jason@lggmedia.com. Please be kind and reference this document in any correspondence so we don't just assume you're a crazy person.

About LGG Media

LGG Media is a Canadian company that relentlessly gathers and aggregates entertainment data for use in a zillion client projects ranging from concert alert systems to enhanced radio players to movie and restaurant review sites. Our data services are made available as an API to enhance existing projects and we also partner with amazing designers to deliver world class online and mobile experiences.

From a technical perspective, we're always looking forward to the latest tools and techniques. While we focus on Microsoft technologies, we pay close attention to the patterns and practices emerging from other development communities and enjoy leveraging best practices such as continuous integration, test driven development, domain driven design, and rapid iterative development.

For more information, please visit our website at <http://www.lggmedia.com>

Revision History

Version	Date	Author	Notes
1.0	February 11, 2009	Jason Doucette	Initial version

Introduction

Coding versions of a website for different devices can be a painful experience. While the design process can be a fairly straightforward creation of additional mockups in Photoshop, there are a number of technical issues that must be overcome to deliver a solution that's easy to maintain over time.

Using Microsoft's ASP.NET MVC framework, it's possible to leverage the built-in separation of the presentation layer to relatively easily support multiple devices with the majority of the site's code staying unchanged and non-repeated.

The Code

Basic Setup

In this example, we're making the best thing ever: a list of websites. Yep, a bookmark list. We've got a huge database of sites with their names, categories, and web and mobile URLs, and we want to display them on a page.

In our first iteration, we're only concerned with the initial website. Mobile hasn't entered the picture yet; we just want to make a page that show a list of sites.

We'll do this with a new controller called SkinDemo:

Listing 1.0: The initial SkinDemoController

```
public class SkinDemoController : Controller
{
    /// <summary>
    /// Simple test data structure...
    /// </summary>
    public class Website
    {
        public string Name { get; set; }
        public string Category { get; set; }
        public string Url { get; set; }
        public string MobileUrl { get; set; }
    }

    public ActionResult Index()
    {
        ViewData.Model = GenerateResult();
        return View();
    }

    /// <summary>
    /// Simple test data generator to build a result
    /// set for the Index action
    /// </summary>
    /// <returns>A List of Websites</returns>
    private static List<Website> GenerateResult()
    {
        return new List<Website>
        {
            new Website {
                Name = "Digg",
                Category = "News",
                Url = "http://www.digg.com",
                MobileUrl = "http://m.digg.com"},
            new Website {
                Name = "Twitter",
                Category = "Social",
                Url = "http://www.twitter.com",
                MobileUrl = "http://m.twitter.com"},
            new Website {
                Name = "Google",
                Category = "Search",
                Url = "http://www.google.com",
                MobileUrl = "http://m.google.com"}
        };
    }
}
```

As you can see in Listing 1.0, we've got a pretty simple controller set up that returns the default view. We're strongly typing our view models here (more on that in another Technical Note someday), but the code to render a table of websites is pretty straightforward – at its core, it's just a foreach loop building a table.

Thanks to the magic of MVC, all we need is a controller and a view and we've got our "site" up and running.

Changing the View

Of course, someone has to come along and complain about the lack of mobile support. Maybe it's someone from the .mobi council, and they've brought their +1 Hat of "ignoring the question of why their TLD isn't shorter" so you can't throw the usual rebuttal in their face.

So, we've got to make a new version of the site that displays a more concise view for mobile devices. The usual way to detect a mobile device is to sniff out the user agent (don't bother with .NET's built in `IsMobileDevice` property, it's pretty random), but how best to output the results? In classic ASP.NET webforms, or even classic ASP, you could go with a few basic options:

- 1) Copy the page code and contents to another URL entirely and optionally redirect/transfer the user based on their device.
- 2) Clutter the view code with if statements (or a multiview control, for example) and code to render the appropriate output based on the device.

In the first case, there's needless duplication of code. Every change made to the page's behavior means changing both pages, which scales horribly as more device-specific pages are added to the mix. In the second case, the page logic is kept in one place, but the corresponding HTML/WML is harder to manage and will generally confuse most editors, producing incorrect error messages and warnings within Visual Studio, for example

With the ASP.NET MVC framework, we can leverage the fact that the view is already isolated to provide a cleaner solution. Here's how.

First, we'll need to modify the `SkinDemoController` class to inherit `BaseController` instead of `Controller`. `BaseController` is a new class we'll invent to handle some extra things for us.

(Note: while it's a great practice to provide your own base controller class so you can easily extend your solution, be careful not to make a "God Class" containing every helper method you can think of!)

Listing 1.1: BaseController.cs

```
public class BaseController : Controller
{
    /// <summary>
    /// A new implementation of View() that redirects to a different
    /// skin based on the user agent. A real implementation
    /// would require support for the other overrides of View()...
    /// </summary>
    /// <returns>The appropriate view for the device</returns>
    protected new ViewResult View()
    {
        if (HttpContext != null)
        {
            string action = null;
            string controller = null;
            string userAgent = HttpContext.Request.UserAgent;

            if (RouteData != null)
            {
                action = RouteData.GetRequiredString("action");
                controller = RouteData.GetRequiredString("controller");
            }
            if (!(string.IsNullOrEmpty(action) ||
                string.IsNullOrEmpty(controller)))
            {
                if (userAgent.ToLower().Contains("iphone"))
                {
                    if (File.Exists(Server.MapPath("/Views/" +
                        controller + "/" + action + "_iphone.aspx")))
                    {
                        return base.View(action + "_iphone");
                    }
                    return base.View();
                }
                // Very simple checks, a real app would use a
                // library of user agent strings...
                if (userAgent.ToLower().Contains("midp") ||
                    userAgent.ToLower().Contains("mot-v551"))
                {
                    if (File.Exists(Server.MapPath("/Views/" +
                        controller + "/" + action + "_mobile.aspx")))
                    {
                        return base.View(action + "_mobile");
                    }
                    return base.View();
                }
            }
        }
        return base.View();
    }
}
```

Here we've provided a new implementation of the View() method that does a rudimentary check of the user agent string provided by the requesting device to figure out if we've got a web request, a mobile request, or an iPhone request. Due to the magic of demos, there's a bunch of stuff missing from this sample code, like a real user agent check (you'd want to consult against a much broader list of user agent strings, which would likely be managed in a separate class) and further implementations of View to handle the cases where a controller or action is supplied. You know, the stuff we call "work."

In the base controller, we're set up to see if there's an extension view set up with the original view's path and filename and a modifier like `_mobile` or `_iphone` on it. Again, in a production app, you might want to be a bit smarter about the view pathname resolution to follow the MVC search paths.

If a specialized view isn't available, we default to the standard web version. Your application might want to display a 404 page or do some other kind of error handling.

Demo

There's a sample of this code online at <http://labs.lggmedia.com/SkinDemo/Index.ftw>

It's set up to handle regular web views, iPhone views, and a WAP view if you've got a Motorola V551 phone. If you use Firefox and the User Agent Switcher plugin, you can get the basic idea without actually using a mobile device.

Next Steps/Other Uses

Other than the "in the real world, you'd have to do this" notes mentioned above, you could easily extend this technique to custom handle other specific devices. Within the view itself, assuming you were using a suitable library of device capabilities, the mobile views could be designed to show images scaled to the correct display width automagically, providing a consistent user experience across a wide variety of handsets.

In theory, this technique could be used for other language support, but in practice most sites would have enough data- and context-related translation that some other technique would be more appropriate.

Feedback

Questions and comments are welcome! Feel free to email Jason@lggmedia.com with your thoughts.