25

Customizing Visual Studio for ASP.NET MVC

This chapter covers:

* Creating custom T4 templates
* Using custom T4 templates
* Adding test project templates

As you saw earlier, specific tooling within Visual Studio makes building ASP.NET MVC applications faster. We will look at two quick ways of customizing these tools.

6.5.1 Creating custom T4 templates

If you right click on an action, you’ll see an option to open the Add View dialog, shown in figure 6.11. In this dialog, you can choose the name of the view, the view model type, and the master page. If you select a strongly typed view, you have the option of choosing an automatic view template. The options are Empty, List, Create, Details, Delete. Figure 6.11 shows us selecting Create for our view content and Product for our view model.

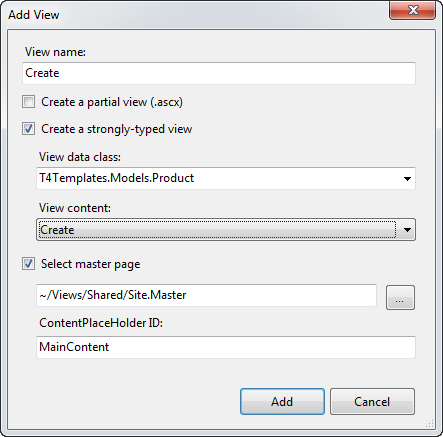


Figure 6.11 The Add View dialog allows you to auto-generate scaffolding for your model.

The options in the View Content dropdown list are T4 templates that are located in

C:\Program Files (x86)\Microsoft Visual Studio 9.0\Common7\IDE\ItemTemplates

NOTE

T4 templates are a little-known feature of Visual Studio. They are code generation template processors built into Visual Studio. T4 templates allow you to customize how files are generated using a familiar syntax.

If we press Add, we’re given a complete form, generated for us by Visual Studio using the default template. Our view now looks like listing 6.27

Listing 6.27 The autogenerated Create view based on the Product object

<%@ Page Title="" Language="C#" MasterPageFile="~/Views/Shared/Site.Master"

Inherits="System.Web.Mvc.ViewPage<T4Templates.Models.Product>" %>

<asp:Content ID="Content1" ContentPlaceHolderID="TitleContent" runat="server">

Create

</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">

<h2>

Create</h2>

<%= Html.ValidationSummary("Create was unsuccessful. Please correct the errors and try again.") %>

<% using (Html.BeginForm())

{%>

<fieldset>

<legend>Fields</legend>

<p>

<label for="Id">

Id:</label>

<%= Html.TextBox("Id") %>

<%= Html.ValidationMessage("Id", "\*") %>

</p>

<p>

<label for="Name">

Name:</label>

<%= Html.TextBox("Name") %>

<%= Html.ValidationMessage("Name", "\*") %>

</p>

<p>

<label for="Description">

Description:</label>

<%= Html.TextBox("Description") %>

<%= Html.ValidationMessage("Description", "\*") %>

</p>

<p>

<label for="ActiveDate">

ActiveDate:</label>

<%= Html.TextBox("ActiveDate") %>

<%= Html.ValidationMessage("ActiveDate", "\*") %>

</p>

<p>

<label for="RetireDate">

RetireDate:</label>

<%= Html.TextBox("RetireDate") %>

<%= Html.ValidationMessage("RetireDate", "\*") %>

</p>

<p>

<input type="submit" value="Create" />

</p>

</fieldset>

<% } %>

<div>

<%=Html.ActionLink("Back to List", "Index") %>

</div>

</asp:Content>

As you can see, lots of code is generated for us. It contains a basic form, with fields corresponding to the object, complete with validation, Submit button, and back link. This can get us started building the application quickly. Of course this is just a starting point, and you’re free to customize it from here. This template is static, and you can create a different, application-specific template for the Create view.

Add a folder in your project called CodeTemplates. Into this folder, copy the contents of the default template folder. You can create subfolders corresponding to the different types of templates (figure 6.12).

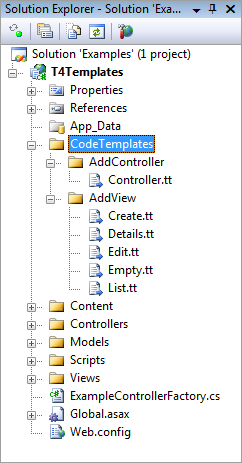


Figure 6.12 Copy the templates from C:\Program Files (x86)\Microsoft Visual Studio 9.0\Common7\IDE\Templates\CSharp\Web\MVC into your project under a CodeTemplates folder to customize them.

These templates will be effective for the current project only. You are free to alter the templates here for your project. You can also add more items to this list. Adding another .tt file in this folder will enable it for selection in the Add View dialog as show in figure 6.13.

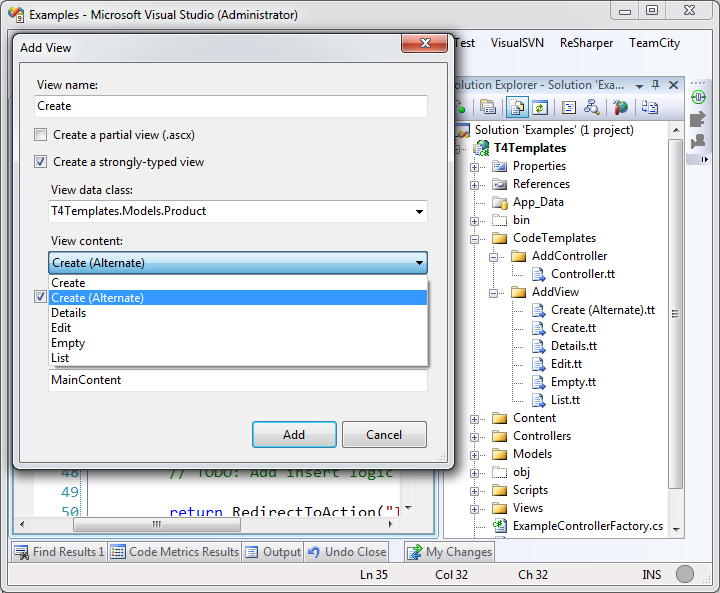


Figure 6.13 Adding new template files in the Add View folder enables them for selection in the Add View dialog.

The templates themselves are fairly complex. Here is an excerpt from the Controller.tt template:

<#@ template language="C#" HostSpecific="True" #>

<#@ output extension="cs" #>

<#

MvcTextTemplateHost mvcHost = (MvcTextTemplateHost)(Host);

#>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using System.Web.Mvc.Ajax;

namespace <#= mvcHost.NameSpace #>

{

public class <#= mvcHost.ItemName #> : Controller

{

//

// GET: /<#= mvcHost.ControllerRootName #>/

...... more ....

As you can see, code blocks are denoted by <# #> blocks. Each template has a Host property that contains basic context information. For MVC templates, this is actually of type MvcTextTemplateHost, so we can see here that the template is casting the Host property and storing it in a variable called mvcHost for use later in the template.

T4 templates can be a little intimidating but you can do a lot of things with them. If you are interested in customizing the templates, download Visual T4 Editor for Visual Studio 2008 Community Edition (free) from Clarius Consulting. This will give you syntax highlighting, which is really helpful when you find yourself writing code that writes code! The tool can be downloaded at <http://www.visualt4.com/downloads.html>. To learn more about T4 Template syntax and the ASP.NET MVC integration, check out <http://blogs.msdn.com/webdevtools/archive/2009/01/29/t4-templates-a-quick-start-guide-for-asp-net-mvc-developers.aspx>.

6.5.2 Adding a custom test project template to the new project wizard

When you first create an ASP.NET MVC project, you’re eventually greeted with the dialog shown in figure 6.14:

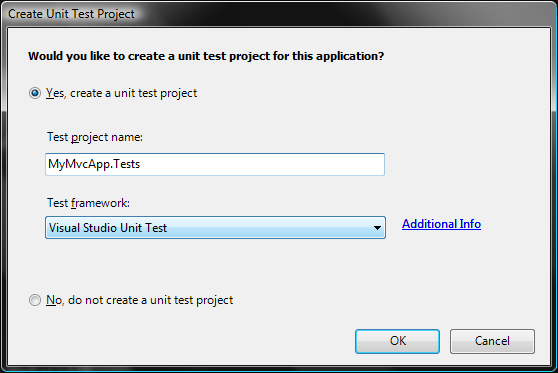


Figure 6.14 When you create a new project, you are asked if you want to create a unit test project.

Unfortunately, the only available test framework that is provided out of the box is the Visual Studio Unit Test framework. Developers who are experienced with testing will no doubt prefer NUnit, MbUnit, or xUnit.NET. There is hope! You can add your framework of choice to this dialog box (and simultaneously implement a custom project template).

The first step is to create a project that represents what you want when you create new ASP.NET MVC applications with the test project included. Make sure all third-party references (such as NUnit, Rhino Mocks) are set to Copy Local. Then go to File > Export Template. Follow the wizard here, which will result in a single zip file. Copy this zip file to:

C:\Program Files\Microsoft Visual Studio\9.0\Common7\IDE\ProjectTemplates\CSharp\Test.

(If you’re running on a 64-bit machine, then adjust the path to C:\Program Files (x86) accordingly). Now that you’ve got the template in the right place, close all instances of Visual Studio, open up the Visual Studio 2008 Command Prompt (as Administrator if UAC is enabled), and run:

devenv /installvstemplates

This will take a few seconds. Now that you have a project template installed into Visual Studio, open regedit and navigate to

HKEY\_LOCAL\_MACHINE\Software\Microsoft\VisualStudio\9.0\MVC\TestProjectTemplates

Here you’ll find the default Visual Studio Unit Test key. To create a custom entry, make a new key here, and add the following String values:

* Package–Leave blank unless you have a custom Visual Studio package GUID to register here.
* Path–Usually CSharp\Test.
* TestFrameworkName–The name that you want to appear in the Unit Test Framework dropdown.
* AdditionalInfo–A URL that provides the user more information about your framework or template. When the user clicks on Additional Info, the browser will navigate to this URL.
* Template–The name of the zip file that contains the template.

Figure 6.15 shows a new template installed in this location.

Note

On 64-bit machines–like the one we are using–the registry path is slightly different (…SOFTWARE\Wow6432Node\Microsoft…). In addition, the Program Files path is actually C:\Program Files (x86)\. Be sure to adjust accordingly for your system as shown in figure 6.15.

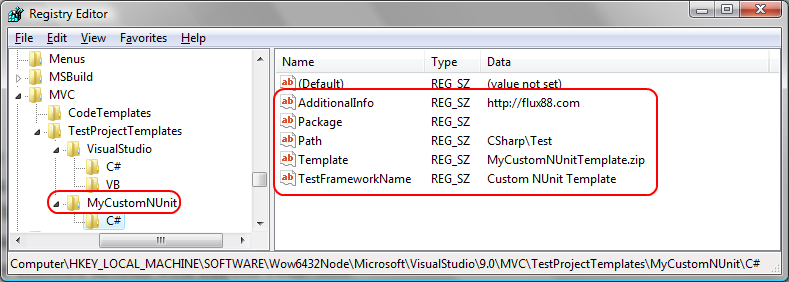


Figure 6.15 Adding a registry entry for a new custom test project template. Note that this registry path is for 64-bit machines.

With all of this in place, we can launch Visual Studio, create a new ASP.NET MVC Web Application project, and be greeted with this the message shown in figure 6.16.

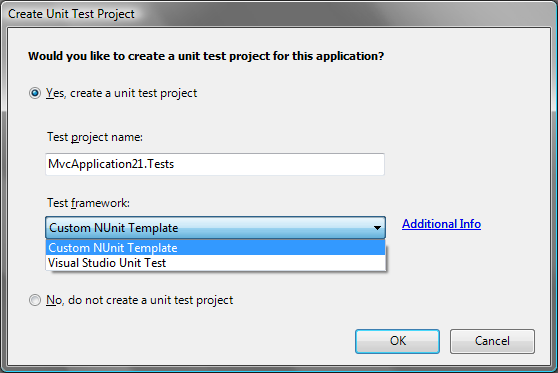


Figure 6.16 Our new test template is now available in the Create Unit Test Project dialog box.

6.6 Summary

In this chapter, you have seen some of the extension points in the ASP.NET MVC Framework. You learned how to create a custom IRouteHandler, to add behavior surrounding the life of MVC requests. You learned how to create custom base controllers to encapsulate and reuse functionality. You also learned how to leverage Dependency Injection and custom controller factories for building these controllers. To accommodate cross controller concerns, you learned how to use attributes to decorate actions with custom behaviors. We implemented a naïve view engine to demonstrate the moving parts, and also discovered how to customize Visual Studio to evolve with you as you adopt new styles for developing applications.

Hopefully you noticed how easy it was to extend the framework. Because most objects that you interact with are either interfaces or abstract base classes, the framework allows you to completely (or almost completely) substitute behavior for your own. At this level of flexibility the ASP.NET MVC Framework shines. Running parallel to the ASP.NET MVC project is the open source project called MvcContrib (<http://mvccontrib.org>). This is the playground for customizations and extensions that people find useful. Your authors recommend that you examine MvcContrib regularly for extensions that might be useful to you (and contribute back if they would be useful to others!).

The next chapter will use some of these extension points and cover tools and techniques for letting the framework scale in the face of complex web applications.