
# Hands-On Lab

## *ASP.NET MVC Razor*

Lab version: 1.1.0

Last updated: 7/11/2012

**CONTENTS**

# Overview

**Note:** This Hands-on Lab assumes you have basic knowledge of **ASP.NET MVC**. If you have not used **ASP.NET MVC** before, we recommend you to go over **ASP.NET MVC Fundamentals** Hand-on Lab.

ASP.NET MVC 3 introduces the new view engine Razor, which was conceived to simplify the current syntax used in ASP.NET web pages. In this lab, you will learn how to create Razor views inside an MVC solution.

In the beginning of this Lab you will first get familiar with Razor syntax and its features: general rules, code blocks, inline HTML and layout pages. Once you have learned the basics, in Exercise 2 you will add Razor views in the MVC Music Store solution to provide the same functionality you have in ASP.NET views, but with a clearer and reduced code.

At the end of this lab, in Exercise 3, you will learn how to create and consume Razor Helpers to use images, grids and custom functionality.

## About Razor View Engine

Many ASP.NET Web pages have C# or VB code blocks in the same place as HTML markup. In some occasions this combination is uncomfortable for writing and delimiting code. To deal with that problem, Razor was designed as an easy to learn, compact and expressive view engine that enables a fluid coding workflow.

Razor is not a new programming language itself, but uses C# or Visual Basic syntax for having code inside a page without ASP.NET delimiter: **<%= %>**. Razor file extension is '**cshtml'** for C# language, and '**vbhtml'** for Visual Basic.

## Razor Syntax – The fundamentals

Before getting introduced to Razor you should first know some simple rules that will help you understand how to write HTML with C# or Visual Basic in the same page:

1. '**@**' is the magic character that precedes code instructions in the following contexts:
    a. '@' For a single code line/values:

    A single code line inside the markup:

    **cshtml**

```
<p>
   Current time is: @DateTime.Now
</p>
```

**vbhtml**

```
<p>
   Current time is: @DateTime.Now
</p>
```

b. '@{ … }' For code blocks with multiple lines:

**cshtml**

```
@{
     var name = "John";
     var nameMessage = "Hello, my name is " + name + " Smith";
}
```

**vbhtml**

```
@Code
     Dim name = "John"
     Dim nameMessage = "Hello, my name is " + name + " Smith"
End Code
```

c. '@:' For single plain text to be rendered in the page.

**cshtml**

```
@{
    @:The day is: @DateTime.Now.DayOfWeek. It is a <b>great</b> day!
}
```

**vbhtml**

```
@Code
    @:The day is: @DateTime.Now.DayOfWeek. It is a <b>great</b> day!
End Code
```

2. HTML markup lines can be included at any part of the code:

It is no need to open or close code blocks to write HTML inside a page. If you want to add a code instruction inside HTML, you will need to use '@' before the code:

**cshtml**

```
@if(IsPost) {
     <p>Hello, the time is @DateTime.Now and this page is a postback!</p>
} else {
     <p>Hello, today is: </p> @DateTime.Now
```

```
}
```

**vbhtml**

```
@If IsPost Then
    @<p>Hello, the time is @DateTime.Now and this page is a postback!</p>
 Else
    @<p>Hello, today is: </p> @DateTime.Now
End If
```

3. Razor uses code syntax to infer indent:

   Razor Parser infers code ending by reading the opening and the closing characters or HTML elements. In consequence, the use of openings "{" and closings "}" is mandatory, even for single line instructions:

**cshtml**

```
// This won't work in Razor. Content has to be wrapped between { }
if( i < 1 ) int myVar=0;
```

**vbhtml**

```
// This won't work in Razor. Content has to be wrapped between { }
If i < 1 Then   Dim myVar As int =0 End if
```

---

## Conditionals and loops with inline HTML

Here you will find examples of conditionals with inline html.

- If statement:

**cshtml**

```
<p>Single line If</p>
@if(result != ""){
    <p>Result: @result</p>
}
```

**vbhtml**

```
<p>Single line If</p>
@If result <> "" Then
    @<p>Result: @result</p>
End If
```

**cshtml**

```
<p>Code block If</p>

@{
    var showToday = false;

    if(showToday){
        @DateTime.Today;
    } else{
        <text>Sorry!</text>
    }
}
```

**vbhtml**

```
<p>Code block If</p>

@Code
    Dim showToday = false

    If showToday Then
        @DateTime.Today
    Else
        @<text>Sorry!</text>
    End if
End Code
```

- Foreach statement:

**cshtml**

```
<p> Single Line Foreach </p>
<ul>
@foreach (var myItem in Request.ServerVariables){
    <li>@myItem</li>
}
</ul>
```

**vbhtml**

```
<p> Single Line Foreach </p>
<ul>
@For Each myItem in Request.ServerVariables
    @<li>@myItem</li>
Next myItem
</ul>
```

**cshtml**

```
<p> Code block Foreach </p>
@{
    <h3>Team Members</h3> string[] teamMembers = {"Matt", "Joanne", "Robert",
"Nancy"};
    foreach (var person in teamMembers)
    {
        <p>@person</p>
    }
}
```

**vbhtml**

```
<p> Code block Foreach </p>
@Code
    @<h3>Team Members</h3>
    Dim teamMembers() As String = {"Matt", "Joanne", "Robert", "Nancy"}
    For Each person in teamMembers
            @<p>@person</p>
    Next person
End Code
```

- While statement:

**cshtml**

```
@{
    var countNum = 0;
    while (countNum < 50) {
        countNum += 1;
        <p>Line #@countNum: </p>
    }
}
```

**vbhtml**

```
@Code
    Dim countNum = 0
    Do While countNum < 50
        countNum += 1
        @<p>Line #@countNum: </p>
    Loop
End Code
```

## Comments

Comments in Razor are delimited by **@* *@.** If you are inside a C# code block, you can also use // and /* */ comment delimiters.

**cshtml**

```
@*
    A Razor Comment
*@

@{
    //A C# comment

    /* A Multi
         line C# comment
    */
}
```

**vbhtml**

```
@*
    A Razor Comment
*@

@Code
    //A C# comment

    /* A Multi
         line C# comment
    */
End Code
```

---

In this Hands-on Lab, you will learn how to:

- Create Razor Layout page templates
- Create MVC Views using Razor syntax
- Create and consume Razor Helpers

---

# System Requirements

You must have the following items to complete this lab:

- ASP.NET and ASP.NET MVC 3

- Visual Studio 2010 Express
- SQL Server Database (Express edition or above)

> **Note:** You can install the previous system requirements by using the Web Platform Installer 3.0: http://go.microsoft.com/fwlink/?LinkID=194638.

## Exercises

This Hands-On Lab is comprised by the following exercises:

1. Exercise 1: Creating a Home page view using Razor syntax
2. Exercise 2: Using Models in Razor views
3. Exercise 3: Creating and Consuming Razor Helpers from a View

Estimated time to complete this lab: **30 minutes**.

> **Note:** Each exercise is accompanied by an **End** folder containing the resulting solution you should obtain after completing the exercises. You can use this solution as a guide if you need additional help working through the exercises.

## Next Step

Exercise 1: Creating a Home page view using Razor syntax

# Exercise 1: Creating a Home Page View Using Razor Syntax

You have been working with ASP.NET C# language and HTML to create controllers and views. In this exercise, you will learn how to create MVC 3 views by using **Razor syntax** and **Razor Layout Pages**. In order to do that you will work with MVC Music Store web application to write its **Home Page view in Razor**.

## Razor Layout Pages

A **Razor Layout page** is the equivalent of an ASP.NET Master Page. It **defines the structure** of a Web page by using **HTML markup**, and defines sections that will be filled with custom content. Web pages linked to a layout will only have to complete the sections with their own HTML. By using layout pages your site will have a **consistent look** that will also be easier to modify.

## Razor Layout Pages – Structure

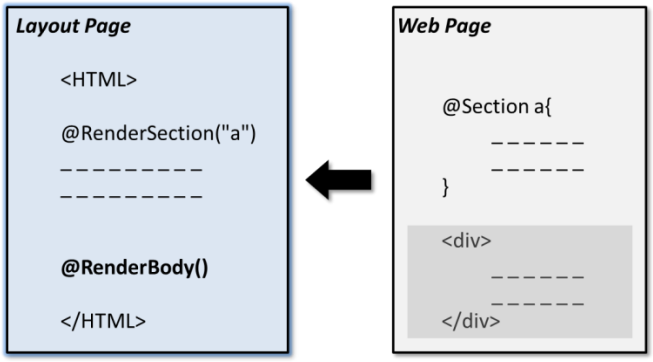


**Figure 1**
*Layout and Web Pages*

In essence, Razor Layout pages are HTML files that define a **Body** and additional **sections** by using the following instructions:

- **@RenderBody():** Renders the content of a page that is not within any named sections.
- **@RenderSection("mySectionName"):** Renders the content of the section within the given name. Sections are mandatory by default, so each page linked to the template will have to declare them.
- **@RenderSection("mySectionName", optional:true):** Renders the content of an optional section. The pages that are linked to the template can omit the definition of an optional section.

**Note**: **Optional sections** allow us to use the same layout in pages that share part of the structure. However, **this feature has to be carefully used if we want to keep a tidy and maintainable layout**. In those cases, when the differences between the pages are significant, it will be better to add a new page layout instead of having a complex one.

Next you will see an example of a layout that defines two sections, one of them optional:

**Site.cshtml**

```
<!DOCTYPE html>
<html>
```

```
<head>
<title>Layout Page Content</title>
    <link href="/Content/Site.css" rel="Stylesheet" type="text/css" />
</head>
<body>
    <div id="header">
        @RenderSection("header")
    </div>

    <div id="main">
        @RenderBody()
    </div>

    <div id="footer">
        @RenderSection("footer", optional:true)
    </div>
</body>
</html>
```

**Site.vbhtml**

```
<!DOCTYPE html>
<html>
<head>
<title>Layout Page Content</title>
    <link href="/Content/Site.css" rel="Stylesheet" type="text/css" />
</head>
<body>
    <div id="header">
        @RenderSection("header")
    </div>

    <div id="main">
        @RenderBody()
    </div>

    <div id="footer">
        @RenderSection("footer", optional:=true)
    </div>
</body>
</html>
```

**Note:** Optional sections can also be defined by using Razor directive **IsSectionDefined("name")**.

The following line is equival to "**@RenderSection("footer",optional:true)**":

**@if (IsSectionDefined("footer")) {**

```
        @RenderSection("footer")

    }
```

Here is an example of a Razor Web page that links to the previous layout:

**cshtml**

```
@{
    LayoutPage = "~/Shared/Site.cshtml";
}

@section header {
<p>
    Razor example
</p>
}

@section footer {
<p>
    Optional page footer
</p>
}

<div>
    Using Layout Pages – Body content
</div>
```

**vbhtml**

```
@Code
    LayoutPage = "~/Shared/Site.vbhtml"
End Code

@Section header
<p>
    Razor example
</p>
End Section

@ Section footer
<p>
    Optional page footer
</p>
End Section

<div>
    Using Layout Pages – Body content
```

```
</div>
```

> **Note:** Most of the HTML elements were moved to the layout, leaving only the body and the section declarations.

### Task 1 – Creating a CSHTML|VBHTML Layout Page

In this task, you will learn how to create a Layout page for Razor views that will work like the Site.master page from the ASP.NET MVC Fundamentals Hands-on Lab exercises.

1. If not already open, start Microsoft Visual Web Developer 2010 Express from **Start | All Programs | Microsoft Visual Studio 2010 Express | Microsoft Visual Web Developer 2010 Express**.
2. In the **File** menu, choose **Open Project**. In the Open Project dialog, browse to Source\Ex01-CreatingAHomeRazorView\Begin, select **MvcMusicStore.sln** and click **Open**.
3. Add a **Layout Page**. Because it is a shared resource you will create it under the **/Views/Shared** folder, in the same place of ASP.NET MasterPage. To do this, expand the **Views** folder and right-click the **Shared** folder. Select **Add** and then the **New Item** command.

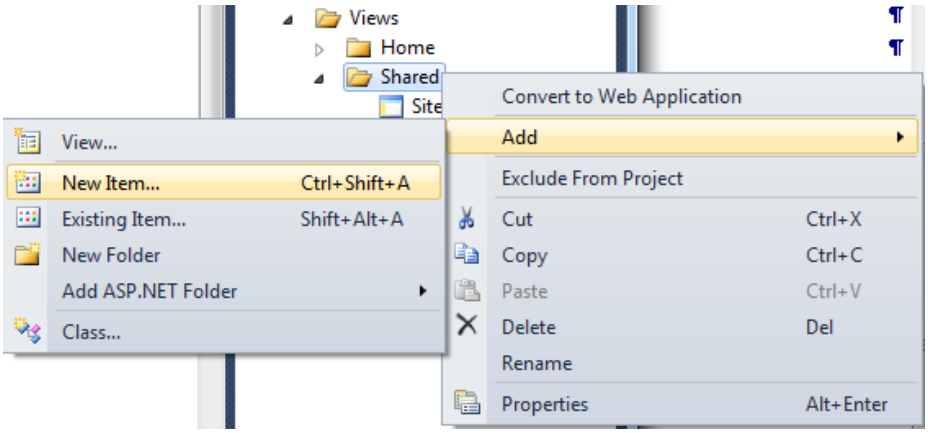


**Figure 2**

*Adding a new item to the solution*

4. In the **Add New Item** dialog box, select the **MVC 3 Layout Page (Razor)** template, located under **Visual [C#|Basic], Web** template list. Change the name to **Site.cshtml|vbhtml** and click **Add**.

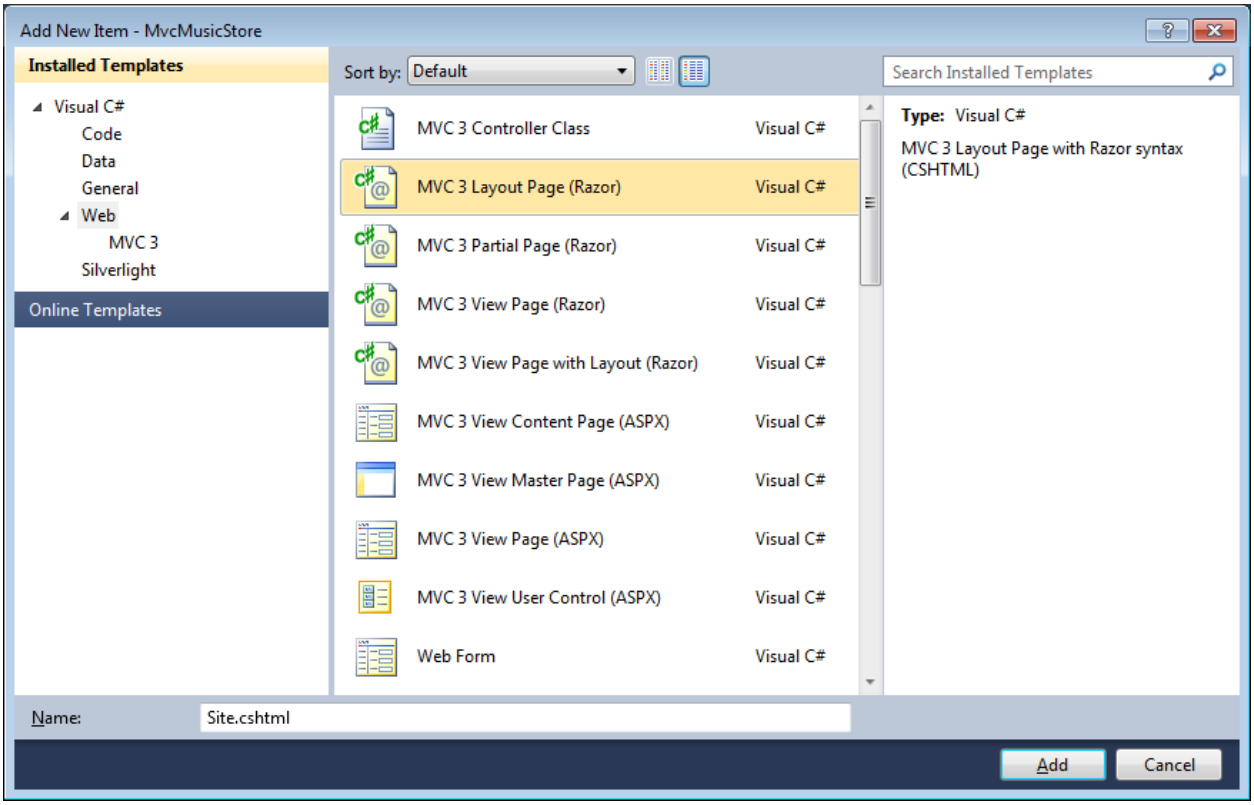


**Figure 3**

*Adding a new Razor Layout Page – C#*

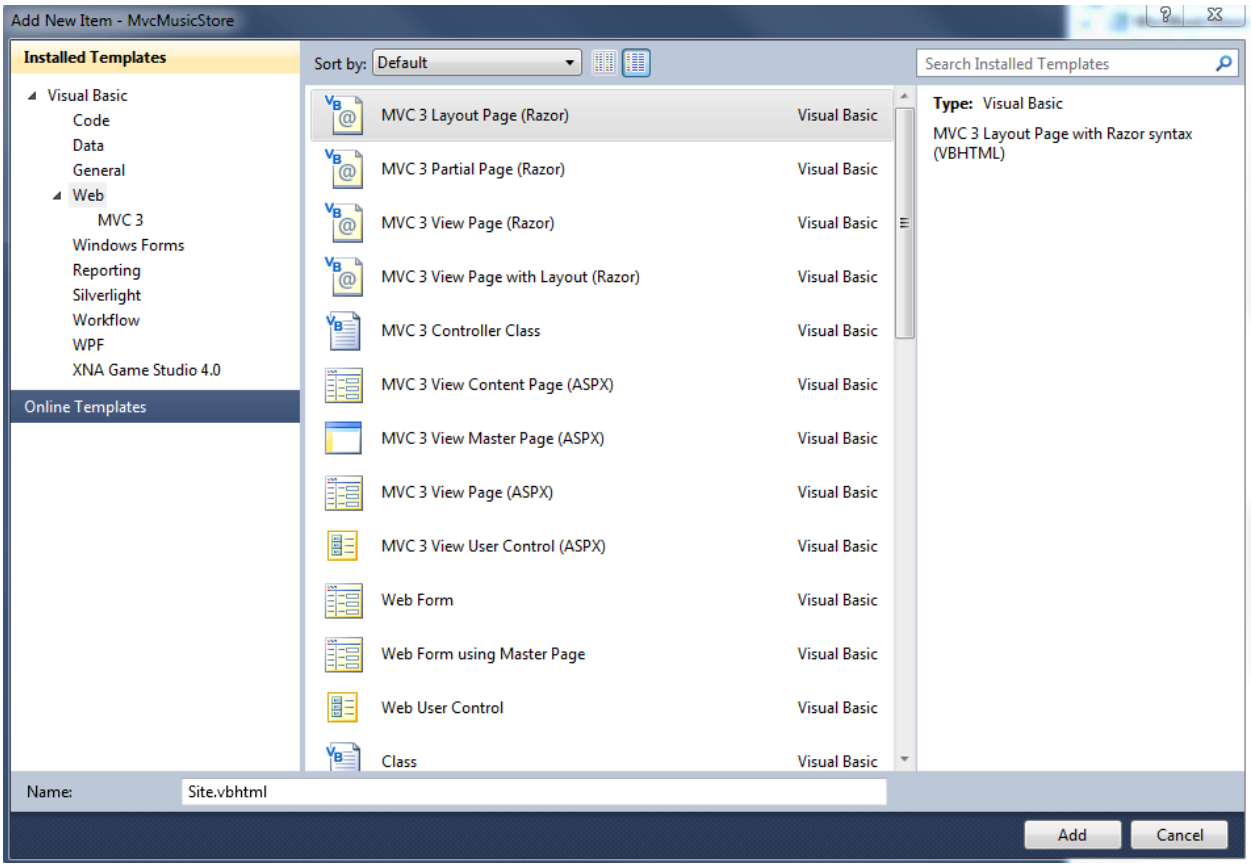


**Figure 4**

*Adding a new Razor Layout Page - VB*

5. Site.cshtml|vbhtml file is added. This template contains the HTML layout for all pages on the site. It includes the **<html>** element for the HTML response, as well as the **<head>** and **<body>** elements.

**cshtml**

```
<!DOCTYPE html>

<html>
<head>
    <title>@ViewBag.Title</title>
</head>
<body>
    <div>
        @RenderBody()
    </div>
</body>
</html>
```

**vbhtml**

```
<!DOCTYPE html>

<html>
<head>
    <title>@ViewBag.Title</title>
</head>
<body>
    <div>
        @RenderBody()
    </div>
</body>
</html>
```

**Note:** This is the standard layout page generated by MVC 3, which displays the body inside a  block.

**@ViewBag.Title** (equivalent of @ViewBag ["Title"] used in ASP.NET to access ViewBag collection) shows the content of "Title" element.

As **@ViewBag** refers to dynamic type collection, it is possible to assign any type of elements inside.

6. Add a reference to Mvc MusicStore stylesheet, to define the style of your site.

**cshtml**

```
<!DOCTYPE html>

<html>
<head>
    <link href="/Content/Site.css" rel="Stylesheet" type="text/css" />
    <title>@ViewBag.Title</title>
</head>
<body>
    <div>
        @RenderBody()
    </div>
</body>
</html>
```

**vbhtml**

```
<!DOCTYPE html>

<html>
<head>
```

```
    <link href="/Content/Site.css" rel="Stylesheet" type="text/css" />
    <title>@ViewBag.Title</title>
</head>
<body>
    <div>
        @RenderBody()
    </div>
</body>
</html>
```

7. Add a common header with links to the Home page and Store area on all pages in the site. In order to do that, add the following code inside the ** ** statement.

**cshtml**

```
<!DOCTYPE html>

<html>
<head>
      <link href="/Content/Site.css" rel="Stylesheet" type="text/css" />
      <title>@ViewBag.Title</title>
</head>

<body>
    <div>
       <div id="header">
            <h1>ASP.NET MVC MUSIC STORE</h1>
            <ul id="navlist">
                <li class="first"><a href="/" id="current">Home</a></li>
                <li><a href="/Store/">Store</a></li>
            </ul>
        </div>
        @RenderBody()
    </div>
</body>
</html>
```

**vbhtml**

```
<!DOCTYPE html>

<html>
<head>
      <link href="/Content/Site.css" rel="Stylesheet" type="text/css" />
      <title>@ViewBag.Title</title>
</head>

<body>
```

```
    <div>
        <div id="header">
            <h1>ASP.NET MVC MUSIC STORE</h1>
            <ul id="navlist">
                <li class="first"><a href="/" id="current">Home</a></li>
                <li><a href="/Store/">Store</a></li>
            </ul>
        </div>
        @RenderBody()
    </div>
</body>
</html>
```

8. Add an optional section before the body by using **IsSectionDefined** directive:

**cshtml**

```
…
      </div>
      @if (IsSectionDefined("header")) {
            @RenderSection("header")
      }
       @RenderBody()
</div>
```

**vbhtml**

```
…
      </div>
      @If IsSectionDefined("header") Then
            @RenderSection("header")
      End if
       @RenderBody()
</div>
```

> **Note:** Optional sections can also be defined by using the directive **@RenderSection("header",optional:true)**

---

### Task 2 – Adding a View Template in Razor

In this task, you will add a view template in Razor for the **Home** page. At the end of the task you should get the same page you use to have when using ASPX templates, but with a simplified syntax.

1. Open **HomeController.[cs|vb]** class and right-click inside the Index Action method display context menu. Select **Add View** menu option to display the dialog box.

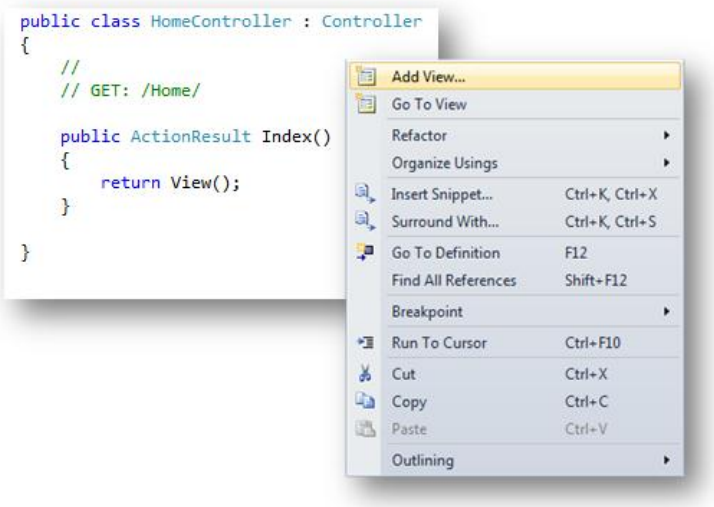


**Figure 5**

*Adding a new Home Index View from a Razor Template - CS*

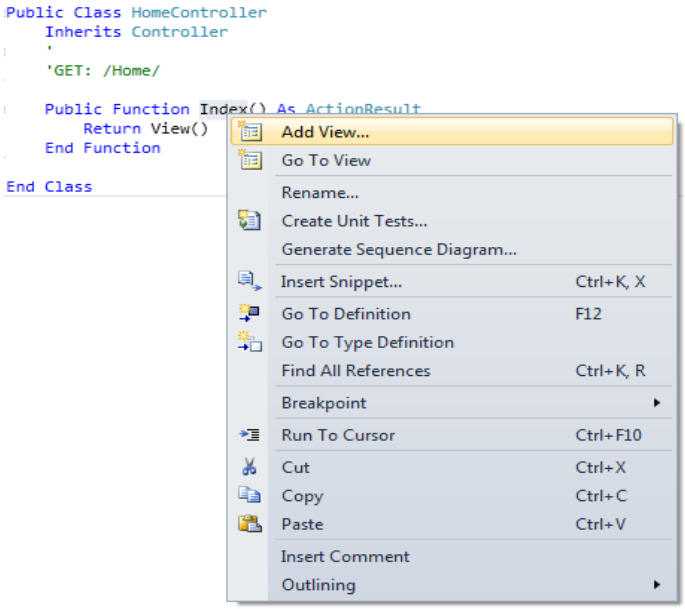


**Figure 6**

*Adding a new Home Index View from a Razor Template - VB*

2. The **Add View** Dialog appears. It allows generating View template files. By default this dialog pre-populates the name of the View template so that it matches the action method that will use it (in this case, View name is set to Index). Fill the dialog box according to the following values and click **Add**:

    a. Choose **Razor (CSHTML|VBHTML)** at "View Engine".

b. Make sure the "Use a layout or master page" checkbox is checked and verify that Razor Layout template path is "**~/Views/Shared/Site.cshtml|vbhtml**".

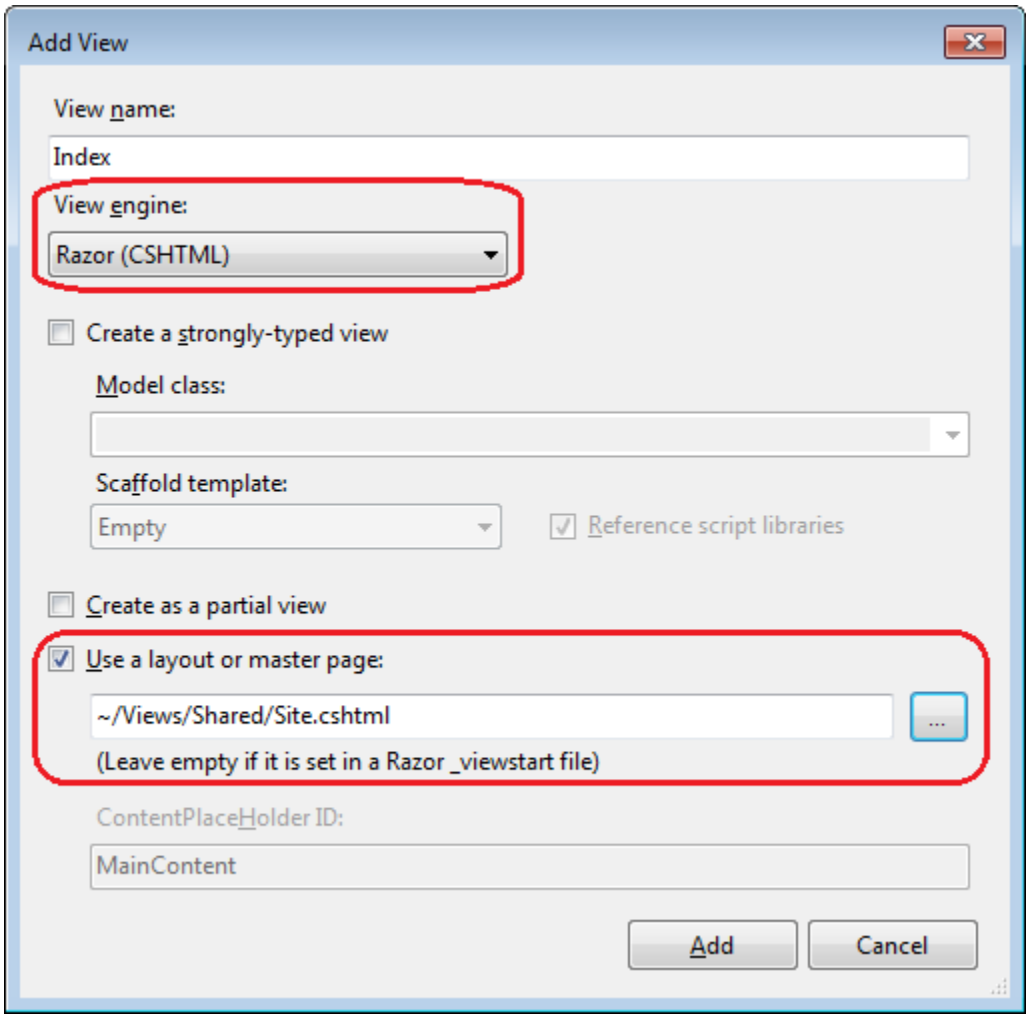


**Figure 7**

*Adding a new Razor View dialog box – C#*

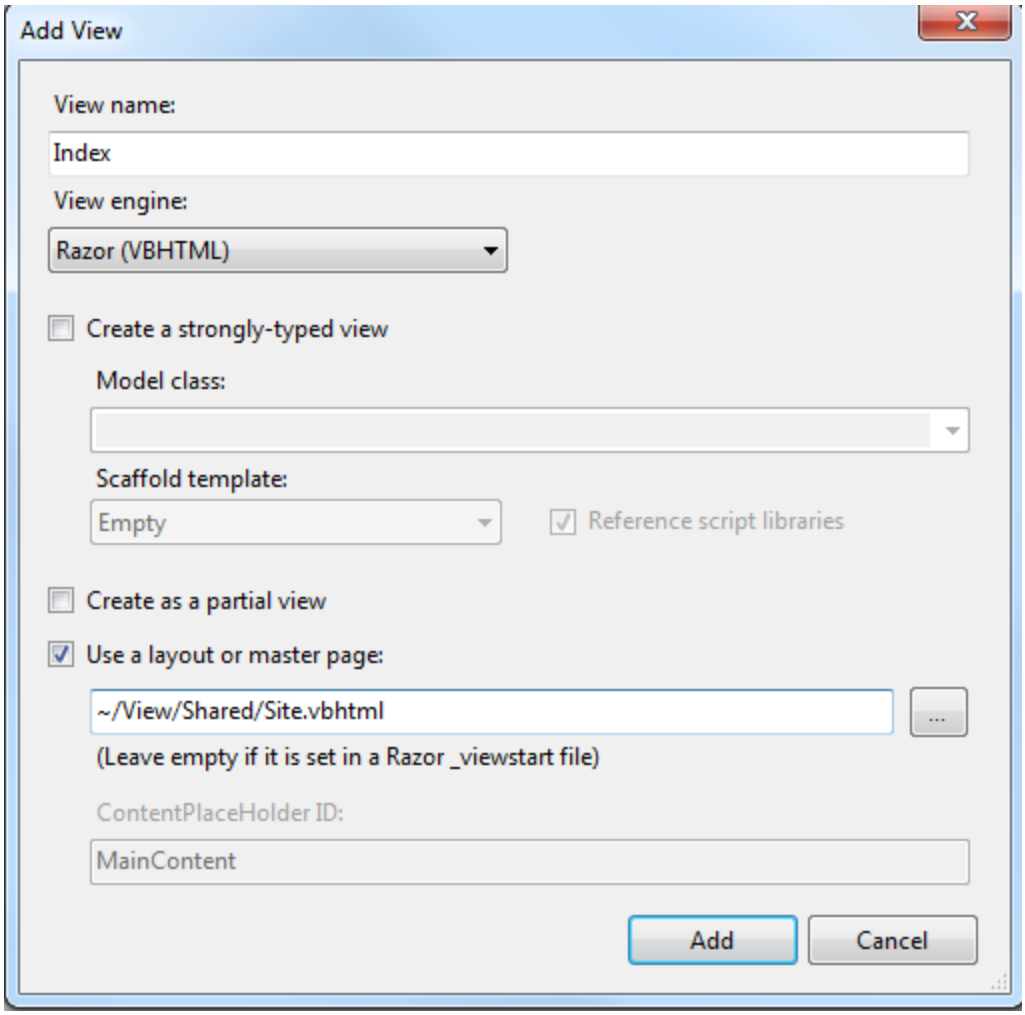


**Figure 8**

*Adding a new Razor View dialog box - VB*

3. Visual Studio generates an **Index.cshtml|vbhtml** Razor view template inside the **Views\Home** folder with this structure:

**cshtml**

```
@{
    ViewBag.Title = "Index";
    Layout = "~/Views/Shared/Site.cshtml";
}

<h2>Index</h2>
```

**vbhtml**

```
@Code
    ViewBag.Title = "Index"
```

```
    Layout = "~/Views/Shared/Site.vbhtml"
End Code

<h2>Index</h2>
```

**Note:** You can change your page layout by modifying the Layout variable value.

4. Add a comment and a title inside a code block with Razor syntax.

**cshtml**

```
@{
    ViewBag.Title = "Home";
    Layout = "~/Views/Shared/Site.cshtml";
}

<h2>This is the Home Page</h2>

@{
    var myname = "MVC Music Store";
    <div>
        My name is @myname
    </div>
}
@*
    <div>
        I'm inside a comment
    </div>
*@
```

**vbhtml**

```
@Code
    ViewBag.Title = "Home"
    Layout = "~/Views/Shared/Site.vbhtml"
End Code

<h2>This is the Home Page</h2>

@Code
    Dim myname = "MVC Music Store"
    <div>
        My name is @myname
    </div>
End Code
@*
    <div>
```

```
        I'm inside a comment
    </div>
*@
```

**Note:** If you have an HTML markup inside a code block (in this example, ** **) and want to use C# code instruction inside, you will need to use the **@** operator. After the HTML markup is closed (in this example , ** )**, you can continue writing C# code again without using **@**.

5. Show current date after the title, and use **@:** delimiter to write a variable inside a code block:

**cshtml**

```
@{
    ViewBag.Title = "Home";
    Layout = "~/Views/Shared/Site.cshtml";
}

<h2>This is the Home Page</h2>

@{
    var myname = "MVC Music Store";
    <div>
        My name is @myname
    </div>
}

@{
    var now = DateTime.Now;
    <text>
        Now is @now
    </text>
}

<div>
@{
    var engine = "Razor";
    @: I'm using @engine engine.
}
</div>

@*
    <div>
        I'm inside a comment
    </div>
*@
```

**vbhtml**

```
@Code
    ViewBag.Title = "Home"
    Layout = "~/Views/Shared/Site.vbhtml"
End Code

<h2>This is the Home Page</h2>

@Code
    Dim myname = "MVC Music Store"
    <div>
        My name is @myname
    </div>
End Code
```

```
@Code
    Dim now = DateTime.Now
    <text>
        Now is @now
    </text>
End Code

<div>
@Code
    Dim engine = "Razor"
    @: I'm using @engine engine.
End Code
</div>

@*
    <div>
        I'm inside a comment
    </div>
*@
```

**Note: @:** operator allows **single lines** that can contain **plain text** or any mixture of text, markup and code that starts with plain text. **@:** must be used once per line.

To write multiple plain text/mixed lines, use the HTML element **<text></text>** instead.

Note that is not necessary to use **@:** inside a code block if you won't be starting with plain text. As you have seen before, HTML markup elements can be included at any part of the code.

6. Add the optative section "Header", which was defined in the layout page. This will render a "Welcome" message before the body section:

**cshtml**

```
@{
    ViewBag.Title = "Home";
    Layout = "~/Views/Shared/Site.cshtml";
}

@section header {
   <h3>Welcome</h3>
}

<h2>This is the Home Page</h2>
…
```

**vbhtml**

```
@Code
    ViewBag.Title = "Home"
    Layout = "~/Views/Shared/Site.vbhtml"
End Code

@Section header
   <h3>Welcome</h3>
End Section

<h2>This is the Home Page</h2>
…
```

---

**Task 3 – Running the Application**

1. Press **F5** to run the application.
2. The project starts in the Home page. Verify that the Razor view is loaded:

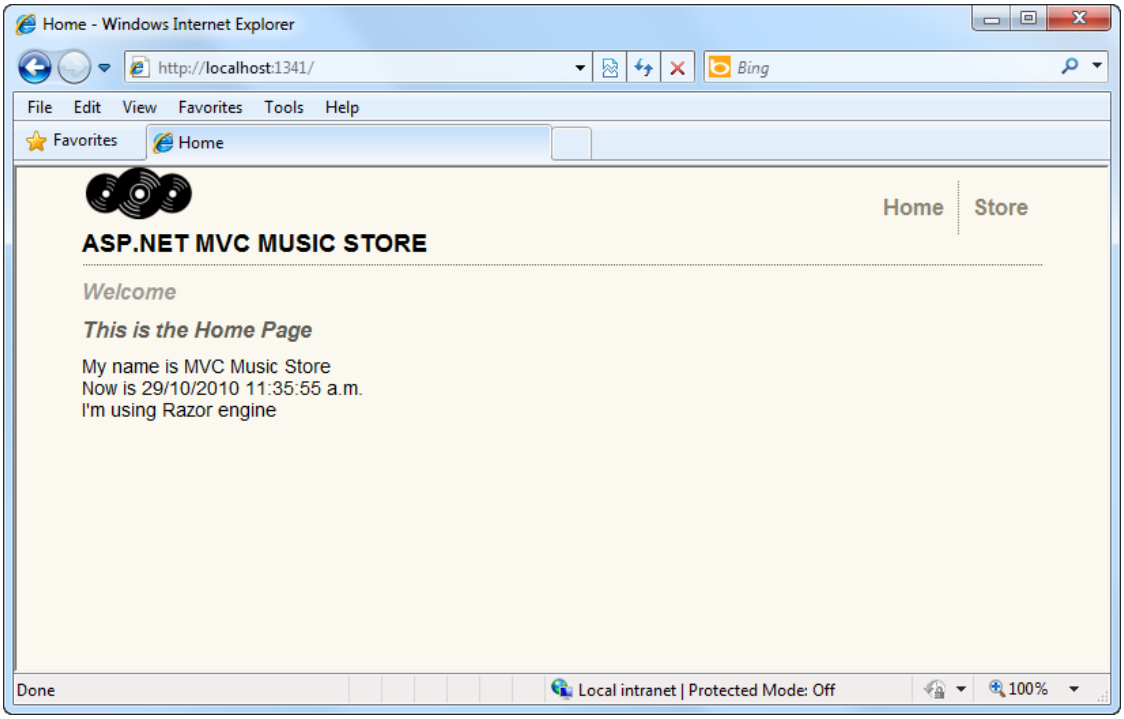


**Figure 9**

*Showing a Razor View*

---

**Task 4 – Passing Parameters to a Layout Page**

In this task, you will learn how to pass parameters from a page to its layout. To do that, you will use the **PageData** dynamic collection to pass a string message that will be rendered in the layout.

1. Open **Index.cshtml|vbhtml** view from the folder **\Views\Home**, and assign a string value to PageData collection:

**cshtml**

```
@{
    ViewBag.Title = "Home";
    Layout = "~/Views/Shared/Site.cshtml";

    PageData["ApplicationTitle"]= "ASP.NET MVC 3 MUSIC STORE – Razor Style";
}
…
```

**vbhtml**

```
@Code
    ViewBag.Title = "Home";
    Layout = "~/Views/Shared/Site.vbhtml"
```

```
    PageData("ApplicationTitle")= "ASP.NET MVC 3 MUSIC STORE – Razor
    Style"
End Code
…
```

2. Open **Site.cshtml|vbhtml** layout page from **\Views\Shared** project folder, and modify it to show PageData **ApplicationTitle** element after the @RenderBody directive:

**cshtml – Site.cshtml**

```
…
        @RenderBody()

        @if (PageData["ApplicationTitle"]!=null) {
           <h3>@PageData["ApplicationTitle"]</h3>
        }
…
```

**vbhtml – Site.vbhtml**

```
…
        @RenderBody()

        @If PageData("ApplicationTitle") IsNot Nothing Then
           <h3>@PageData["ApplicationTitle"]</h3>
        End if
…
```

---

**Task 5 – Running the Application**

1. Press **F5** to run the application.
2. The project starts in the Home page. Verify that the Razor view is loaded showing the PageData text:

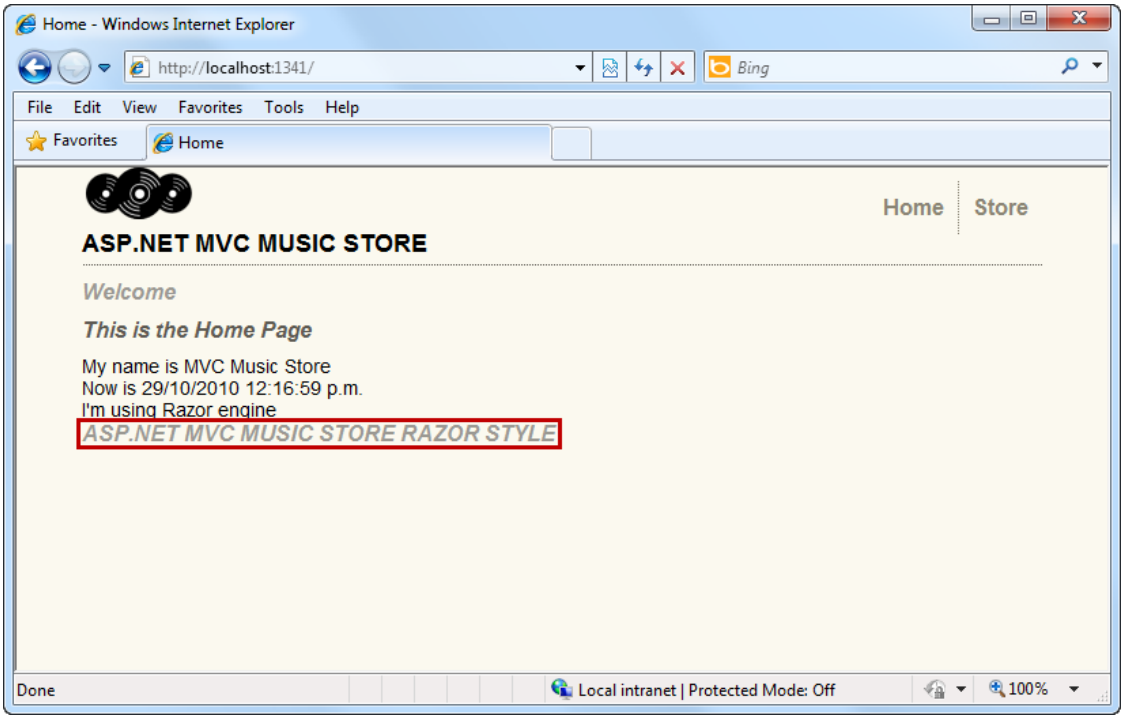


**Figure 10**

*A Razor View Showing a PageData element*

In the next exercise you will learn how to create more Razor views that will access to Music Store model and show content.

## Next Step

Exercise 2: Using Models in Razor Views

# Exercise 2: Using Models in Razor Views

In this exercise, you will work with models in Razor views. Initially, you will use the **@inherits** directive that was released as part of MVC 3.0, and then the new **@model** directive that is now supported in the Razor engine. These directives provide a simplified way to reference your strongly-typed models in your view files. You will add view templates used by action methods to browse the albums in the music store.

**Task 1 – Using the @inherits Directive**

In this task, you will replace the **Index** action method of the **HomeController** to retrieve the Date & time formatted in the webserver's culture.

1. If not already open, start Microsoft Visual Web Developer 2010 Express from **Start | All Programs | Microsoft Visual Studio 2010 Express | Microsoft Visual Web Developer 2010 Express**.
2. In the **File** menu, choose **Open Project**. In the Open Project dialog, browse to Source\Ex02-UsingModelsInRazorViews\Begin, select MvcMusicStore.sln and click **Open**.
3. In the **Solution Explorer**, browse to the **Controllers** folders and then double-click the **HomeController.[cs|vb]** class.
4. Replace the implementation of the view method, to pass the current Date and Time to the view. To do this, replace the following code.

**C#**

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

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

        public ActionResult Index()
        {
            return View(DateTime.Now);
        }

    }
}
```

**Visual Basic**

```
Imports System
Imports System.Collections.Generic
Imports System.Linq
Imports System.Web
Imports System.Web.Mvc

Namespace MvcMusicStore.Controllers

    public class HomeController : Controller

        '
        ' GET: /Home/
```

```
        Public Function Index() As ActionResult

            return View(DateTime.Now)
        End Function

    End Class
End Namespace
```

5. Open the **Index** view associated to this controller. To do this in the Solution Explorer navigate to **Views\Home** and double-click **Index.cshtml|vbhtml**.
6. Replace the **@model dynamic;** line at the top with the following one:

**C#**

```
@inherits System.Web.Mvc.WebViewPage<DateTime>
```

**Visual Basic**

```
@inherits System.Web.Mvc.WebViewPage (Of DateTime)
```

> **Note:** The **@inherits** directive indicates that you want to derive the view from System.Web.Mvc.WebViewPage<DateTime> class. This way, you can define strong-type models inside a Razor's view.

7. Add a reference to the **System.Globalization** namespace bellow the **@inherits** directive. To do this, add the following line:

**cshtml**

```
@inherits System.Web.Mvc.WebViewPage<DateTime>

@using System.Globalization;
```

**vbhtml**

```
@inherits System.Web.Mvc.WebViewPage(Of DateTime)

@Imports System.Globalization
```

8. Replace the code block that informs the Date and Time with the following one.

**cshtml**

```
@{
	var now = DateTime.Now;
	<text>
		Current time is @now
	</text>
	<text>
		Current time is @Model.ToString(CultureInfo.CurrentUICulture)
	</text>
}
```

**vbhtml**

```
@Code
	Dim now = DateTime.Now
	@<text>
		Current time is @now
	</text>
	@<text>
		Current time is @Model.ToString(CultureInfo.CurrentUICulture)
	</text>
End Code
```

**Note:** The @Model is replaced by the type passed to the view: DateTime.Now. Additionally, the ToString method is invoked with the CurrentUICulture to show the Culture configuration for the user interface.

---

**Task 2 – Running the Application**

1. Press **F5** to run the application.
2. The project starts in the Home page. Verify that Razor view is loaded and the Date andTime is displayed:

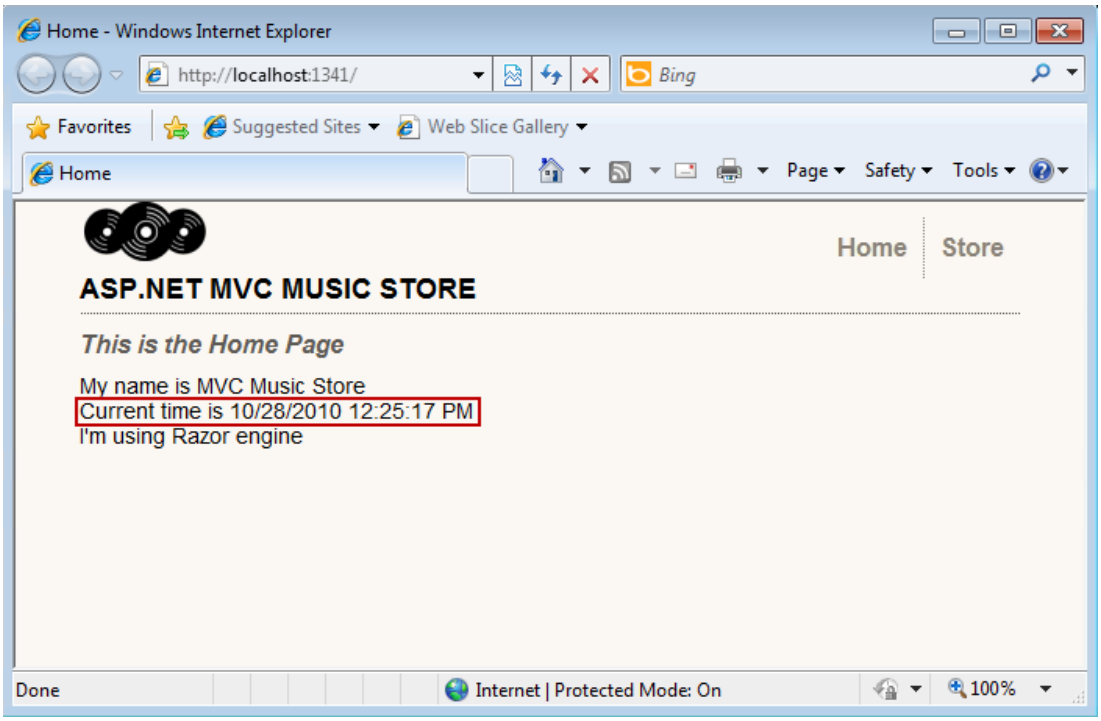


**Figure 11**

*Running the Home page*

---

### Task 3 – Adding a View Template in Razor

In this task, you will add a new Index.cshtml|vbhtml view, in this case, using the **@model** directive. This directive, represents an improvement for the **@inherits**, simplifying the use of strongly-typed model classes in the view. You simply write **@model *StrongModelType*** at the top of the cshtml file, replacing the @inherits directive. You will add a view template in Razor for the Store Index page, which shows a list of Genres. At the end of the task you should get the same functionality you had when using ASPX templates, but with a simplified code.

1. Before creating the new View template, you should build the project so that the **Add View Dialog** knows about the **ViewModel** class to use. Build the project by selecting the **Debug** menu item and then **Build MvcMusicStore**.

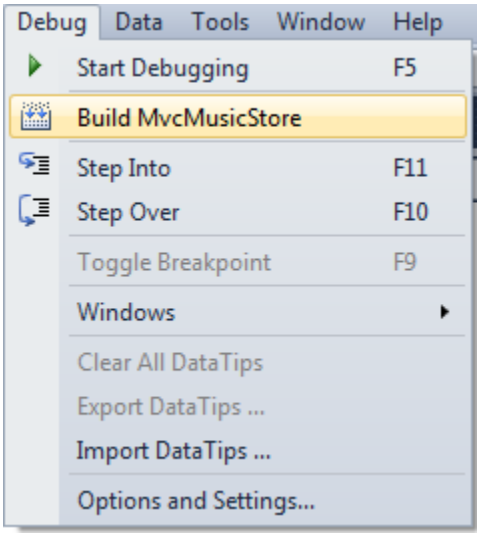


**Figure 12**

*Building the project in Visual Web Developer 2010*

2. Open the **StoreController.[cs|vb]** file and right-click inside the Index Action method to display the context menu. Select **Add View** menu option to display the dialog box.

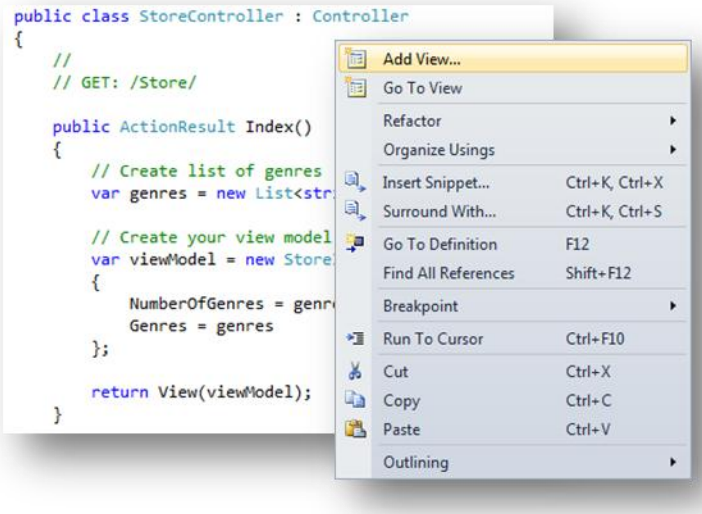


**Figure 13**

*Adding a View from the Action method – C#*

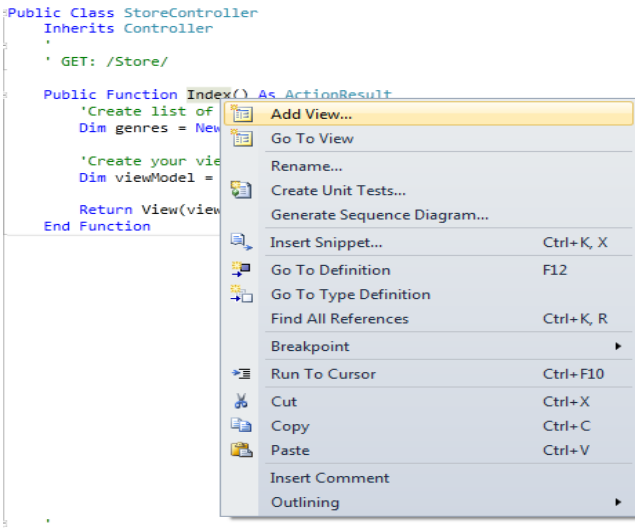


**Figure 14**

*Adding a View from the Action method - VB*

3. The **Add View** dialog appears. Fill the dialog box according to the following values:

    a. Check "Create a strongly-typed view".

    b. Choose "**StoreIndexViewModel (MvcMusicStore.ViewModels)**" at Model class.

    c. Check "Use a layout or master page" and verify that Razor Layout template path is "**~/Views/Shared/Site.cshtml|vbhtml**" by clicking the "**…**" button and then click **Add**.

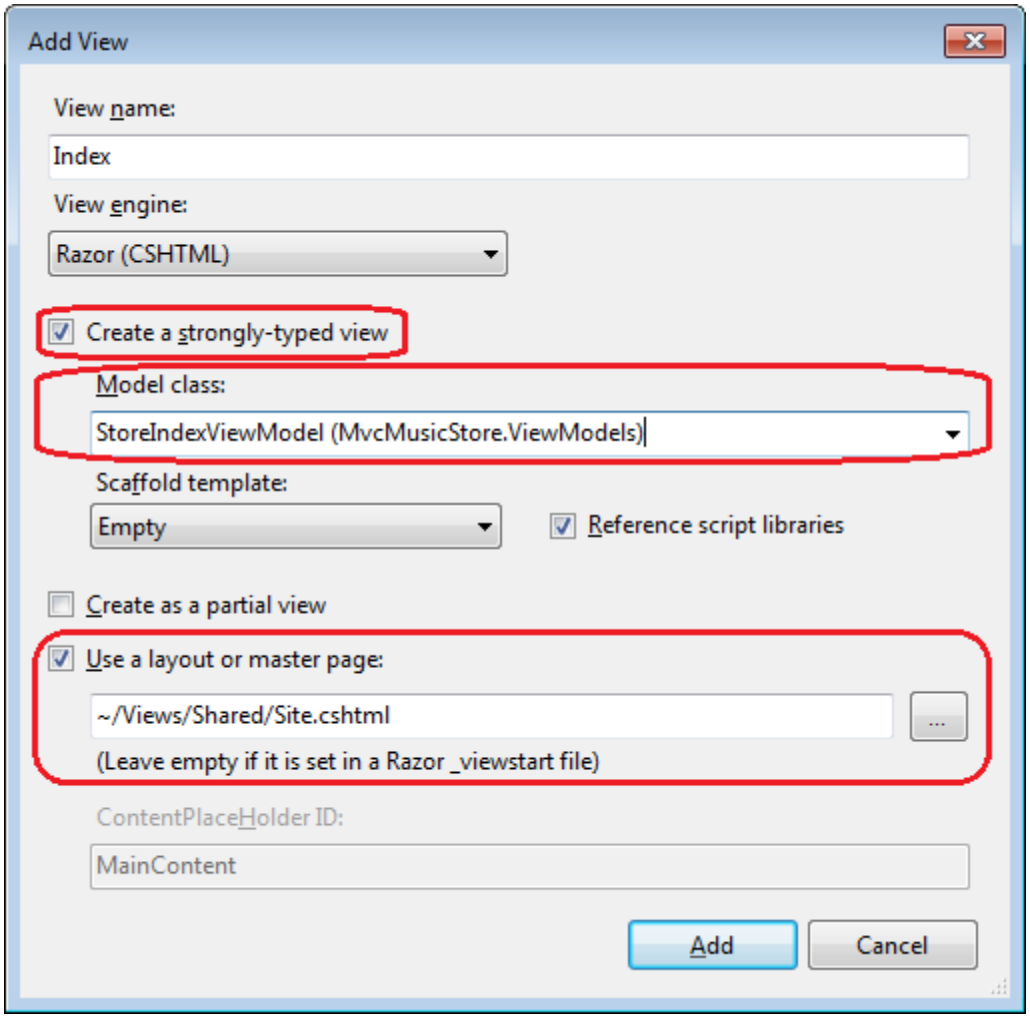


**Figure 15**

*Adding a new Razor View dialog box – C#*

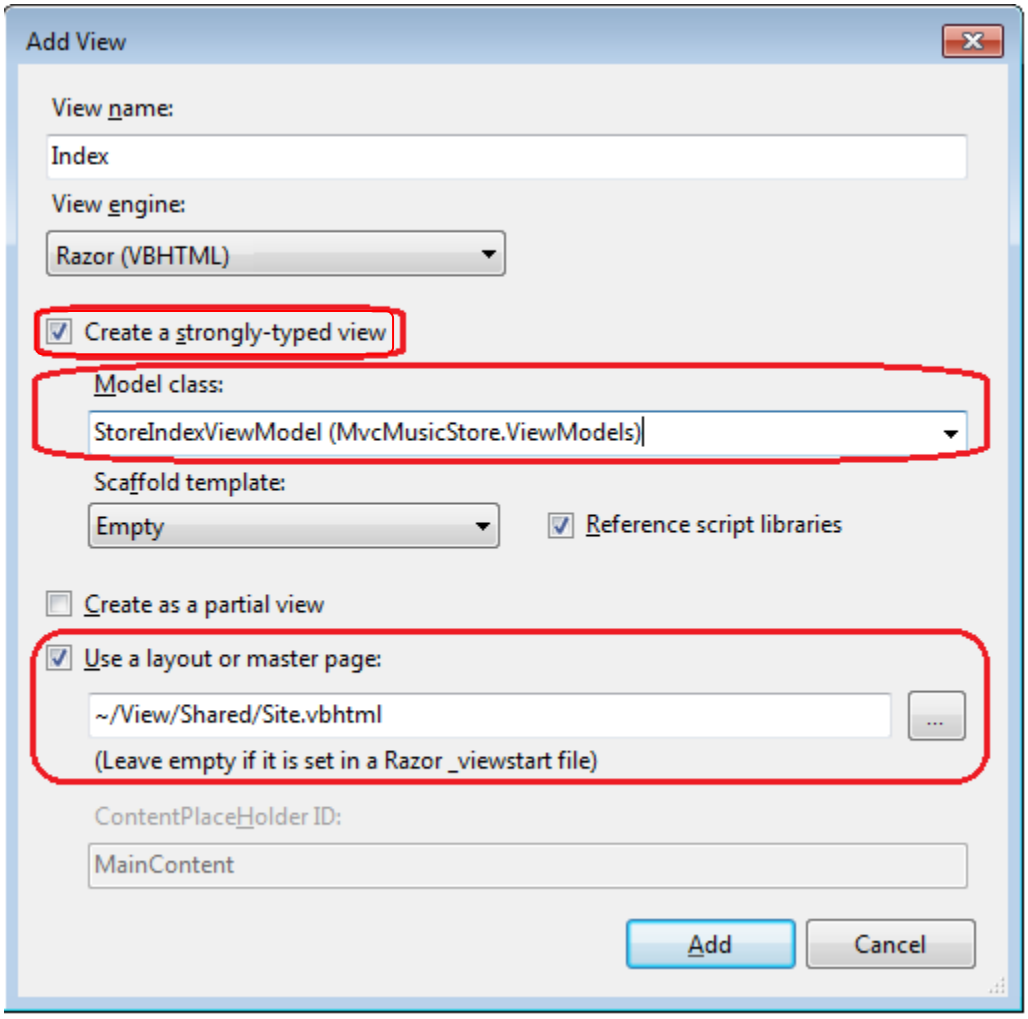


**Figure 16**

*Adding a new Razor View dialog box – VB*

4. Visual Studio generates an **Index.cshtml|vbhtml** Razor view template inside the **Views\Store** folder with the following structure:

**cshtml**

```
@model MvcMusicStore.ViewModels.StoreIndexViewModel

@{
    ViewBag.Title = "Index";
    Layout = "~/Views/Shared/Site.cshtml";
}

<h2>Index</h2>
```

**vbhtml**

```
@modeltype MvcMusicStore.ViewModels.StoreIndexViewModel

@Code
    ViewBag.Title = "Index"
    Layout = "~/Views/Shared/Site.vbhtml"
End Code

<h2>Index</h2>
```

> **Note:** The **@model *StrongModelType*** directive at the top of the View specifies its base class. By default Razor will make the view to derive from **System.Web.Mvc.WebViewPage<TModel>**.
>
> Inside the @{ } block , the view sets the layout that will be used, and also sets the value for the **ViewBag.Title** element.

5. Add logic in the Index .cshtml|vbhtml to show the list of genres from the model. To do this, replace the content of the generated file with the following one:

**cshtml**

```
@model MvcMusicStore.ViewModels.StoreIndexViewModel

@{
    ViewBag.Title = "Browse Genres";
    Layout = "~/Views/Shared/Site.cshtml";
}

<h2>Browse Genres</h2>

<p>Select from @Model.NumberOfGenres</p>

@foreach (string genreName in Model.Genres) {
<li>
    @Html.ActionLink(genreName, "Browse", new { genre = genreName }, null)
</li>
}
```

**vbhtml**

```
@model MvcMusicStore.ViewModels.StoreIndexViewModel

@Code
    ViewBag.Title = "Browse Genres"
    Layout = "~/Views/Shared/Site.vbhtml"
End Code
```

```
<h2>Browse Genres</h2>

<p>Select from @Model.NumberOfGenres</p>

@For Each genreName in Model.Genres
@<li>
    @Html.ActionLink(genreName, "Browse", New With{Key.genre = genreName },
null)
</li>
Next genreName
```

> **Note:** The foreach loops through the genres and generates an **ActionLink** to the Browse action method. This method is implemented in the provided solution, but its view will be created in task 5.

---

**Task 4 – Running the Application**

1. Press **F5** to run the application.
2. The project starts in the Home page. Change the URL to **/Store** to verify that Razor view is loaded:

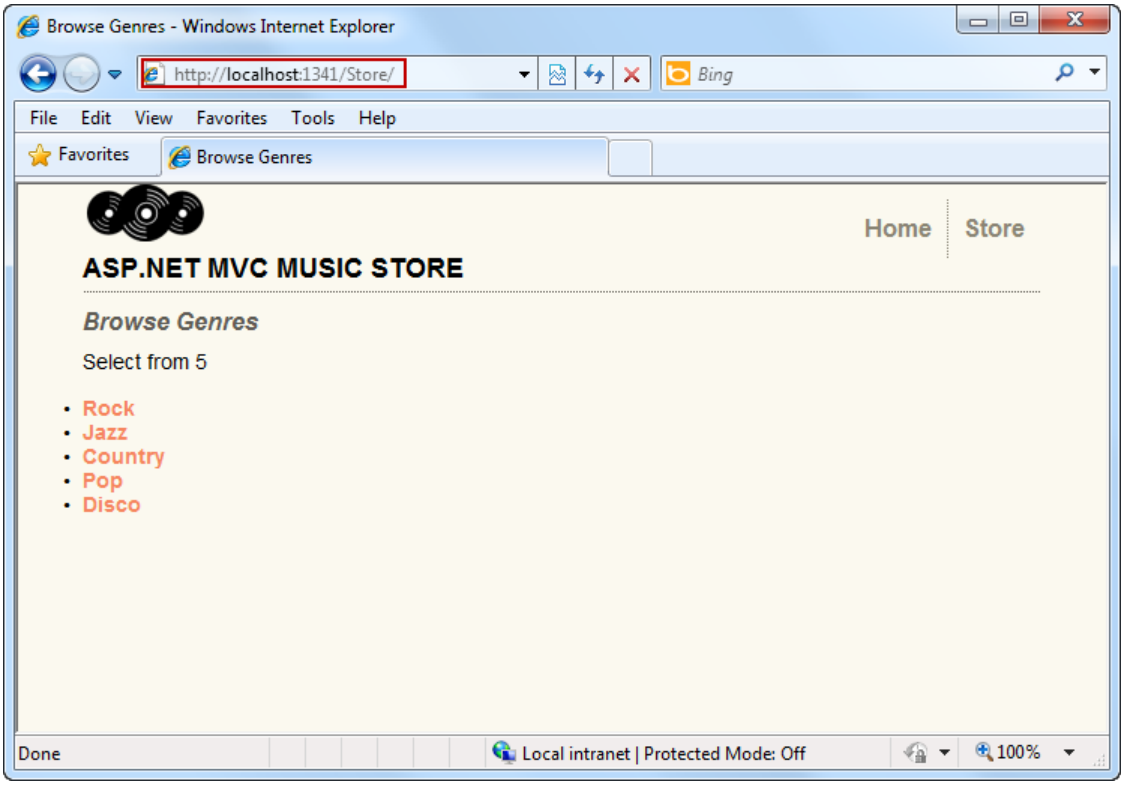


**Figure 17**

*Browsing genres with a Razor View*

---

**Task 5 – Creating Browse CSHTML View and Adding Details CSHTML View**

In the previous task, you have learned how to add a Razor view. Now you will include the views that are missing from the original Music Store Application: **Browse** and **Details**.

1. The cshtml files are included in the **Source\Assets\cshtml** folder of this Lab. In order to add them to the application, drag them from a **Windows Explorer** window into the **Solution Explorer** in Visual Web Developer Express, as shown below:

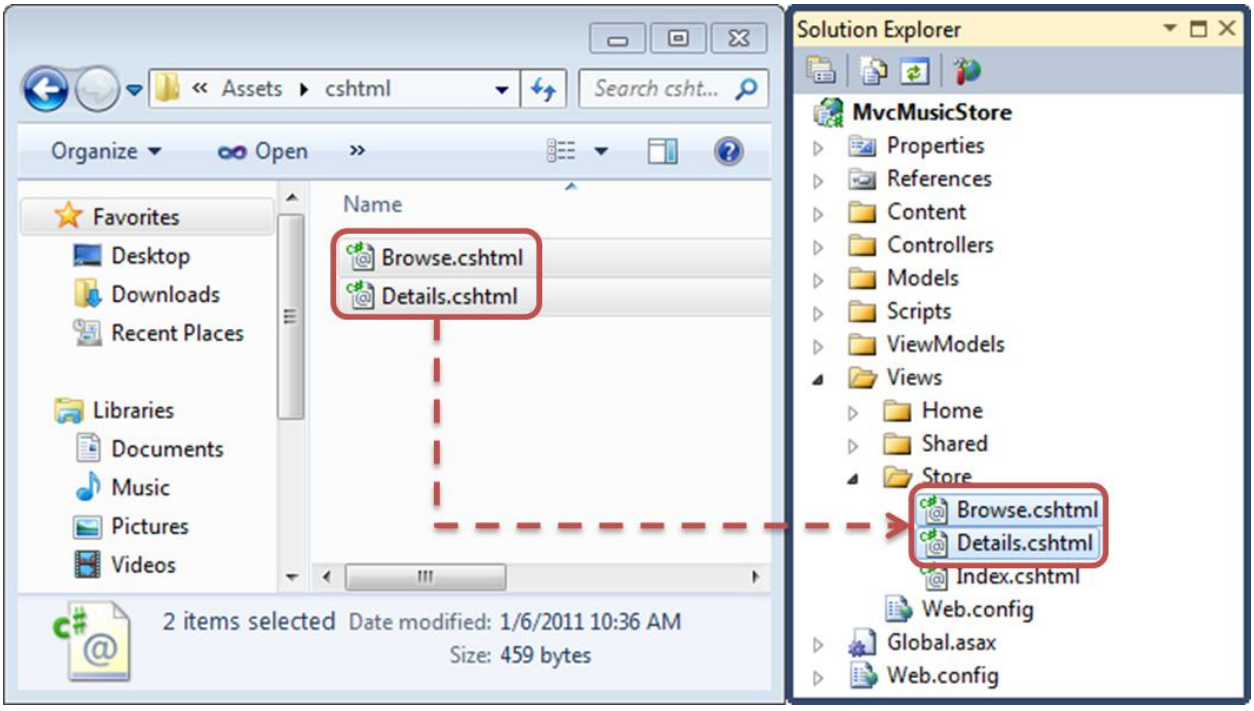


**Figure 18**

*Dragging views – C#*

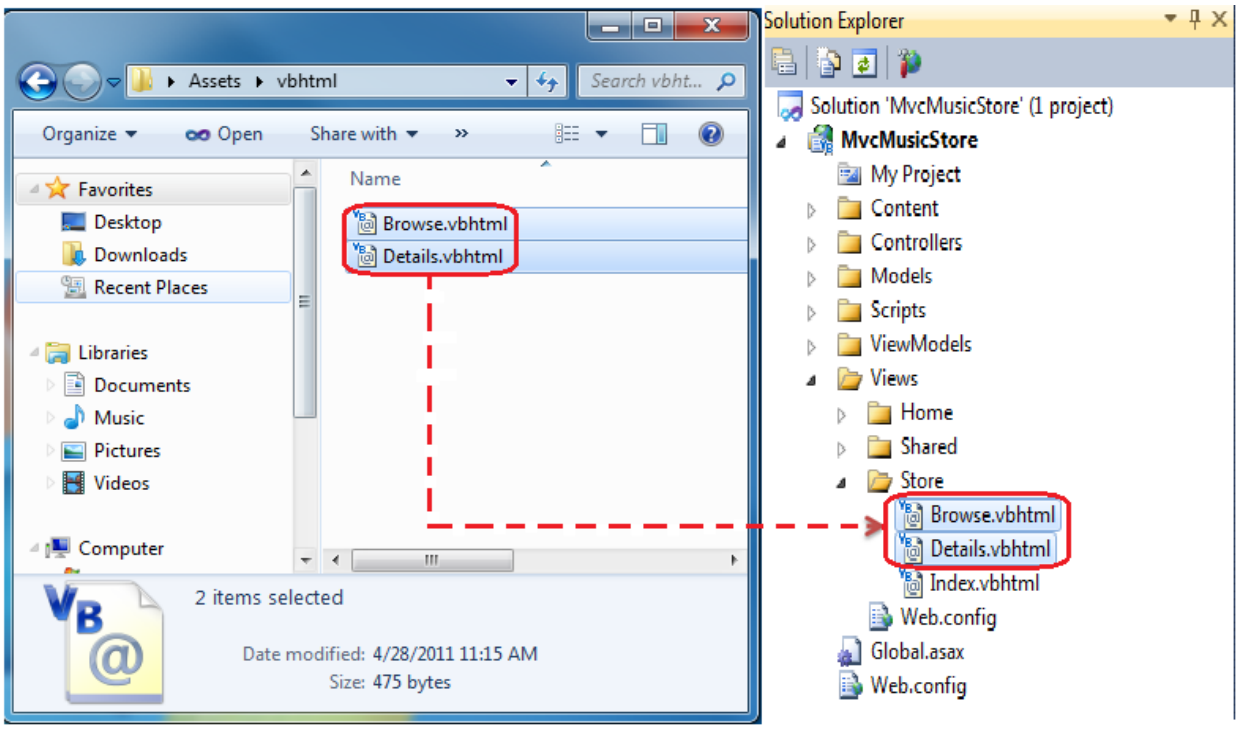


**Figure 19**

*Dragging views - VB*

2. Visual Studio will add **Browse.cshtml|vbhtml** and **Details.cshtml|vbhtml** Razor views.

**cshtml – Browse.cshtml**

```
@model MvcMusicStore.ViewModels.StoreBrowseViewModel

@{
    ViewBag.Title = "Browse Albums";
    Layout = "~/Views/Shared/Site.cshtml";
}

    <h2>Browsing Genre: @Model.Genre.Name</h2>

    <ul>

    @foreach (var album in Model.Albums) {
       <li>@album.Title</li>
    }
    </ul>
```

**vbhtml – Browse.vbhtml**

```
@modeltype MvcMusicStore.ViewModels.StoreBrowseViewModel
```

```
@Code
    ViewBag.Title = "Browse Albums"
    Layout = "~/Views/Shared/Site.vbhtml"
End Code

    @<h2>Browsing Genre: @Model.Genre.Name</h2>

    <ul>

    @For Each album in Model.Albums
       @<li>@album.Title</li>
    Next album
    </ul>
```

**Note:** The Browse.cshtml|vbhtml page will loop all the albums for the selected genre, listing their titles.

**cshtml – Details.cshtml**

```
@model MvcMusicStore.Models.Album

@{
    ViewBag.Title = "Details";
    Layout = "~/Views/Shared/Site.cshtml";
}

   <h2>Album: @Model.Title </h2>
```

**vbhtml – Details.vbhtml**

```
@modeltype MvcMusicStore.Models.Album

@Code
    ViewBag.Title = "Details"
    Layout = "~/Views/Shared/Site.vbhtml"
End Code

   <h2>Album: @Model.Title </h2>
```

**Note:** The Details.cshtml|vbhtml page displays the album title.

**Task 6 – Running the Application**

In this task, you will run the application to verify that Razor views are displayed as expected.

1. Press **F5** to run the application.
2. The project starts in the Home page. Change the URL to **/Store** and click on **Rock** genre to verify that **Details** view can be loaded, with the same appearance as the .aspx pages:

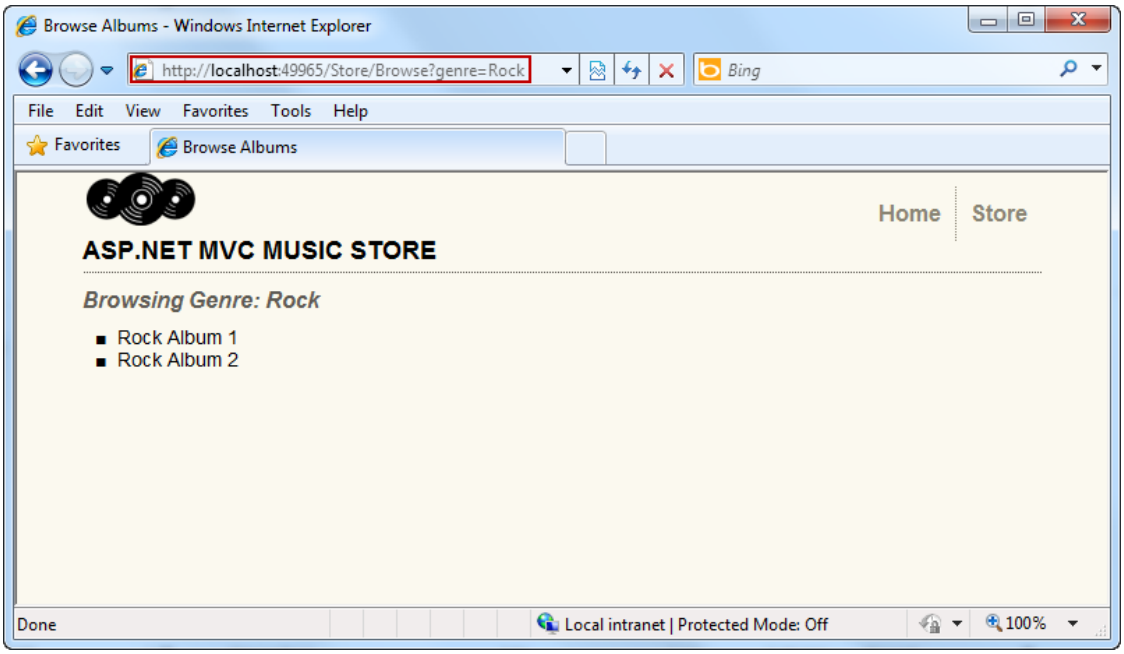


**Figure 20**

*Browsing genres albums with Razor View*

3. Change the URL to **/Store/Details/5** to verify an album's information.

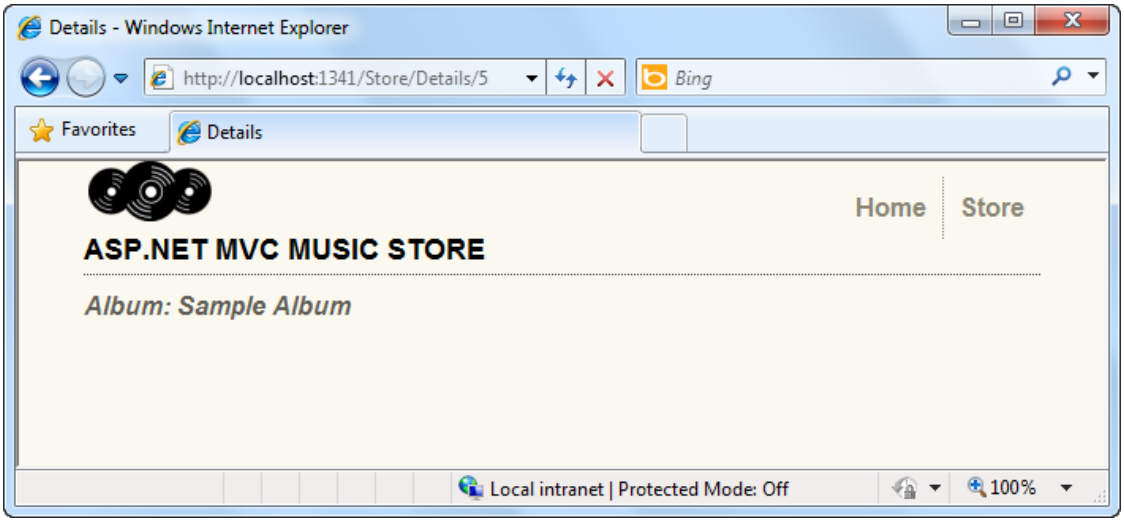


**Figure 21**

*Browsing album details with Razor View*

## Next Step

Exercise 3: Creating and Consuming Razor Helpers

# Exercise 3: Creating and Consuming Razor Helpers

ASP.NET Web Pages Helpers are components that provide certain functionality, and can be accessed by any Razor or ASP.NET Web Page by using a single line of code. You can build your own helpers, which are stored as .cshtml|vbhtml razor files, or use the built-in helpers from ASP.NET or Razor.

Using helpers will increase your code reusability, organization and maintainability because you will be creating separate units for specific features. In addition, you could take advantage of built-in or third party helpers to easily interact with multimedia, social networks, security, email and data grids.

In this exercise, you will learn how to consume and create Razor helpers in your solution. To do that, you will first use a WebImage helper to display images with watermarks, and then you will use a WebGrid helper to show the album data. At the end, you will create a custom helper with Razor syntax to show the current date.

### Task 1 – Consuming the Razor WebImage Helper

In this task, you will learn how to use Razor WebImage Helper to display an image and add a watermark on it.

**Note: WebImage** Razor Helper provides functionality to **manage images** in your page.

The usage of WebImage helper requires two steps:

1. You need to work with Razor syntax and create **a variable that will contain the image**:

```
@{ var myImage = WebImage("/Content/myImage.jpg")
   // Work with the image…
 }
@Code Dim myImage = WebImage("/Content/myImage.jpg")
   ' Work with the image…
 End Code
```

2. Then, in order to load the image in the page, you have to show the variable that contains the image inside an **HTML <img/>** tag:

```
<img src="@myImage"/>
```

Here you will find a reference of the additional **WebImage** Razor functions that are available:

- ***WebImage(string Path):*** Loads an image from the specified path.

- ***WebImage.AddImagesWatermark(string Path)***

- ***WebImage.AddTextWatermark(string text):*** Adds the specified text to the image.

- ***WebImage.FlipHorizontal()/ WebImage.FlipVertical()****:* Flips the image horizontally /vertically.

- ***WebImage.GetImageFromRequest([opt]string postedFileName):*** Gets and image from request after being posted.

- ***WebImage.Resize(width, height):*** Resizes the image to the size specified in pixels.

- ***WebImage.RotateLeft() / WebImage.RotateRight():*** Rotates the image left / right.

- ***WebImage.Save(path):*** Saves the image to the specified path.

You can find more information about Razor WebImage Helper in ASP.NET Web Pages with Razor Syntax Book Beta.

1. If not already open, start Microsoft Visual Web Developer 2010 Express from **Start | All Programs | Microsoft Visual Studio 2010 Express | Microsoft Visual Web Developer 2010 Express**.
2. In the **File** menu, choose **Open Project**. In the Open Project dialog, browse to Source\Ex03-CreatingAndUsingHelpersInRazor\Begin, select **MvcMusicStore.sln** and click **Open**.
3. Open **Index.cshtml|vbhtml** view at **Views/Home** project folder and add WebImage Helper logic to load an image and save it in a different location. To do this, you will first use WebImage() to load an image inside a variable:

**cshtml**

```
@model dynamic

@{
    ViewBag.Title = "Index";
    Layout = "~/Views/Shared/Site.cshtml";
}

    <h2>Index</h2>

@{
      var image = new WebImage(@"~\Content\Images\home-showcase.png");
```

```
      var imagePath = @"Content/Images/newimage.png";
      var saveImagePath = @"~\Content\Images\newimage.png";
      image.Save(saveImagePath);
}
<img src="@imagePath" alt="image with watermark"/>
```

**vbhtml**

```
@modeltype Object

@Code
    ViewBag.Title = "Index"
    Layout = "~/Views/Shared/Site.vbhtml"
End Code

    <h2>Index</h2>
```

```
@Code
      Dim image = New WebImage("~\Content\Images\home-showcase.png")
      Dim imagePath = "Content/Images/newimage.png"
      Dim saveImagePath = "~\Content\Images\newimage.png"
      image.Save(saveImagePath)
End Code
<img src="@imagePath" alt="image with watermark"/>
```

> **Note:** ImagePath separator character is "**/**" to allow browser compatibility.

4. Add a watermark to the image by using AddWatermark() method:

**cshtml**

```
…
@{
      var image = new WebImage(@"~\Content\Images\home-showcase.png");
      image = image.AddImageWatermark(@"~\Content\Images\logo.png",
horizontalAlign: "Left", verticalAlign: "Top");
      var imagePath = @"Content/Images/newimage.png";
      var saveImagePath = @"~\Content\Images\newimage.png";
      image.Save(saveImagePath);
}
<img src="@imagePath" alt="image with watermark"/>
```

**vbhtml**

```
…
@Code
```

```
      Dim image = new WebImage("~\Content\Images\home-showcase.png")
      image = image.AddImageWatermark("~\Content\Images\logo.png",
horizontalAlign: "Left", verticalAlign: "Top")
      Dim imagePath = "Content/Images/newimage.png"
      Dim saveImagePath = "~\Content\Images\newimage.png"
      image.Save(saveImagePath)
End Code
<img src="@imagePath" alt="image with watermark"/>
```

**Note: AddImageWatermark** method accepts more parameters than the ones used here, which are optional and provide a default value. Here is the complete method definition:

**C#**

```
AddImageWatermark(
  string watermarkImageFilePath,
  string format (optional, default is null),
  int width (optional, default is 0, in pixel units),
  int height (optional, default is 0, in pixel units),
  string horizontalAlign (optional, default is "Right"),
  string verticalAlign (optional, default is "Bottom"),
  int opacity (optional, default is 100, in percentage),
  int padding (optional, default is 5, in pixel units)
);
```

**Visual Basic**

```
AddImageWatermark(String watermarkImageFilePath,
                  String format (optional, TypeOf default Is Nothing),
                  Integer width (optional, TypeOf default Is 0, in
                  pixel units),
                  Integer width (optional, TypeOf default Is 0, in
                  pixel units),
                  Integer height (optional, TypeOf default Is 0, in
                  pixel units),
                  String horizontalAlign (optional, TypeOf default Is
                  "Right"),
                  String verticalAlign (optional, TypeOf default Is
                  "Bottom"),
```

```
                Integer opacity (optional, TypeOf default Is 100, in
                percentage),
                Integer)
```

**Task 2 – Running the Application**

In this task, you will run the application to verify that the image with the watermark is loaded.

1. Press **F5** to run the application.
2. The application starts in the Home page and displays the image with the watermark:

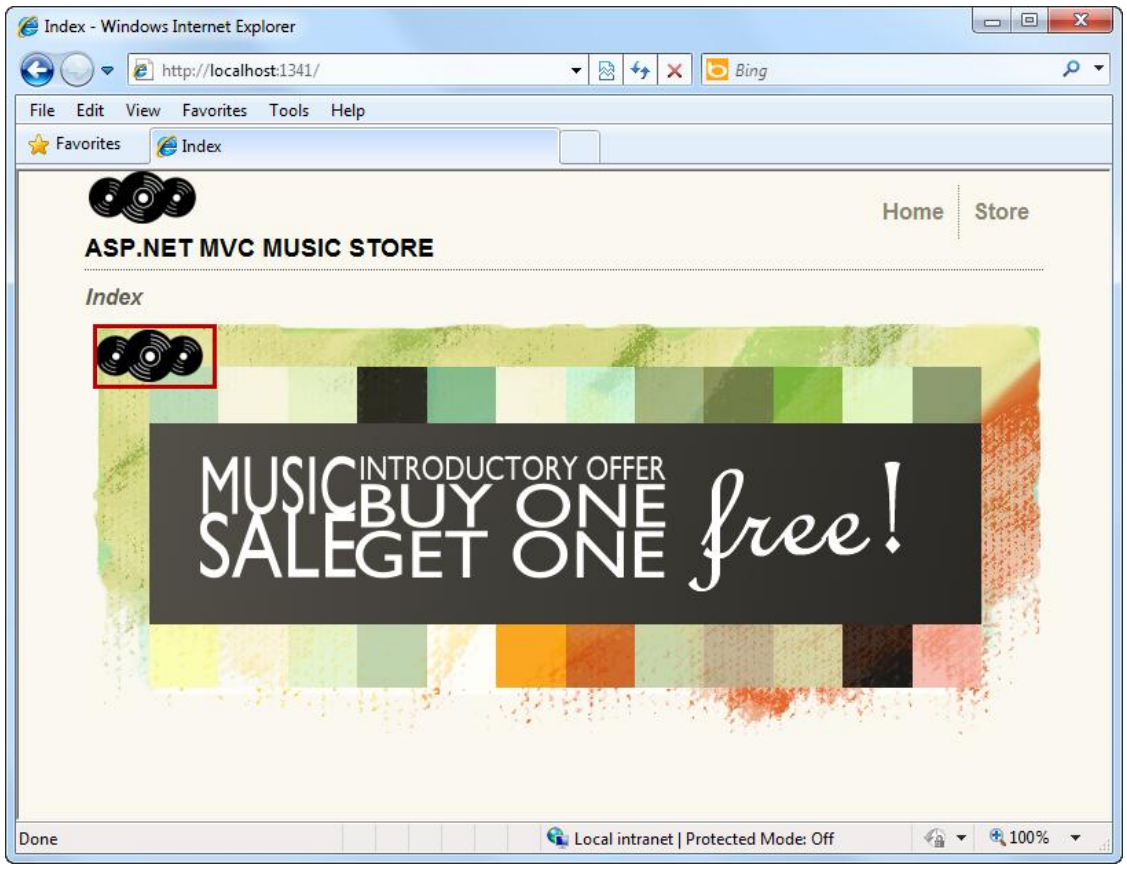


**Figure 22**

*Image and watermark using Razor WebImage helper*

**Task 3 – Adding a WebGrid Helper to Browse View**

In this task, you will add a WebGrid helper to show a grid of albums inside Browse view.

**Note:** The **WebGrid** Helper renders an HTML table that displays data from the model or from a database. It supports data formatting, paging and sorting.

It generates a collection of Columns and parameters within the given data source, which is rendered as an HTML table that can be accessed after the grid is generated.

To **use** a grid **from a model**, you will first need to generate the grid and assign it to a variable with the WebGrid creation method, which receives the model data structure as a required parameter:

**C#**

```
@{
     var grid = new WebGrid(Model.Elements);
}
```

**Visual Basic**

```
@Code
     Dim grid = New WebGrid(Model.Elements)
End Code
```

Then, to **render** the grid in the page you will have to use a GetHtml() method:

```
@{
     var grid = new WebGrid(Model.Elements);
     grid.GetHtml();
}
```

**Visual Basic**

```
@Code
     Dim grid = new WebGrid(Model.Elements)
     grid.GetHtml()
End Code
```

Here you will find method references if you want to use additional features:

**C#**

```
  WebGrid(dynamic IEnumerable<object> source (required field),
     IEnumerable<string> columnNames (default is null),
     string defaultSort (default is null),
     int rowsPerPage (default it 10),
     bool canPage (default is true),
```

```
     bool canSort (default is true),
     string ajaxUpdateContainerId (default is null),
     string fieldNamePrefix (default is null),
     string pageFieldName (default is null),
     string selectionFieldName (default is null),
     string sortFieldName (default is null),
     string sortDirectionFieldName  (default is null)
  );

Visual Basic
WebGrid(dynamic IEnumerable(Of Object) source (required field),
         IEnumerable(Of String) columnNames (TypeOf default Is Nothing),
         String defaultSort (TypeOf default Is Nothing), Integer rowsPerPage
         (default it 10),
         Boolean canPage (TypeOf default Is True),
         Boolean canSort (TypeOf default Is True),
         String ajaxUpdateContainerId (TypeOf default Is Nothing),
         String fieldNamePrefix (TypeOf default Is Nothing),
         String pageFieldName (TypeOf default Is Nothing),
         String selectionFieldName (TypeOf default Is Nothing),
         String sortFieldName (TypeOf default Is Nothing),
         String sortDirectionFieldName (TypeOf default Is Nothing))

C#
   WebGrid.GetHtml(string tableStyle (default is null),
        string headerStyle (default is null),
        string footerStyle (default is null),
        string rowStyle (default is null),
        string alternatingRowStyle (default is null),
        string selectedRowStyle (default is null),
        bool displayHeader (default is true),
        bool fillEmptyRows (default is false),
        string emptyRowCellValue (default is null),
        IEnumerable<WebGridColumn> columns (default is null),
        IEnumerable<string> exclusions (default is null),
```

```
        WebGridPagerModes mode (default is 3),
        string firstText (default is null),
        string previousText (default is null),
        string nextText (default is null),
        string lastText (default is null),
        int numericLinksCount (default is null)
  );

Visual Basic
WebGrid.GetHtml(String tableStyle (TypeOf default Is Nothing),
                String headerStyle (TypeOf default Is Nothing),
                String footerStyle (TypeOf default Is Nothing),
                String rowStyle (TypeOf default Is Nothing),
                String alternatingRowStyle (TypeOf default Is Nothing),
                String selectedRowStyle (TypeOf default Is Nothing),
                Boolean displayHeader (TypeOf default Is True),
                Boolean fillEmptyRows (TypeOf default Is False),
                String emptyRowCellValue (TypeOf default Is Nothing),
                IEnumerable(Of WebGridColumn) columns (TypeOf default Is
                Nothing),
                IEnumerable(Of String) exclusions (TypeOf default Is
                Nothing),
                WebGridPagerModes mode (TypeOf default Is 3),
                String firstText (TypeOf default Is Nothing),
                String previousText (TypeOf default Is Nothing),
                String nextText (TypeOf default Is Nothing),
                String lastText (TypeOf default Is Nothing),
                Integer numericLinksCount (TypeOf default Is Nothing))
```

1. Open **Browse.cshtml|vbhtml** view at **Views/Store** project folder.
2. Create a WebGrid that uses the model to show the albums. Additionally the grid will be sorted by name and show 10 elements. To do this, you will create the grid using **Model.Albums** and two extra parameters: **defaultSort** and **rowsPerPage**.

**cshtml**

```
@model MvcMusicStore.ViewModels.StoreBrowseViewModel

@{
    ViewBag.Title = "Browse Albums";
    Layout = "~/Views/Shared/Site.cshtml";
}

@{
    var grid = new WebGrid(Model.Albums, defaultSort: "Name", rowsPerPage:
10);
}
```

**vbhtml**

```
@modeltype MvcMusicStore.ViewModels.StoreBrowseViewModel

@Code
    ViewBag.Title = "Browse Albums"
    Layout = "~/Views/Shared/Site.vbhtml"
End Code

@Code
    Dim grid = new WebGrid(Model.Albums, defaultSort:= "Name", rowsPerPage:=
10)
End Code
```

3. Display the **WebGrid** specifying two values in the Column collection: Title and Artist.

**cshtml**

```
@model MvcMusicStore.ViewModels.StoreBrowseViewModel

@{
    ViewBag.Title = "Browse Albums";
    Layout = "~/Views/Shared/Site.cshtml";
}

@{
var grid = new WebGrid(Model.Albums, defaultSort: "Name", rowsPerPage: 10);
}

@grid.GetHtml(columns: grid.Columns(
      grid.Column("Title"),
      grid.Column("Artist")
))
```

**vbhtml**

```
@modeltype MvcMusicStore.ViewModels.StoreBrowseViewModel

@Code
    ViewBag.Title = "Browse Albums"
    Layout = "~/Views/Shared/Site.vbhtml"
End Code

@Code
Dim grid = New WebGrid(Model.Albums, defaultSort:= "Name", rowsPerPage:= 10)
End Code

@grid.GetHtml(columns:= grid.Columns(
      grid.Column("Title"),
      grid.Column("Artist")
))
```

---

**Task 4 – Running the Application**

In this task, you will run the application to verify that the WebGrid is loaded and shows album data as expected.

1. Press **F5** to run the application.
2. The project starts at Home. Change the URL to **/Store/Browse** to verify that the Browse view loads the WebGrid as configured in the last steps:

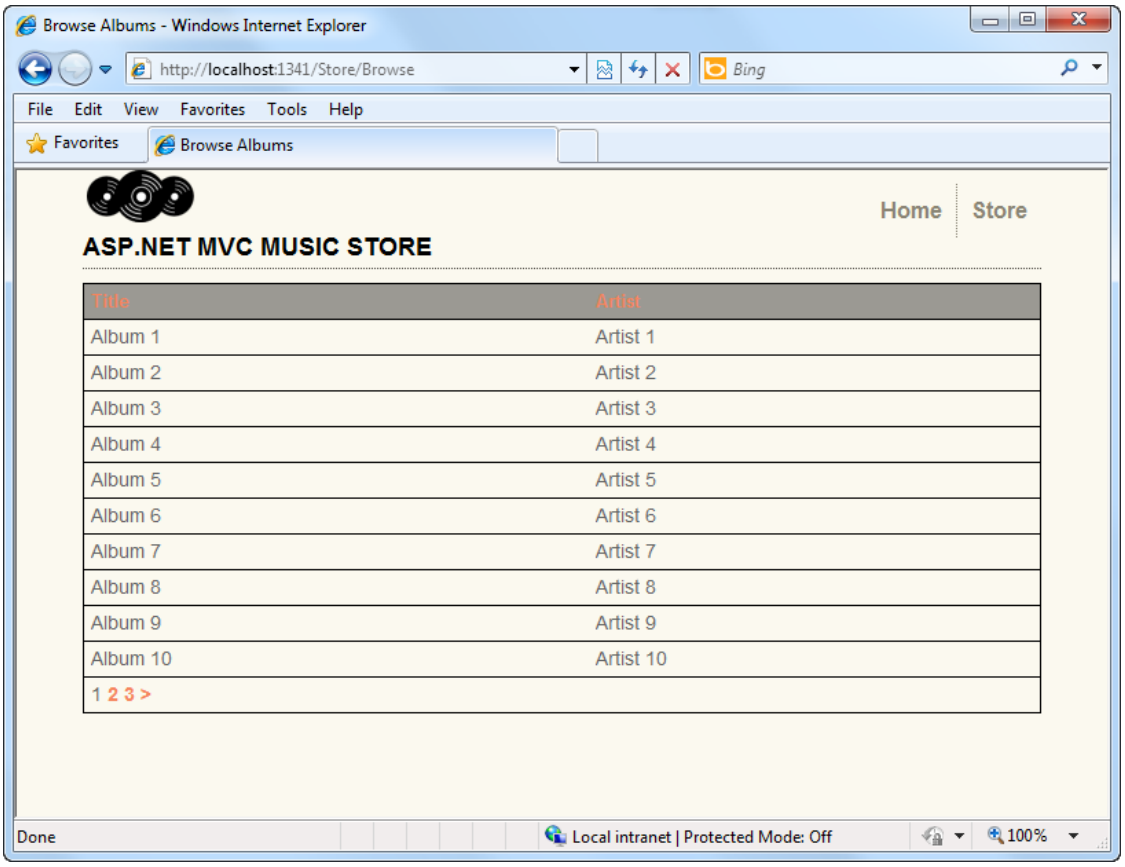


**Figure 23**

*A Razor WebGrid showing the albums*

### Task 5 – Creating a Custom Helper

Up to now you have been using Razor third party helpers. In this task, you will learn how to create your own custom helper in Razor. In order to do that you will add to the solution a helper with the name **"ShowTime()"** that will show the current date and time.

1. Add a new element into App_Code by doing right-click on App_Code folder, and selecting "**Add|New Item**".

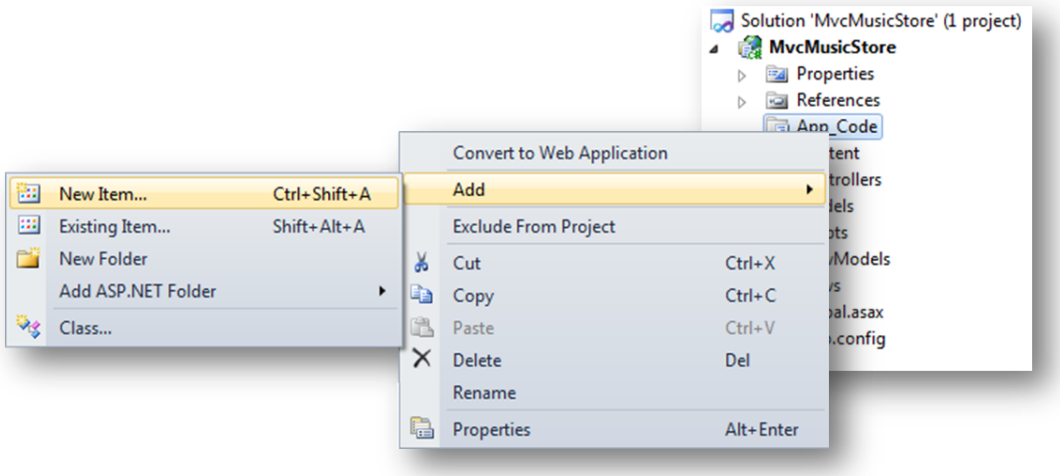


**Figure 24**

*Adding a new item into App_Code*

2. In the Add New Item dialog, select from MVC 3 templates, **MVC 3 Partial Page (Razor)**. Use the name **MyHelpers.cshtml|vbhtml** and click **Add**. This will create an empty Razor file with the name MyHelpers.cshtml|vbhtml.

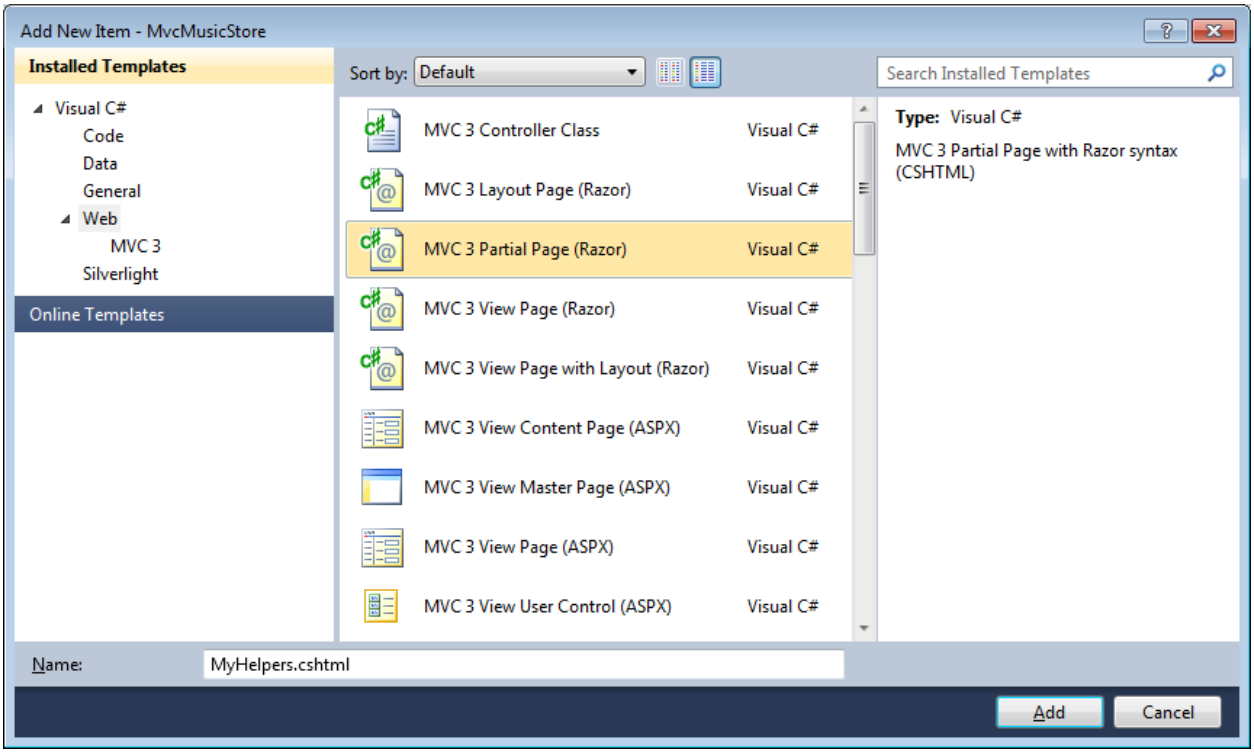


**Figure 25**

*Adding a new Razor Helper – C#*

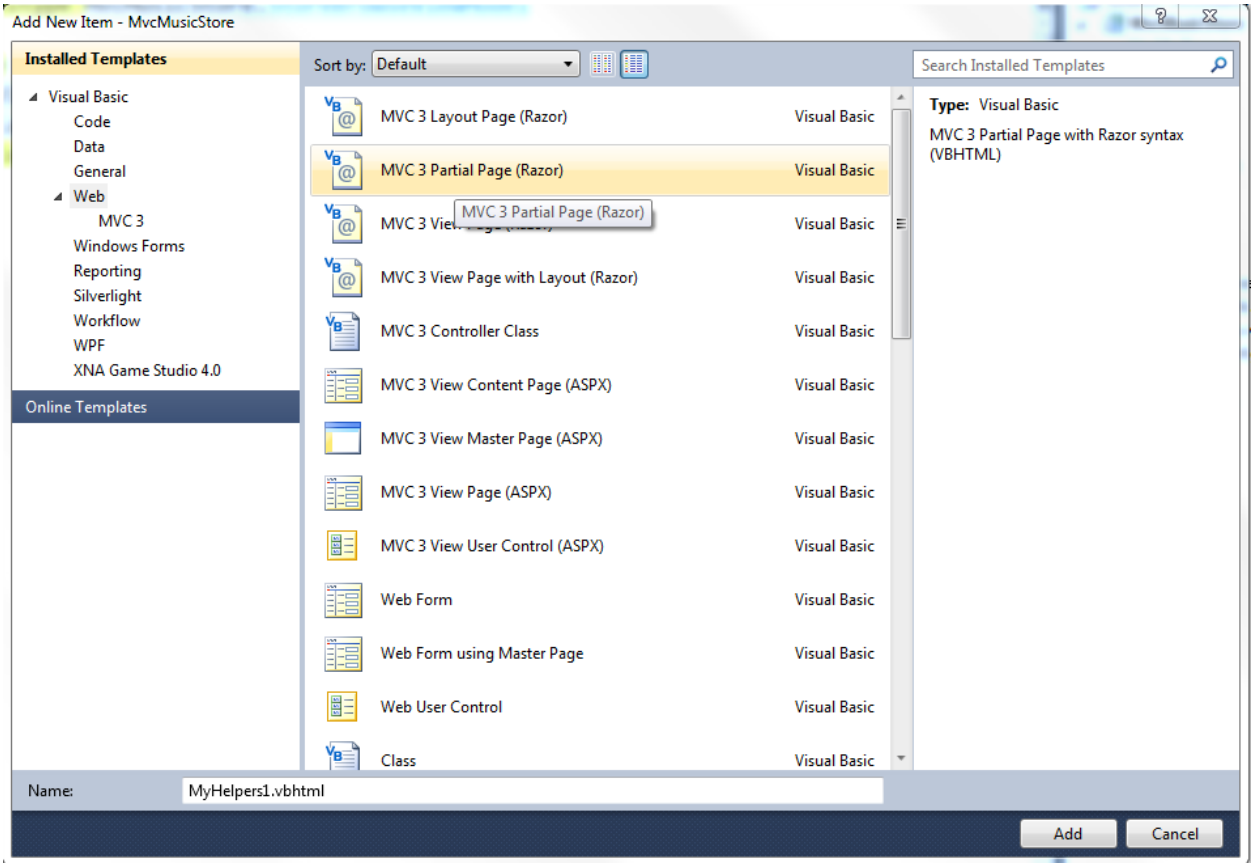


**Figure 26**

*Adding a new Razor Helper - VB*

3. Open **MyHelpers.cshtml|vbhtml** and declare the helper:

**cshtml**

```
@helper ShowTime(){
}
```

**vbhtml**

```
@helper ShowTime()
End helper
```

4. In ShowTime helper, include HTML code to show current date. To do that, you will use the @DateTime.Now code inside an HTML markup block:

**cshtml**

```
@helper ShowTime(){
    <div>
        Current Time is @DateTime.Now
```

```
    </div>
}
```

**vbhtml**

```
@helper ShowTime()
    @<div>
        Current Time is @DateTime.Now
    </div>
End helper
```

5. Open **Index.cshtml|vbhtml** View in the project folder **Views/Home** to call the helper:

**cshtml**

```
@model dynamic

@{
    ViewBag.Title = "Index";
    Layout = "~/Views/Shared/Site.cshtml";
}

<h2>Index</h2>

@MyHelpers.ShowTime()

@{
      var image = new WebImage(@"~\Content\Images\home-showcase.png");
      image = image.AddImageWatermark(@"~\Content\Images\logo.png",
horizontalAlign: "Left", verticalAlign: "Top");
      var imagePath = @"Content/Images/newimage.png";
      var saveImagePath = @"~\Content\Images\newimage.png";
      image.Save(saveImagePath);
}
<img src="@imagePath" alt="image with watermark"/>
```

**vbhtml**

```
@modeltype Object

@Code
    ViewBag.Title = "Index"
    Layout = "~/Views/Shared/Site.vbhtml"
End Code

<h2>Index</h2>

@MyHelpers.ShowTime()
```

```
@Code
      Dim image = New WebImage("~\Content\Images\home-showcase.png")
      image = image.AddImageWatermark("~\Content\Images\logo.png",
horizontalAlign:= "Left", verticalAlign:= "Top")
      Dim imagePath = "Content/Images/newimage.png"
      Dim saveImagePath = "~\Content\Images\newimage.png"
      image.Save(saveImagePath)
End Code
<img src="@imagePath" alt="image with watermark"/>
```

**Task 6 – Running the Application**

In this task, you will run the application to verify that the helper is working as expected.

1. Press **F5** to run the application.
2. The project starts at Home:

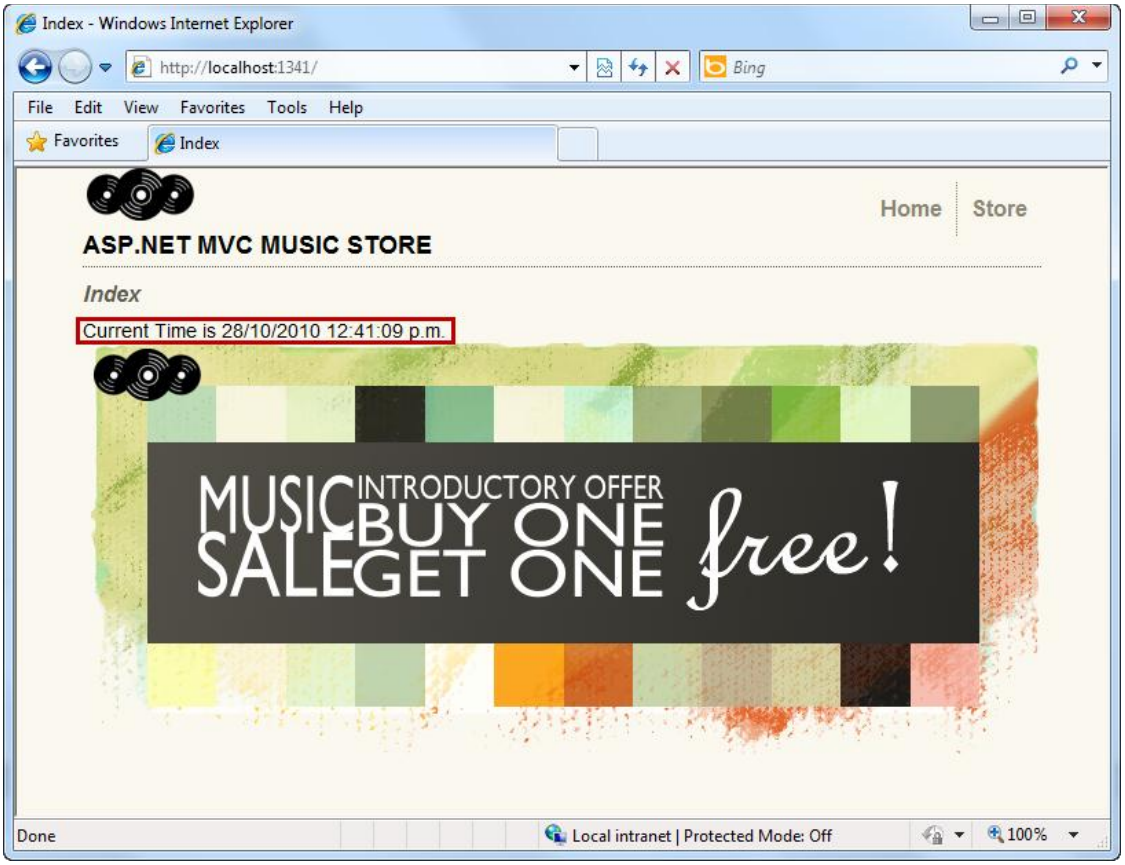


**Figure 27**

*Current time displayed with a custom helper*

## Next Step

Summary

# Summary

By completing this Hands-On Lab you have learned the fundamentals of Razor for MVC Views:

- How to create and use a Razor View
- How to use Razor Layout pages
- How to use Razor WebGrid and WebImage Helpers
- How to use and create Razor Helpers