

AMV203: ASP.NET MVC for Web Forms Programmers

Paul Litwin

Fred Hutchinson Cancer Research Center /
Deep Training

paul@deeptesting.com

twitter.com/plitwin



Paul Litwin

- **Developer**

- Focus: ASP.NET, ASP, C#, SQL Server, Reporting Services
- MCSD
- Microsoft MVP
- Programmer Manager with Fred Hutchinson Cancer Research Center (Seattle)

- **Co-Founder and Senior Trainer**

- Deep Training
 - www.deeptraining.com

- **Conference Chair/Speaker**

- Chair, Microsoft ASP.NET Connections
- Member INETA Speakers Bureau

- **Author of over a dozen books & courses, including...**

- *AppDev SQL Server 2005 & 2008 Reporting Services Courses*
- *ASP.NET for Developers*
- *Access Cookbook*
- *Access 2002 Desktop/Enterprise Dev Handbook*



Slides & Samples Download

- You can download them from:
 - www.deeptraining.com/litwin

ASP.NET MVC for Web Forms

Agenda

- **What?**
 - What is ASP.NET MVC?
- **Why?**
 - Why can't I just use Web Forms?
 - Why use ASP.NET MVC?
- **How?**
 - How does it work?
 - Show me how to build an app using it

ASP.NET MVC

What?

ASP.NET MVC

- A way to build highly-testable apps on top of the ASP.NET Framework
- An alternative to using RAD Web Forms

MVC Design Pattern

- Conery, Hanselman, Haack, Guthrie define it as...
 - An architectural pattern used to separate an application into three main aspects
 - **Model**. Set of classes that represent data and business rules for how data can be changed and manipulated
 - **View**. Application's user interface
 - **Controller**. Set of classes that handles communication from user, overall application flow, and application-specific logic

ASP.NET MVC

Why?

What's Wrong with Web Forms?

- Nothing...but...
 - Web forms don't support as precise control over "separation of concerns"
 - Too much of an application ends up in the "code behind"
 - High level of abstraction
 - Not as "close to the metal" as some devs want
 - Web forms are not as well suited for testing
 - Testing the "code behind" code is difficult
 - Search engine optimization (SEO) issues
 - ASP.NET 4.0 addresses this

Why Separate Concerns?

- Makes your app code more...
 - testable
 - modifiable / refactorable
 - reusable
 - maintainable
 - scalable

What Web Forms & ASP.NET MVC Have in Common

- Both use .aspx pages
 - Both can use Master pages and user controls
 - though you can use a different view engine if you'd like in ASP.NET MVC (NHaml, Spark, Brail, NVelocity, *Razor*, etc.)
- Both can use any data access framework (ADO.NET, LINQ, Entity Framework, etc.)
- Both use Visual Studio
- Both built on ASP.NET
 - ASP.NET runtime, localization, HTML encoding, ...
- Both run on IIS

ASP.NET MVC

How?

Basic Tenets of ASP.NET MVC

- Convention over configuration
- Don't repeat yourself (DRY)
- Pluggability whenever possible
 - We will use in-the-box parts of ASP.NET MVC but almost everything (data access, testing framework, view engine, validation, etc. can be replaced)
- Try to be helpful, but get out of the developer's way

Review: Model, View, Controller

- **Model**

- Data and business rules

- **View**

- Application's UI. Will employ a view for each page that is displayed to users

- **Controller**

- Classes that are responsible for application workflow and logic

Besides the M, the V, and the C...

- **Routing**
 - Nice URLs are possible because of routing engine
 - Routes are stored in global.asax
- **Binding to data**
 - Default ModelBinder auto-magically converts form values and route data into data
 - Allows you to strongly type your pages & controller methods
- **Scaffolding Views**
 - ASP.NET MVC will automatically provide scaffolding to help you create your views
 - Helpful, but almost always will want to modify views it creates
- **Html Helpers**
 - Methods you can use in views to generate Html elements, usually bound to data

Building an ASP.NET MVC Application

- Remainder of talk will focus on building an end-to-end ASP.NET MVC application that can be used to display and edit product records from the Northwind database

Demo 1 – ASP.NET MVC App

- Create ASP.NET MVC Application
- Go ahead and let VS create test app

ASP.NET MVC Project Anatomy

- **Content**
 - Style sheets, images
- **Controllers**
 - *AccountController.cs*
 - *HomeController.cs*
- **Models**
- **Scripts**
 - JavaScript, jQuery
- **Views**
 - Account
 - Home
 - Shared
- **Global.asax**



Demo 1 – ASP.NET MVC App (continued)

- Remove following starter files
 - Controllers|HomeController.cs
 - All views in Views|Home folder
 - In Test app, Controllers|HomeControllerTest.cs

Building Models

- Model is the part of the ASP.NET MVC app that provides data access
 - You can build model using
 - ADO.NET
 - LINQ to SQL
 - Entity Framework
 - Nhibernate
 - ...

Demo 2 – Creating Basic Model

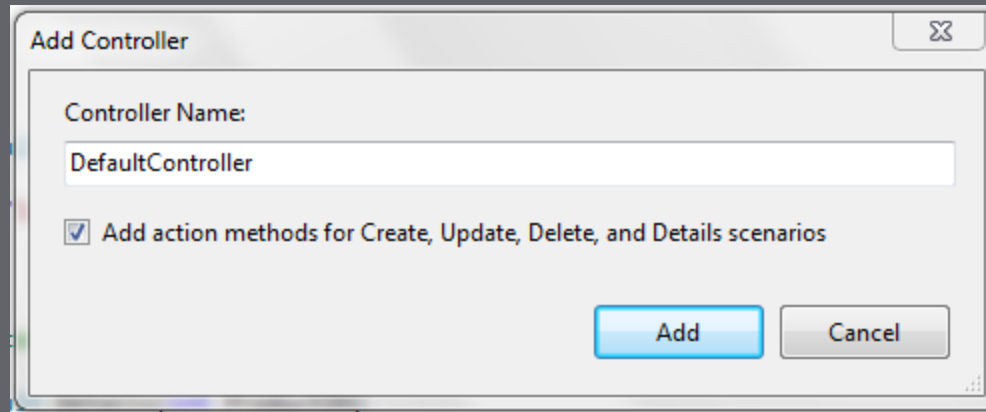
- We will use LINQ to SQL
- We will build a model on 3 tables from Northwind
 - Products
 - Change ProductID to id
 - Categories
 - Suppliers
- Set UpdateCheck property on Product fields to Never

Demo 3 – Adding Repositories to Model

- While we could have controller call LINQ to SQL classes, directly, it's better to provide another layer using a repository pattern
- We will create repository classes for each model object
 - Similar to creating a business tier in N-tier app design

Creating Controllers

- The intelligence of your application
 - Ferries data back & forth to model
 - Methods that eventually call views
 - Add Controller dialog can scaffold CRUD methods for you



Demo 4 – Creating Home Controller

- Create HomeController which will...
 - Be responsible for CRUD operations for Products
 - Replace form collection in post methods for (Create, Edit, Delete) with strongly-typed Product objects

Constructing Views

- Default view engine is the WebForms View Engine (the aspx page)
- The UI of the application

Demo 5 – Creating Views

- Scaffold our views by right-clicking on each of the HomeController methods and selecting **Add View**
 - Use strong-typed view
`ProductsApp.Models.Product`
- Need to fix up Create and Edit views
 - Remove ProductID from Create
 - Don't allow editing of ProductID in Edit
 - Replace Discontinued textbox with checkbox in Create & Edit

Demo 5EF – Swapping Out Model

- With Controller and Views built, let's now switch model to use Entity Framework
- Need to make a few changes to repository classes but otherwise, everything else should continue to work “as is”

Demo 6 – Adding Dropdown Support

- Add ProductFormModel class to HomeController
- Change Edit/Create methods to pass ProductFormModel to views
- Change type of Edit and Create views to derive from ProductFormModel class
- Replace each instance of “model.” with “model.Product.”
- Replace category and supplier Html.TextBoxFor helpers with Html.DropDownListFor helpers

Demo 7 – Adding Validation

- Add “Buddy” partial class for Products with data annotations
- Add `if (!ModelState.IsValid)` to Create and Edit post methods
- Adding client-side support requires a couple of additional steps

Refactoring App to Support Testing

- Currently, controller and model are tightly coupled; Can't test app without database dependency
- Can be fixed by using dependency injection
 - Extract interface
 - Update controller to support Constructor Injection
 - Create fake repository that also derives from interface

Demo 8 – Refactoring for Testing

- Right-click on **ProductRepository** class and select **Refactor|Extract Interface**
- Make interface public
- Modify **HomeController** with two constructors
 - Parameterless that creates new **ProductRepository**
 - Parameterized that allows passing in **IProductRepository**
- Add **FakeProductRepository** class
 - : **IProductRepository**
 - Right-click on interface and select **Implement Interface**

Constructing Tests

- For each unit test
 - Arrange
 - Arrange test, setting up any pre-requisites
 - Act
 - Act to test feature of application
 - Assert
 - Assert what you expect

Demo 9 – Creating & Running Unit Tests

- In Test project, right-click on **Controllers**, and select **Add New Test** and select **Unit Test**
- Create **CreateTestProducts** method
- Create tests using **FakeProductRepository**
- To run tests, right-click on **HomeControllerTest** class name and select **Run Tests**

ASP.NET MVC Pluses and Minuses

- **Pluses**

- A new way to build highly-testable apps on top of ASP.NET Framework
- No post-backs
- Requests map to methods rather than pages
- Right out of box, supports TDD and Unit Tests

- **Minuses**

- Not as RAD as using Web Forms
- Requires more coding
- No drag and drop experience
- Doesn't build on all your experience with ASP.NET server controls

A Hybrid Approach

- You can create hybrid web applications that combine best of ASP.NET Web Forms and ASP.NET MVC
 - Great for migrating existing Web Forms app to MVC
 - See slides at end of talk for a step-by-step approach to creating hybrid projects

What About Me?

- I am intrigued with ASP.NET MVC
- However...
 - My brain does hurt when reading about design patterns
 - I love Web Forms
- OTOH...
 - I like the idea of unit testing, maintainability, and best practices

Learning Resources

- Professional ASP.NET MVC 2.0
 - Galloway, Hanselman, Haack
- ASP.NET MVC 2 in Action
 - Palermo, Scheirman, Bogard, Hexter
- Pro ASP.NET MVC V2 Framework
 - Sanderson
- Head First Design Patterns
 - Freeman & Freeman
- Videos, Posts, & Articles on <http://asp.net/mvc>

Continuing the Discussion

Today

10:15am

AMV204: The Zen of ASP.NET and MVC

JAVIER LOZANO

11:45

AMV202: Building HTML Helpers for ASP.NET MVC

DINO ESPOSITO

4:30

AAR301: Guiding Principles for ASP.NET MVC Applications

SCOTT ALLEN

Tonight at 7:30-9:30

DevConnections Open Spaces

Tomorrow

9:30

AGN201: Learning TDD with ASP.NET MVC

SCOTT ALLEN

11:45

AMV301: Everything New for ASP.NET MVC

SCOTT ALLEN

2:30

AMV305: Oh ASP.NET MVC, How Extensible Art Thou?

JAVIER LOZANO



Slides & Samples Download

- You can download them from:
 - www.deeptraining.com/litwin

Your Feedback is Important

Please fill out a session evaluation form
drop it off at the conference registration
desk.

Thank you!

Postscript: Creating Hybrid Project in VS 2010 (1 of 4)

1. Add ref to System.Web.Mvc.dll - need to browse for it under C:\Program Files (x86)\Microsoft ASP.NET\ASP.NET MVC 2\Assemblies
2. Add ref to System.Web.Routing
3. Create an Empty MVC Project
 - a) Copy the Controllers, Models and Views directories into the Web Forms application
 - b) Copy the RegisterRoutes method in Global.asax to the Global.asax in the Web Forms project
 - c) Add RegisterRoutes(RouteTable.Routes); to the Application_Start in the Web Forms project
 - d) Add using System.Web.Mvc and using System.Web.Routing to Global.asax in the Web Forms project

Postscript: Creating Hybrid Project in VS 2010 (2 of 4)

4. Add the following to the <compilation> section of web.config

```
<assemblies>
```

```
  <add assembly="System.Web.Abstractions,  
    Version=4.0.0.0, Culture=neutral,  
    PublicKeyToken=31BF3856AD364E35" />
```

```
  <add assembly="System.Web.Routing,  
    Version=4.0.0.0, Culture=neutral,  
    PublicKeyToken=31BF3856AD364E35" />
```

```
  <add assembly="System.Web.Mvc, Version=2.0.0.0,  
    Culture=neutral,  
    PublicKeyToken=31BF3856AD364E35" />
```

```
</assemblies>
```

Postscript: Creating Hybrid Project in VS 2010 (3 of 4)

5. Add the following to the <pages> section of web.config

```
<namespaces>
```

```
  <add namespace="System.Web.Mvc" />
```

```
  <add namespace="System.Web.Mvc.Ajax" />
```

```
  <add namespace="System.Web.Mvc.Html" />
```

```
  <add namespace="System.Web.Routing" />
```

```
</namespaces>>
```

Postscript: Creating Hybrid Project in VS 2010 (4 of 4)

6. Enable MVC Tooling in Web Form Application

- a) Right-click on project and select unload
- b) Right-click on project and select Edit .csproj
- c) In <ProjectTypeGuids> section add at *beginning* of list of Guids: {F85E285D-A4E0-4152-9332-AB1D724D3325}
- d) Right-click on project and select reload

7. Close and reopen solution