Student Guide

Information in this document is subject to change without notice. Companies, names and data used in examples herein are fictitious unless otherwise noted. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Object Innovations.

Product and company names mentioned herein are the trademarks or registered trademarks of their respective owners.

Object Innovations® is a registered trademark of Object Innovations.

Authors: Robert Hurlbut and Robert J. Oberg

Copyright ©2011 Object Innovations Enterprises, LLC All rights reserved.

Object Innovations
877-558-7246
www.objectinnovations.com

Printed in the United States of America.
# Table of Contents (Overview)

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 1</td>
<td>Introduction to ASP.NET MVC</td>
</tr>
<tr>
<td>Chapter 2</td>
<td>Getting Started with ASP.NET MVC</td>
</tr>
<tr>
<td>Chapter 3</td>
<td>ASP.NET MVC Architecture</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>The Model</td>
</tr>
<tr>
<td>Chapter 5</td>
<td>The Controller</td>
</tr>
<tr>
<td>Chapter 6</td>
<td>The View</td>
</tr>
<tr>
<td>Chapter 7</td>
<td>Routing</td>
</tr>
<tr>
<td>Chapter 8</td>
<td>Unit Testing</td>
</tr>
<tr>
<td>Chapter 9</td>
<td>Security</td>
</tr>
<tr>
<td>Chapter 10</td>
<td>Combining ASP.NET MVC and Web Forms</td>
</tr>
<tr>
<td>Chapter 11</td>
<td>Deployment</td>
</tr>
<tr>
<td>Appendix A</td>
<td>Learning Resources</td>
</tr>
</tbody>
</table>
Directory Structure

- The course software installs to the root directory `C:\OIC\MvcCs`.
  - Example programs for each chapter are in named subdirectories of chapter directories `Chap02`, `Chap03`, and so on.
  - The Labs directory contains one subdirectory for each lab, named after the lab number. Starter code is frequently supplied, and answers are provided in the chapter directories.
  - The Demos directory is provided for doing in-class demonstrations led by the instructor.

- Data files install to the directory `C:\OIC\Data`. 
Table of Contents (Detailed)

Chapter 1: Introduction to ASP.NET MVC

Review of ASP.NET Web Forms ................................................................. 3
Advantages of ASP.NET Web Forms ......................................................... 4
Disadvantages of ASP.NET Web Forms .................................................... 5
Model-View-Controller Pattern ............................................................... 6
ASP.NET MVC ......................................................................................... 7
Advantages of ASP.NET MVC ............................................................... 8
Disadvantages of ASP.NET MVC ........................................................... 9
Goals of ASP.NET MVC ......................................................................... 10
Unit Testing ......................................................................................... 11
Summary .............................................................................................. 12

Chapter 2: Getting Started with ASP.NET MVC

An ASP.NET MVC 3 Testbed ................................................................. 15
Visual Studio ASP.NET MVC Project ..................................................... 16
New ASP.NET MVC 3 Project ............................................................... 17
Starter Application ............................................................................... 18
Simple App with Controller Only ........................................................ 19
Action Methods and Routing ............................................................... 25
Action Method Return Type ............................................................... 26
Rendering a View ............................................................................... 27
Creating a View in Visual Studio ......................................................... 28
The View Web Page ............................................................................... 29
Dynamic Output .................................................................................. 30
Embedded Scripts ............................................................................. 31
Embedded Script Example ................................................................. 32
Using a Model ..................................................................................... 33
Controller Using the Model ............................................................... 34
View Using the Model ......................................................................... 35
Strongly-Typed View ......................................................................... 36
Strongly-Typed View in Visual Studio ............................................... 37
Lab 2 .................................................................................................. 38
Summary .............................................................................................. 39

Chapter 3: ASP.NET MVC Architecture

The Controller in ASP.NET MVC .......................................................... 45
The View in ASP.NET MVC ................................................................. 46
The Model in ASP.NET MVC ............................................................... 47
How MVC Works ................................................................................ 48
Using Forms ...................................................................................... 49
HTML Helper Functions ..................................................................... 50
Displaying Sign In Form ................................................................... 51
Handling Form Submission ............................................................... 52
Chapter 4: The Model ........................................................................................................ 69
  Microsoft Technologies for the Model ................................................................. 71
  SmallPub Database ............................................................................................. 72
  ADO.NET ............................................................................................................. 74
  Language Integrated Query (LINQ) .................................................................... 75
  ADO.NET Entity Framework 4.1 ....................................................................... 76
  EDM Example ....................................................................................................... 77
  SmallPub Tables ..................................................................................................... 79
  SmallPub Entity Data Model .................................................................................. 80
  Entity Data Model Concepts .................................................................................. 81
  Querying the EDM ............................................................................................... 82
  Class Diagram ....................................................................................................... 83
  Context Class ........................................................................................................ 84
  List of Categories ................................................................................................. 85
  List of Books .......................................................................................................... 87
  LINQ to Entities Demo ......................................................................................... 89
  A LINQ Query ....................................................................................................... 90
  IntelliSense ........................................................................................................... 91
  Controller ............................................................................................................... 92
  Web.config ........................................................................................................... 93
  Using a Parameter ............................................................................................... 94
  Modifying a Data Source ..................................................................................... 95
  LINQ to Entities Insert Example ......................................................................... 97
  LINQ to Entities Delete Example ...................................................................... 98
  Entity Framework in a Class Library .................................................................... 99
  Data Access Class Library .................................................................................... 100
  Client Code ........................................................................................................... 101
  ASP.NET MVC Database Clients .......................................................................... 102
  ASP.NET MVC Database Client Demo ............................................................... 103
  Lab 4 .................................................................................................................... 109
  Scaffold in ASP.NET MVC 3 ............................................................................... 110
  NuGet Package Manager .................................................................................... 111
  Entity Framework 4.1 Code First ....................................................................... 112
  Scaffold and Code First Demo ........................................................................... 113
  A CRUD User Interface ....................................................................................... 114
  The Database ....................................................................................................... 116
  DbContext Class ................................................................................................... 118
Chapter 9: Security ................................................................. 271
Web Security ............................................................................. 273
Input Forgery .............................................................................. 274
Cross-Site Scripting ................................................................. 276
XSS Example .............................................................................. 278
Entering JavaScript ..................................................................... 280
The Attack .................................................................................. 281
Defending Against the Attack ...................................................... 282
Session Hijacking ....................................................................... 283
Cross-Site Request Forgery .......................................................... 285
XSRF Example ........................................................................... 287
Controller and Model Code ......................................................... 289
View Code .................................................................................. 290
Attacker Code ............................................................................ 291
The Defense .............................................................................. 292
SQL Injection .............................................................................. 293
Using the MVC Framework Securely ........................................... 297
Authorize Attribute ................................................................. 298
ChildActionOnly Attribute .......................................................... 299
RequireHttps Attribute ............................................................... 300
ValidateInput Attribute ............................................................... 301
Summary .................................................................................. 302

Chapter 10: Combining ASP.NET MVC and Web Forms ................. 303
Chapter 11: Deployment .......................................................... 323
  Server Requirements ......................................................... 325
  Continuous Integration ....................................................... 327
  Environment Configurations .............................................. 329
  Configuration File Transformation ...................................... 330
  MvcBuildViews for View Compile Checking ....................... 331
  Release Build View Compile Checking ............................... 332
  XCOPY .............................................................................. 333
  Web Deploy ........................................................................ 334
  Web Deploy Example ....................................................... 335
  Lab 11 .............................................................................. 342
  Summary ........................................................................... 343

Appendix A: Learning Resources ........................................ 347
Chapter 1

Introduction to ASP.NET MVC
Introduction to ASP.NET MVC

Objectives

After completing this unit you will be able to:

• Describe advantages and disadvantages of ASP.NET Web Forms.

• Understand the Model-View-Controller (MVC) pattern

• Outline the parts of an ASP.NET MVC application.

• Describe advantages and disadvantages of ASP.NET MVC.

• Understand the use of unit testing in creating ASP.NET MVC applications.
Review of ASP.NET Web Forms

• ASP.NET Web Forms provide a way to build web applications.

• You can use compiled, object-oriented languages with ASP.NET, including C# and Visual Basic.
  – All the power of the .NET Framework is available to you, including the extensive class library.

• Code and presentation elements can be cleanly separated.
  – Code can be provided in a separate section of a Web page from user interface elements.
  – The separation can be carried a step further by use of separate “code behind” files.

• ASP.NET Web Forms comes with an extensive set of server controls that provide significant functionality out of the box.

• Server controls transparently handle browser compatibility issues.
  – A special set of Mobile Controls can emit either HTML or WML, depending on the characteristics of the client device.

• Configuration is handled by XML files without need of any registry settings, and deployment can be done simply by copying files.
Advantages of ASP.NET Web Forms

- ASP.NET Web Forms continue to be supported and have their own advantages:
  - A rich event model supported in hundreds of server controls facilitates easy development of Web server applications, following a familiar GUI development paradigm.
  - View state makes it easy to manage state information.
  - The model works well for individuals and small teams doing rapid application development.
  - The large number of built-in and third-party components also facilitates rapid application development.

- In general, Web Forms are quite easy to work with and generally require less code.
Disadvantages of ASP.NET Web Forms

- Key disadvantages of ASP.NET Web Forms include:
  - ViewState tends to be large depending on the number of server controls contained on the page, thus increasing the size of the page and the length of the response time from server to browser.
  - ASP.NET Web Forms provide tight coupling with the code-behind classes which make automated testing of the back-end code apart from the web pages more difficult.
  - Because the code-behind classes are so tightly coupled to the web forms, developers are encouraged to mix presentation code with application logic in the same code-behind classes which can lead to fragile and unintelligible code.
  - Limited control of HTML rendered through use of server controls.
The Model-View-Controller (MVC) design pattern divides an application into three conceptual components:

- A **model** represents the data and operations that are meaningful to the domain of the application. It implements the application logic for the domain.

- **Views** display a user interface for portions of the model. Typically the UI is created from model data.

- **Controllers** handle incoming requests, work with the model, and select a view to render a UI back to the user.
ASP.NET MVC

- ASP.NET MVC is a framework based on ASP.NET for creating Web applications.
  - It is an alternative to Web Forms.

- ASP.NET MVC 1.0 is available as a separate download and can be installed on top of .NET 3.5 SP1 and Visual Studio 2008 SP.

- ASP.NET MVC 2.0 is integrated into .NET 4.0 and Visual Studio 2010.

- ASP.NET MVC 3.0 is a separate download and adds important new features, such as the Razor view engine.
  - When you install ASP.NET MVC 3 on top of Visual Studio 2010, you will have ASP.NET MVC 2 and ASP.NET MVC 3 both available side by side.

- ASP.NET MVC does not replace Web Forms but is an alternative approach to creating Web applications.

- It relies on the same ASP.NET infrastructure as does Web Forms and is integrated with ASP.NET features such as master pages and membership-based authentication.
Advantages of ASP.NET MVC

- **Key advantages of ASP.NET MVC include:**
  - The MVC pattern promotes *separation of concerns* into input logic (controller), business logic (model) and UI (view). This aids in managing complexity.
  - These components are loosely coupled, promoting parallel development.
  - This loose coupling also facilitates automated testing.
  - Views are created using standard HTML and cascading style sheets, giving the developer a high degree of control over the user interface.
  - There is no view state, reducing the load on the browser in rendering a page.

- **Separation of Concerns:**
  - Each component has one responsibility
  - SRP – Single Responsibility Principle
  - DRY – Don’t Repeat Yourself
  - More easily testable
  - Helps with concurrent development
Disadvantages of ASP.NET MVC

- **Key disadvantages of ASP.NET MVC include:**
  - Writing View contents the old ASP-like way (though there are new View engines that are coming out).
  - Unit testing and Test Driven Development (TDD) are encouraged and used more but also bring a steep learning curve.
  - Need to understand HTML controls and style sheets, but at the same time this allows a designer to work independently of the coders.
Goals of ASP.NET MVC

- The ASP.NET MVC Framework has the following goals:
  - Frictionless Testability
  - Tight control over markup
  - User/Search Engine friendly URLs
  - Leverage the benefits of ASP.NET
  - Conventions and Guidance

- Extensibility
  - Replace any component of the system
  - Interface-based architecture
  - Very few sealed methods / classes
Unit Testing

- Unit testing lets you specify the expected behavior of individual classes or other small code units in isolation.

- ASP.NET MVC encourages unit testing of the Models and the Controllers of the application to verify expected behaviors.

- Separation of concerns makes unit testing of individual components feasible.

- You may use Microsoft’s MSTest for unit testing or other unit test framework such as NUnit.
  
  - NUnit is free open source software and is used in this course. It can be used with the free version of Visual Studio 2010, Visual Web Developer Express 2010.

  - MSTest is built into Visual Studio 2010 Professional edition and higher.
Summary

- ASP.NET Web Forms is still used and has many advantages and disadvantages

- The Model-View-Controller (MVC) pattern is useful in creating applications that have separation of concerns

- ASP.NET MVC also has many advantages and disadvantages in developing web applications

- Unit testing is helpful and encouraged in developing ASP.NET MVC applications
Chapter 2

Getting Started with ASP.NET MVC
Getting Started with ASP.NET MVC

Objectives

After completing this unit you will be able to:

• Install the ASP.NET MVC application framework.

• Understand how ASP.NET MVC is used within Visual Studio.

• Create a Hello World application with ASP.NET MVC.

• Understand how Views are rendered.

• Understand how dynamic output works.
An ASP.NET MVC 3 Testbed

• This courses uses the following software:
  – Visual Studio 2010 Service Pack 1
  – ASP.NET MVC 3
  – SQL Server Express 2008 R2
  – SQL Server Compact 4.0

• Recommended operating system is Windows 7.

• To practice deployment on IIS, you should also have IIS installed, which should be done before installing Visual Studio 2010.

• The easiest way to get a complete installation of all the required software is to use Microsoft’s Web Platform Installer 3.0,
  – Navigate to http://www.microsoft.com/web/downloads/platform.aspx and download and install the Web Platform Installer
  – After running the Web Platform Installer, find Visual Web Developer Express 2010 SP1 and install it. This will install ASP.NET MVC 3 and other needed software as well.
  – You may also wish to install SQL Server 2008 R2 Management Studio Express, which can be done by the Web Platform Installer.
Visual Studio ASP.NET MVC Project

• Let’s use Visual Studio (or Visual Web Developer) to create an ASP.NET MVC 3 Web Application project:

  ![Image of Visual Studio new project dialog]

  • Click OK.
New ASP.NET MVC 3 Project

- Select Internet Application.

- A feature of MVC 3 is support of multiple view engines. Select ASPX.
  
  - The default for MVC 3 is the Razor View Engine, which we will discuss in Chapter 6.
  
  - Leave unchecked the checkboxes for HTML 5 markup and create a unit test project.
Starter Application

• Notice that there are separate folders for Controllers, Models and Views.

• Build and run this starter application:
Simple App with Controller Only

- To start learning how ASP.NET MVC works, let’s create a simple app with only a controller.

1. Create a new ASP.NET MVC 3 Web Application project with the name **MvcSimple** in the **Demos** folder.

2. This time choose the Empty Project Template.
Demo: Controller Only (Cont’d)

3. Right-click over the Controllers folder and choose Add | Controller from the context menu.

4. Provide the name **HomeController**

5. Under Scaffolding options choose Empty controller for the Template.

6. Click Add.
Demo: Controller Only (Cont’d)

7. Examine the generated code **HomeController.cs**.

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MvcSimple.Controllers
{
    public class HomeController : Controller
    {
        // GET: /Home/

        public ActionResult Index()
        {
            return View();
        }
    }
}
```
8. Replace the code for the `Index()` method by the following. Also, provide a similar `Foo()` method.

```csharp
public class HomeController : Controller
{
    //
    // GET: /Home/
    
    public string Index()
    {
        return "Hello from Index";
    }

    public string Foo()
    {
        return "Hello from Foo";
    }
}
```

9. Build and run.

10. Examine the URL Visual Studio used to invoke the application. (Naturally the port number varies.)

```
http://localhost:51148/
```
Demo: Controller Only (Cont’d)

11. Now try using these URLs\(^1\). You should get the same result.

http://localhost:51148/Home/
http://localhost:51148/Home/Index/

12. Now try this URL.

http://localhost:51148/Home/Foo/

You will see the second method `Foo()` invoked:

![Image of Hello from Foo]

13. Finally, let’s add a second controller `SecondController.cs`.

---

\(^1\) The trailing forward slash in these URLs is optional.
14. Provide the following code for the **Index()** method of the second controller.

```csharp
public class SecondController : Controller
{
    //
    // GET: /Second/
    
    public string Index()
    {
        return "Hello from second controller";
    }
}
```

15. You can invoke this second controller using either of these URLs:

```
http://localhost:51148/Second/
http://localhost:51148/Second/Index/
```

In either case we get the following result. The program at this point is saved in **MvcSimple\Controller** in the chapter folder.
Action Methods and Routing

- Every public method in a controller is an action method.
  - This means that the method can be invoked by some URL.

- The ASP.NET MVC routing mechanism determines how each URL is mapped onto particular controllers and actions.

- The default routing is specified in the Global.asax.cs file.

```csharp
routes.MapRoute(
    "Default", // Route name
    "{controller}/{action}/{id}", // URL with parameters
    new { controller = "Home", action = "Index", id = UrlParameter.Optional } // Parameter defaults
);
```

- Our example provides several examples of routing.

- If desired, additional route maps can be set up in Global.asax.cs.
Action Method Return Type

- An action method normally returns a result of type `ActionResult`.
  - An action method can return any type, such as `string`, `int`, and so on, but then the return value is wrapped in an `ActionResult`.

- The most common action of an action method is to call the `View()` helper method, which returns a result of type `ViewResult`, which derives from `ActionResult`.

- The table shows some of the important action result types, which all derive from `ActionResult`.

<table>
<thead>
<tr>
<th>Action Result</th>
<th>Helper Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ViewResult</td>
<td>View()</td>
<td>Renders a view as a Web page, typically HTML</td>
</tr>
<tr>
<td>RedirectResult</td>
<td>Redirect()</td>
<td>Redirects to another action method using its URL</td>
</tr>
<tr>
<td>JsonResult</td>
<td>Json()</td>
<td>Returns a serialized Json object</td>
</tr>
<tr>
<td>FileResult</td>
<td>File()</td>
<td>Returns binary data to write to the response</td>
</tr>
</tbody>
</table>
Rendering a View

• Our primitive controllers simply returned a text string to the browser.

• Normally, you will want an HTML page returned. This is done by rendering a view.
  – The controller will return a ViewResult using the helper method View().

```csharp
public ViewResult Index()
{
    return View();
}
```

• Try doing this in theMvcSimple program. Build and run.

Server Error in '/' Application.

The view 'Index' or its master was not found or no view engine supports the searched locations. The following locations were searched:
~/Views/Home/Index.aspx
~/Views/Home/Index.ascx
~/Views/Shared/Index.aspx
~/Views/Shared/Index.ascx
~/Views/Home/Index.cshhtml
~/Views/Home/Index.vbhtml
~/Views/Shared/Index.cshhtml
~/Views/Shared/Index.vbhtml
Creating a View in Visual Studio

- The error message is quite informative!
  - Let us create an appropriate file `Index.aspx` in the folder `Views/Home`.

- In Visual Studio you can create a view by right-clicking in the action method. Choose Add View.
  - Continue to use the ASPX view engine.
  - Clear the check box for master page and click Add.
The View Web Page

- A file Index.aspx is created in the Views\Home folder.
  - Edit this file to display a welcome message from the view. To make it stand out, use H2 format.
  - Note that this page inherits from the ViewPage class\(^2\).

```csharp

<!DOCTYPE html>
<html>
<head runat="server">
  <title>Index</title>
</head>
<body>
  <h2>Hello from the View</h2>
</body>
</html>
```

- Build and run.

\(^2\) The template parameter \(<\text{dynamic}\)> is new in .NET 4.0. Many of our programs won’t use it. Chapter 6 discusses this topic.
Dynamic Output

• The *ViewPage* class has a property *ViewData* that can be used for passing data from the controller to the view, enabling the rendering of dynamic output.
  
  – *ViewData* is of type *ViewDataDictionary*, which stores data named by string keys.

• This code in the controller stores the current time.

```csharp
public ViewResult Index()
{
    ViewData["time"] =
        DateTime.Now.ToLongTimeString();
    return View();
}
```

• This markup in the view page displays the data.

```html
<h2>Hello from the View</h2>
<%= ViewData["time"] %>
```

• Here is a run:

![Index - Windows Internet Explorer](img)

```html
Hello from the View
5:52:11 PM
```

– The program is saved in * MvcSimple\View*. 
Embedded Scripts

- The view page illustrates use of embedded C# script.
  - This feature is commonly used in ASP.NET MVC.

- There are two forms of embedded script.
  - The pair `<%= ... %>` delimits an expression which is converted to a string and outputted.
    `<%= ViewData["time"] %>`
  - The pair `<% ... %>` delimits programming statements that do not result in output.
    `<%
        int day = 0;
        while (day < 12)
        {
        %>`
Embedded Script Example

- See MvcSimple\Script.

```%
int day = 0;
int gifts = 0;
int total = 0;
while (day < 12)
{
    day += 1;
    gifts += day;
    total += gifts;
%
}
On day <%= day%> number of gifts = <%= gifts%><br/>
<% } %>
<p>Total number of gifts = <%= total %></p>
```
Using a Model

- Our next version of the program uses a model.
  - See MvcSimple\Model in the chapter folder.

- The model contains a class defining a Person.
  - See the file Person.cs in the Models folder of the project.
  - There are public properties Name and Age.
  - Unless otherwise assigned, Name is “John” and Age is 33.

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace MvcSimple.Models
{
    public class Person
    {
        public string Name { get; set; }
        public int Age { get; set; }
        public Person()
        {
            Name = "John";
            Age = 33;
        }
    }
}
```
Controller Using the Model

- The controller instantiates a *Person* object and passes it in *ViewData*.

  - Note that we need to import the *MvcSimple.Models* namespace.

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using MvcSimple.Models;

namespace MvcSimple.Controllers
{
    public class HomeController : Controller
    {
        //
        // GET: /Home/

        public ViewResult Index()
        {
            ViewData["person"] = new Person();
            return View();
        }
    }
}
```
View Using the Model

- The view displays the output using appropriate script.

  - Again we need to import the `MvcSimple.Models` namespace.

```csharp
<%@ Import Namespace="MvcSimple.Models" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 T...>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Model Demo</title>
</head>
<body>
    <%
    Person p = (Person)ViewData["person"];%
    
    <h2>Using model data:</h2>
    <p>Name = <%= p.Name %></p>
    <p>Age = <%= p.Age %></p>
</body>
</html>
```

- The output:
**Strongly-Typed View**

- You may pass a single model object to a view through the use of a *strongly-typed* view.
  - For an example see MvcSimple\StronglyTyped.

- To see how this works, first rewrite the controller.

```csharp
public ViewResult Index()
{
    Person p = new Person();
    return View("Index", p);
}
```
  - The first parameter to the overload of the `View()` method is the name of the view, and the second parameter is a model object.

- Next, rewrite the view page.

```html
...
<body>
    <h2>Using model data:</h2>
    <p>Name = <%= Model.Name %> </p>
    <p>Age = <%= Model.Age%> </p>
</body>
</html>
```
  - The class `Person` is passed as a type parameter to the generic `ViewPage` class, and the model object can be accessed through the variable `Model`.
  - We no longer need the script code casting to a `Person`.
Strongly-Typed View in Visual Studio

- To create a view in Visual Studio, right-click inside an action method and choose Add View from the context menu.

- You may create a strongly-typed view by selecting a view data class (model) from the dropdown.
  - You should build the application first in order that the dropdown be populated.

![Add View](image)
Lab 2

Contact Manager Application

In this lab you will implement an ASP.NET MVC application that creates a contact and displays it on the page. The lab previews the use of forms and HTML helper methods.

Detailed instructions are contained in the Lab 2 write-up at the end of the chapter.

Suggested time: 30 minutes
Summary

• You can begin creating an ASP.NET MVC application with the controller, which handles various URL requests.

• From an action method of a controller you can create a view using Visual Studio.

• You can pass data from the controller to the view by using the ViewData dictionary.

• With a model you can encapsulate the business data and logic.

• You can pass a single model object from a controller to a view through the use of a strongly-typed view.
Lab 2

Contact Manager Application

Introduction

In this lab you will implement an ASP.NET MVC application that creates a contact and displays it on the page. The lab previews the use of forms and HTML helper methods.

Suggested Time:  30 minutes

Root Directory:  C:\OIC\MvcCs

Directories:  Labs\Lab2  (do your work here)
Chap02\MvcContact  (solution)

Instructions

1. Create a new ASP.NET MVC 3 Empty Web Application MvcContact with ASPX View Engine in the working directory.

2. Create a new Contact model class under the Models folder of your new project. Include the following string properties: FirstName and LastName.

3. Add a HomeController to the project.

4. Add a view corresponding to the Index() action method.

5. Make the title of the view “Contact Manager”. Provide text boxes for inputting FirstName and LastName using Html helper methods. Provide a submit button with value “Save Contact”.

6. Build and run the application. You should see the input form displayed.
7. Provide another **Index()** action method to respond to an HTTP POST from the form. Supply the appropriate attributes to make the first method respond to GET and the second method to POST. In the second method provide a parameter of type **Person**.

```csharp
//
// GET: /Home/
[AcceptVerbs(HttpVerbs.Get)]
public ActionResult Index()
{
    return View();
}

//
// POST: /Home/
[AcceptVerbs(HttpVerbs.Post)]
public ActionResult Index(Contact contact)
{
    ViewData["contact"] = contact;
    return View();
}
```

8. Import the namespace **MvcContact.Models** so that you can access the **Contact** class.

9. Build the project to make sure you get a clean compile.

10. Inside the body of the **Index** view, below the **Html.EndForm()** helper method, provide code that will display the contact information if the **ViewData["contact"]** has been populated.

```csharp
<% Contact contact = ViewData["contact"] as Contact; %>  
<% if (contact != null) { %> 
    <p>  
    New Contact:<br/>
    <%= contact.FirstName %><%= contact.LastName %>  
    </p>  
<% } %>  
```

11. Add an import of the namespace **MvcContacts.Models** to top of the **Index** view under the `<@ Page %>` directive.

```csharp
<%@ Import Namespace="MvcContact.Models" %>
```

12. Build and run. Enter some data for First Name and Last Name. You should see output something like this:
First Name: John
Last Name: Smith

New Contact:
John Smith

13. You are done!
Chapter 10

Combining ASP.NET MVC and Web Forms
Combining ASP.NET MVC and Web Forms

Objectives

After completing this unit you will be able to:

- Describe how Web Forms can be used in an ASP.NET MVC application.
- Describe how ASP.NET MVC can be used in a Web Forms application.
Using Web Forms in an MVC Application

• Despite enormous conceptual differences between ASP.NET Web Forms and ASP.NET MVC, the technologies’ shared underlying infrastructure makes them fairly easy to integrate.

• Great integration within ASP.NET
  – All the same providers still work
  – Membership, Session, Caching, etc.
  – ASP.NET Designer Surface in VS 2008, VS 2010

• It is much easier to use a Web Forms page in an existing MVC application than using MVC in an existing Web Forms application.
Linking and Redirecting from Web Forms Pages to MVC Actions

- If you are targeting .NET Framework 4.0, your Web Forms pages have built-in support for routing.
  
  - You can generate URLs and perform redirection to MVC actions as follows:

```csharp
protected void Page_Load(object sender, EventArgs e)
{
    // You can generate a URL based on routing parameters
    string url = Page.GetRouteUrl( new {
        controller = "Home", action = "Index" });

    // ... or you can redirect to a location based on routing parameters
    Response.RedirectToRoute(new {
        controller = "Home", action = "Index" });
}
```
Typically, IIS is set by default to redirect to Default.aspx when a browser navigates to your site using a "/" as the end point of the URL.

You may want to include a Default.aspx page in your MVC application in order to redirect requests for "/" to the default MVC handler.

Add this code in the code-behind file in order to redirect requests for Default.aspx to the default MVC handler:

```csharp
public void Page_Load(object sender, EventArgs e)
{
    // Change the current path so that the Routing
    // handler can correctly interpret
    // the request, then restore the original path
    // so that the OutputCache module
    // can correctly process the response (if
    // caching is enabled).

    string originalPath = Request.Path;
    HttpContext.Current.RewritePath(
        Request.ApplicationPath, false);
    IHttpHandler httpHandler =
        new MvcHttpHandler();
    httpHandler.ProcessRequest(_HTTPContext.Current);
    HttpContext.Current.RewritePath(
        originalPath, false);
}
```
Web Forms in MVC Example

- This example illustrates using Web Forms in an ASP.NET MVC application.
  - See MvcWithWebForms in the chapter folder.
  - Building and running the example project, you see this:

  ![Web Forms in MVC Example](image)

  - There are two links: one to see a Description Web Forms page with no associated Master Page and another to see a Description Web Forms page with an associated Master Page.
  - Click the link to see the Description page with no Master Page:
Web Forms in MVC Example (Cont’d)

Your description would go here.
Notice this is an ASP.NET Web Form page rendered on an MVC site!

- Click back to the original page and click the link to see the Description page with an associated Master Page:

Your description would go here.
Notice this is an ASP.NET Web Form page rendered on an MVC site! (using a variation of the MVC MasterPage as well!)
Sharing Data between ASP.NET MVC and ASP.NET Web Forms

• Because the ASP.NET MVC framework is built on top of ASP.NET Web Forms, the following objects are always available in both technologies:
  – HttpContext
  – Session
  – Server
  – Request
  – Response
  – Cookies

• Here is an example of storing a value in the Session object in a Web Form code-behind:

```csharp
protected void Page_Load(object sender, EventArgs e)
{
    Session["Msg"] = "Message from ASP.NET";
}
```

• And an example of retrieving the shared Session object in an MVC action method:

```csharp
public ActionResult Index()
{
    ViewData["Message"] = "ASP.NET MVC!";
    ViewData["Shared"] = Session["Msg"] ?? "";
    return View();
}
```
Using Web Form Controls in MVC

Views

- In some cases, you can drop an existing ASP.NET server control into an MVC view.

- This works for render-only controls that generate HTML but don’t issue postbacks to the server.
  - Some example controls are SiteMapPath and Repeater
  - To set control properties or invoke data binding against ViewData or Model contents, you can do so by putting a `<script runat=”server”>` block anywhere in your view page.
  - Here is an example of binding to a Repeater control:

```csharp
<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        MyRepeater.DataSource = ViewData["products"];  
        MyRepeater.DataBind();
    }
</script>
```
Using MVC in a Web Forms Application

• In order to use MVC in a Web Forms application, several steps must be performed to convert the existing Web Forms application to understand how to use routing, controllers, and views.

• Once the transformation of the project is complete, you can begin to leverage the features of MVC (such as REST-like URLs, separation of concerns, testability, etc.) alongside your existing ASP.NET pages.

• There are two types of ASP.NET web applications in Visual Studio: web site projects and web application projects (both use Web Forms, but are distinctly different).
  
  – You can only convert a web application project and not a web site project to work with MVC.

  – First, convert your web site project to a web application project (not covered here) and then proceed with the steps to convert your new web application project into a hybrid MVC - Web Forms application.
Steps to Create Hybrid MVC - Web Forms Application

- These are the steps needed to convert an existing Web Forms application into a Hybrid MVC - Web Forms application:
  - Add a reference to `System.Web.Mvc.dll` - you will need to browse for it under C:\Program Files\Microsoft ASP.NET\ASP.NET MVC 3\Assemblies
  - Add a reference to `System.Web.Routing`
  - Create an Empty MVC Project
    a) Copy the Controllers, Models and Views directories into the Web Forms application
    b) Copy the `RegisterRoutes()` method in the `Global.asax` file to the `Global.asax` file in the Web Forms project
    c) Add `RegisterRoutes(RouteTable.Routes);` to the `Application_Start()` in the Web Forms project
  - Add the following to the `<compilation>` section of `web.config`:
Steps to Create Hybrid MVC - Web Forms Application (Cont’d)

<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>

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

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

− 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

− Close and reopen the solution
Lab 10

Converting an Existing Web Forms Application to Use MVC

In this lab you will convert an existing ASP.NET Web Forms application to use ASP.NET MVC.

Detailed instructions are contained in the Lab 10 write-up at the end of the chapter.

Suggested time: 45 minutes
Summary

- Because ASP.NET MVC sits on top of the ASP.NET Framework, Web Forms will work as expected in an MVC application.

- Many common objects such as HttpContext, Session, Server, etc. can be used in both technologies.

- To use ASP.NET MVC in an existing Web Forms application you will need to go through several steps to convert the Web Forms application.
Lab 10

Converting an Existing Web Forms Application to Use MVC

Introduction

In this lab you will convert an existing ASP.NET Web Forms application to use ASP.NET MVC.

Suggested Time: 45 minutes

Root Directory: C:\OIC\MvcCs

Directories:

- Labs\Lab10 (do your work here)
- Chap10\MvcConvertWebForms\Step1 (solution to Part 1)
- Chap10\MvcConvertWebForms\Step2 (solution to Part 2)
- Chap10\MvcConvertWebForms\Step3 (solution to Part 3)
- Chap10\MvcConvertWebForms\Step4 (solution to Part 4)

Part 1. Create Web Forms Application

1. Create a new ASP.NET Web Forms Application MvcConvertWebForms in the working directory.

2. Build and run the application. You should see the following form displayed.
3. You are now at Step 1.

**Part 2. Convert Web Forms Application to use MVC**

1. Continue working with the existing **MvcConvertWebForms** application created in Step 1.

2. Add a reference to System.Web.Mvc.dll - you will need to browse for it under C:\Program Files\Microsoft ASP.NET\ASP.NET MVC 3\Assemblies.


4. Add new folders Controllers and Views to the Web Forms application.

5. Copy the RegisterRoutes method in the **Global.asax.cs** file in the MVC project to the **Global.asax.cs** file in the Web Forms project.

6. Add `RegisterRoutes(RouteTable.Routes);` to the **Application_Start()** method in the Web Forms project.


8. Build and verify the application compiles. Run the application and it should still look like a Web Forms application at this point.

9. Add the following to the `<compilation>` section of **web.config** in the Web Forms project:

   ```
   <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=3.0.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" />
   </assemblies>
   ```

10. Add the following to the `<pages>` section of **web.config** in the Web Forms project
    
    *(Note: You may need to create a `<pages>` section in the `<system.web>` section)*:

    ```
    <namespaces>
    <add namespace="System.Web.Mvc" />
    </namespaces>
    ```
11. We will need to enable MVC Tooling in the Web Forms Application. Close the Web Forms project.

12. Open the file **MvcConvertWebForms.csproj**.

13. In <ProjectTypeGuids> section add at beginning of list of Guids: {f85e285d-a4e0-4152-9332-ab1d724d3325}. Save the file, and then close it.

14. Open the solution.

15. Build and verify the application compiles. Run the application and it should still look like a Web Forms application at this point.

16. You are now at Step 2.

**Part 3. Use new Hybrid MVC-Web Forms Application with Default Web Forms**

1. Continue working with the existing **MvcConvertWebForms** application modified in Step 2.

2. In the **Default.aspx** page, add the following code at the bottom of the Main Content section:

   ```html
   <p>
   <a href="/Home/Index">ASP.NET MVC Home Page</a>
   </p>
   ```

3. Add aHomeController to the project.

4. Add a view corresponding to the **Index()** action method. Do not select a master page.

5. Make the title of the view “Hello from MVC”. Add code in the body to say "This page is rendering through MVC".

   ```html
   <html xmlns="http://www.w3.org/1999/xhtml" >
   <head runat="server">
     <title>Hello from MVC</title>
   </head>
   <body>
   This page is rendering through MVC
   </body>
   </html>
   ```

6. Set **Default.aspx** as the start page. Build and run the application. You should see the following form:
7. Click on the ASP.NET MVC Home Page link and you get this:

8. You are now at Step 3.

**Part 4. Use new Hybrid MVC-Web Forms Application with Default MVC**

1. Continue working with the existing `MvcConvertWebForms` application modified in Step 3.

2. We modify Default.aspx to bypass this page as the Home page for the new Hybrid MVC-Web Forms Application and instead redirect requests to the default MVC routing path. Add this code to the Default.aspx code-behind file:
public void Page_Load(object sender, EventArgs e)
{
    // Change the current path so that the Routing
    // handler can correctly interpret
    // the request, then restore the original path
    // so that the OutputCache module
    // can correctly process the response (if
    // caching is enabled).

    string originalPath = Request.Path;
    HttpContext.Current.RewritePath(Request.ApplicationPath, false);
    IHttpHandler httpHandler = new MvcHttpHandler();
    httpHandler.ProcessRequest(HttpContext.Current);
    HttpContext.Current.RewritePath(originalPath, false);
}


4. On the Default.aspx page, remove all code and replace with this:

    <%@ Page Language="C#" AutoEventWireup="true"
    CodeBehind="Default.aspx.cs" Inherits="MvcConvertWebForm._Default" %>

    <%- Please do not delete this file. It is used to ensure that
ASP.NET MVC is activated by IIS when a user makes a "/" request to
the server. --%>

5. Rebuild and run the application. You should now see the MVC Home page.

6. You are now at Step 4.