**Walkthrough: Creating a Basic MVC Project with Unit Tests in Visual Studio**

This walkthrough shows you how to create an ASP.NET MVC application in Visual Studio. In this walkthrough, you will create and run the sample MVC application. Then you will customize the application by adding a controller and a view.

In addition, this walkthrough shows how to use test-driven development (TDD). In the walkthrough, you create a project that contains unit tests for the MVC application.

A Visual Studio project with source code is available to accompany this topic: [Download](http://go.microsoft.com/fwlink/?LinkId=155337).

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifPrerequisites

In order to complete this walkthrough, you will need:

* Microsoft Visual Studio 2008 Service Pack 1 or later.

|  |
| --- |
| **Description: NoteNote** |
| Visual Studio Standard Edition and Visual Web Developer Express do not do not support unit-test projects. You can use these versions of Visual Studio to run the parts of this walkthrough that pertain to creating and running an ASP.NET MVC project. However, you will not be able to work with unit tests as described in this walkthrough. |

* The ASP.NET MVC 2 framework. If you have installed Visual Studio 2010, the ASP.NET MVC 2 is already installed on your computer. To download the most up-to-date version of the framework, see the [ASP.NET MVC download](http://go.microsoft.com/fwlink/?LinkId=179610) page.

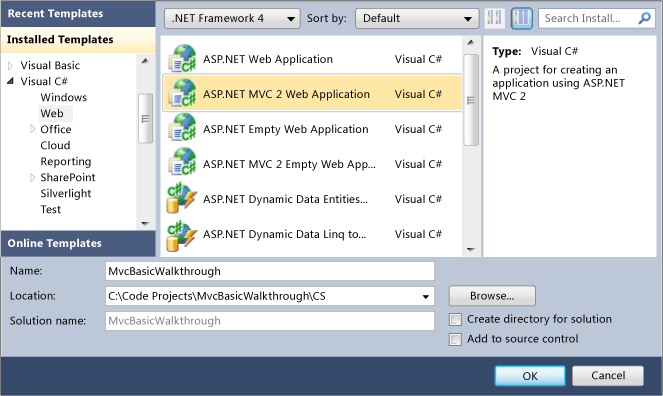
Description: http://i.msdn.microsoft.com/Global/Images/clear.gifCreating a New MVC Project

To begin, you will create a new ASP.NET MVC project.

**To create a new MVC project**

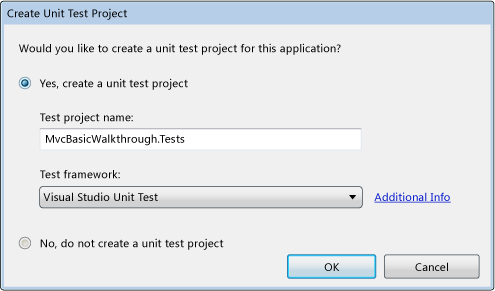
1. On the **File** menu, click **New Project**.

The **New Project** dialog box is displayed.



1. In the upper-right corner, make sure that **.NET Framework 3.5** is selected.
2. Under **Project types**, expand either **Visual Basic** or **Visual C#**, and then click **Web**.
3. Under **Visual Studio installed templates**, select **ASP.NET MVC 2 Web Application**.
4. In the **Name** box, enter **MvcBasicWalkthrough**.
5. In the **Location** box, enter a name for the project folder.
6. If you want the name of the solution to differ from the project name, enter a name in the **Solution Name** box.
7. Select **Create directory for solution**.
8. Click **OK**.

The **Create Unit Test Project** dialog box is displayed.



|  |
| --- |
| **Description: NoteNote** |
| If you are using the Standard or Express editions of Visual Studio, the **Create Unit Test Project** dialog box is not displayed. Instead, the new MVC application project is generated without a test project. |

1. Select **Yes, create a unit test project**.

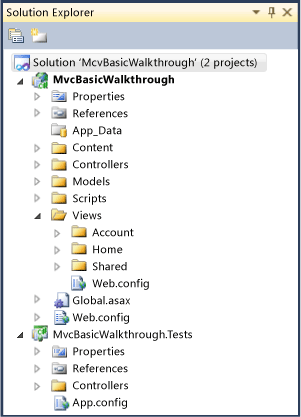
By default, the name of the test project is the application project name with "Tests" added. However, you can change the name of the test project. By default, the test project will use the Visual Studio Unit Test framework. For information about how to use a third-party test framework, see [How to: Add a Custom ASP.NET MVC Test Framework in Visual Studio](http://msdn.microsoft.com/en-us/library/dd381614.aspx).

1. Click **OK**.

The new MVC application project and a test project are generated. (If you are using the Standard or Express editions of Visual Studio, the test project is not created.)

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifExamining the MVC Project

The following illustration shows the folder structure of a newly created MVC solution.

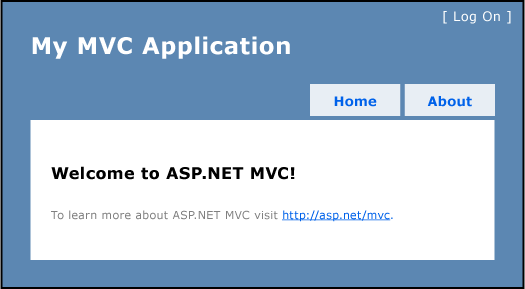


The folder structure of an MVC project differs from that of an ASP.NET Web site project. The MVC project contains the following folders:

* Content, which is for content support files. This folder contains the cascading style sheet (.css file) for the application.
* Controllers, which is for controller files. This folder contains the application's sample controllers, which are named AccountController and HomeController. The AccountController class contains login logic for the application. The HomeController class contains logic that is called by default when the application starts.
* Models, which is for data-model files such as LINQ-to-SQL .dbml files or data-entity files.
* Scripts, which is for script files, such as those that support ASP.NET AJAX and jQuery.
* Views, which is for view page files. This folder contains three subfolders: Account, Home, and Shared. The Account folder contains views that are used as UI for logging in and changing passwords. The Home folder contains an Index view (the default starting page for the application) and an About page view. The Shared folder contains the master-page view for the application.

If you are using an edition of Visual Studio other than Standard or Express, a test project was also generated. The test project has a Controllers folder that contains the HomeControllerTest class. This class has a unit test for each HomeController action method (Index and About).

The newly generated MVC project is a complete application that you can compile and run without change. The following illustration shows what the application looks like when it runs in a browser.



The unit-test project is also ready to compile and run. For this walkthrough, you will add a controller with an action method and a view, and you will add a unit test for the action method.

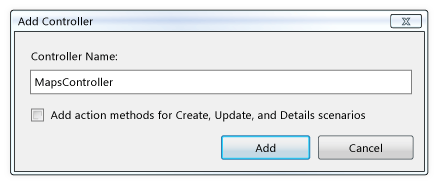
Description: http://i.msdn.microsoft.com/Global/Images/clear.gifAdding a Controller

You will now add a controller that contains logic for downloading city maps from the Microsoft Virtual Earth Web service.

**To add a controller to the MVC project**

1. In **Solution Explorer**, right-click the **Controllers** folder, click **Add**, and then click **Controller**.

The **Add Controller** dialog box is displayed.



1. In the **Name** box, type **MapsController**.

The ASP.NET MVC framework requires controller names to end with "Controller", such as HomeController, GameController, or MapsController.

1. Clear the **Add action methods for Create, Update, and Details scenarios** check box.
2. Click **Add**.

Visual Studio adds the MapsController class to the project and opens it in the editor.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifCreating an Action-Method Stub

To apply test-driven development (TDD) techniques to this project, you should write the unit test for an action method before you write the action method itself. However, if you want your unit test to compile, you must have a stub for the planned action method, which in this walkthrough is ViewMaps.

**To add an action-method stub**

1. Open or switch to the MapsController class.
2. Replace the Index action method with the following code in order to create the ViewMaps action-method stub.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl46_ctl00_ctl00_code');" \o "Copy Code)

Function ViewMaps()

' Add action logic here

Throw New NotImplementedException()

End Function

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl46_ctl00_ctl01_code');" \o "Copy Code)

public ActionResult ViewMaps()

{

// Add action logic here

throw new NotImplementedException();

}

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifAdding Unit Tests for Action Methods

Next, you will add a controller test class to the tests project. In the class, you will add a unit test for the ViewMaps action method. The unit test will fail, because the ViewMaps action-method stub throws an exception. When you finish the action method later in this walkthrough, the test will pass.

**To add unit tests for the action methods**

1. In the tests project, right-click the **Controllers** folder, click **Add**, and then click **Class**.
2. In the **Name** text box, type **MapsControllerTest**.
3. Click **Add**.

Visual Studio adds the MapsControllerTest class to the test project.

1. Open the MapsControllerTest class and enter the following code:

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl47_ctl00_ctl00_code');" \o "Copy Code)

Imports System

Imports System.Collections.Generic

Imports System.Text

Imports System.Web.Mvc

Imports Microsoft.VisualStudio.TestTools.UnitTesting

Imports MvcBasicWalkthrough

<TestClass()> \_

Public Class MapsControllerTest

Private testContextInstance As TestContext

'''<summary>

'''Gets or sets the test context which provides

'''information about and functionality for the current test run.

'''</summary>

Public Property TestContext() As TestContext

Get

Return testContextInstance

End Get

Set(ByVal value As TestContext)

testContextInstance = value

End Set

End Property

<TestMethod()> \_

Public Sub ViewMaps()

' Arrange

Dim controller As MapsController = New MapsController()

' Act

Dim result As ViewResult = controller.ViewMaps()

' Assert

Assert.IsNotNull(result)

End Sub

End Class

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl47_ctl00_ctl01_code');" \o "Copy Code)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System.Web.Mvc;

using MvcBasicWalkthrough;

using MvcBasicWalkthrough.Controllers;

namespace MvcBasicWalkthrough.Tests.Controllers

{

[TestClass]

public class MapsControllerTest

{

[TestMethod]

public void ViewMaps()

{

// Arrange

MapsController controller = new MapsController();

// Act

ViewResult result = controller.ViewMaps() as ViewResult;

// Assert

Assert.IsNotNull(result);

}

}

}

This code defines the unit tests for the action method that you will finish later.

1. In **Solution Explorer**, select the test project and then press CTRL+F5 to run the unit tests.

The test fails, because the ViewMaps action method currently throws an exception.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifAdding Code to the Action Method

You will now add code to the MapsController class for the ViewMaps action method in order to render the Maps view.

**To add code to the action method**

1. Open the MapsController class and replace the ViewMaps action-method stub with the following code in order to render the Maps view:

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl48_ctl00_ctl00_code');" \o "Copy Code)

Function ViewMaps() As ActionResult

Return View()

End Function

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl48_ctl00_ctl01_code');" \o "Copy Code)

public ActionResult ViewMaps()

{

return View();

}

1. Save and close the file.

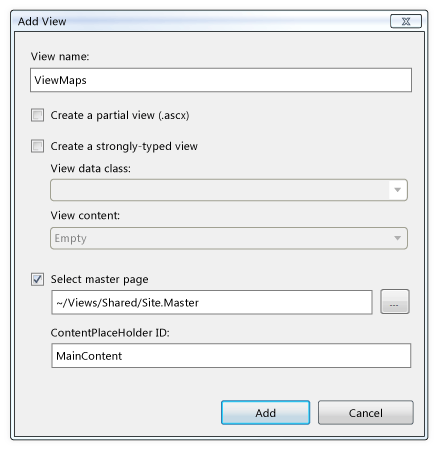
Description: http://i.msdn.microsoft.com/Global/Images/clear.gifAdding a View

Next, you will add a Maps view. To keep the views organized, you will first add a Maps folder under the Views folder.

**To add a page-content view to the MVC project**

1. Open the MapsController class, right-click inside the ViewMaps action method, and then click **Add View**.

The **Add View** dialog box is displayed.



1. In the **View name** box, enter **ViewMaps**.
2. Clear the **Create a partial view (.ascx)** check box and the **Create a strongly-typed view** check box.
3. Select the **Select master page** check box and set the master page to **~/Views/Shared/Site.Master**.
4. Set **ContentPlaceHolder ID** to "MainContent".
5. Click **Add**.

The new view is added to the project in the Maps folder.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifAdding Content to the View

Next, you will add content to the new view.

**To add content to the view**

1. Open ViewMaps.aspx and add the following content inside the **Content** element:

None

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl50_ctl00_ctl00_code');" \o "Copy Code)

<h2>My City Maps</h2>

Select map:

<select onclick="GetMap(value);">

<option value="Seattle">Seattle, WA</option>

<option value="LasVegas">Las Vegas, NV</option>

<option value="SaltLake">Salt Lake City, UT</option>

<option value="Dallas">Dallas, TX</option>

<option value="Chicago">Chicago, IL</option>

<option value="NewYork">New York, NY</option>

<option value="Rio">Rio de Janeiro, Brazil</option>

<option value="Paris">Paris, France</option>

<option value="Naples">Naples, Italy</option>

<option value="Keta">Keta, Ghana</option>

<option value="Beijing">Beijing, China</option>

<option value="Sydney">Sydney, Australia</option>

</select>

<br />

<br />

<div id='earthMap' style="position:relative; width:400px; height:400px;">

</div>

<script charset="UTF-8" type="text/javascript"

src="http://dev.virtualearth.net/mapcontrol/mapcontrol.ashx?v=6.2&mkt=en-us">

</script>

<script type="text/javascript">

var map = null;

var mapID = '';

function GetMap(mapID)

{

switch (mapID)

{

case 'Seattle':

map = new VEMap('earthMap');

map.LoadMap(new VELatLong(47.6, -122.33), 10 ,'i', true);

break;

case 'LasVegas':

map = new VEMap('earthMap');

map.LoadMap(new VELatLong(36.17, -115.14), 10 ,'i' ,true);

break;

case 'SaltLake':

map = new VEMap('earthMap');

map.LoadMap(new VELatLong(40.75, -111.89), 10 ,'i' ,true);

break;

case 'Dallas':

map = new VEMap('earthMap');

map.LoadMap(new VELatLong(32.78, -96.8), 10 ,'i' ,true);

break;

case 'Chicago':

map = new VEMap('earthMap');

map.LoadMap(new VELatLong(41.88, -87.62), 10 ,'i' ,true);

break;

case 'NewYork':

map = new VEMap('earthMap');

map.LoadMap(new VELatLong(40.7, -74), 10 ,'i' ,true);

break;

case 'Rio':

map = new VEMap('earthMap');

map.LoadMap(new VELatLong(-22.91, -43.18), 10 ,'i' ,true);

break;

case 'Paris':

map = new VEMap('earthMap');

map.LoadMap(new VELatLong(48.87, 2.33), 10 ,'i' ,true);

break;

case 'Naples':

map = new VEMap('earthMap');

map.LoadMap(new VELatLong(40.83, 14.25), 10 ,'i' ,true);

break;

case 'Keta':

map = new VEMap('earthMap');

map.LoadMap(new VELatLong(5.92, 0.983), 10 ,'i' ,true);

break;

case 'Beijing':

map = new VEMap('earthMap');

map.LoadMap(new VELatLong(39.91, 116.39), 10 ,'i' ,true);

break;

case 'Sydney':

map = new VEMap('earthMap');

map.LoadMap(new VELatLong(-33.86, 151.21), 10 ,'i' ,true);

}

}

</script>

This markup defines a drop-down list for selecting a map and the JavaScript logic for retrieving the selected map from the Microsoft Virtual Earth Web service.

1. Save and close the file.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifAdding a Tab to the Master-Page Menu

You will now add an item to the master-page menu that calls the ViewMaps action method.

**To add a tab to the master-page menu**

1. In the Shared folder, open the Site.master file and locate the unordered list (**ul** element) in the **div** element whose ID is "menucontainer".
2. Add the following code to the list between the **Index** and **About Us** tabs:

None

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl51_ctl00_ctl00_code');" \o "Copy Code)

<li><%= Html.ActionLink("My City Maps", "ViewMaps", "Maps")%></li>

The ActionLink method is a helper method that links to an action method. It takes the following parameters: the text for the link, the name of the action method, and the name of the controller.

1. Save and close the file.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifTesting the MVC Application

You can now test the application.

**To test the MVC application**

1. On the **Test** menu, click **Run**, and then click **All Tests in Solution**.

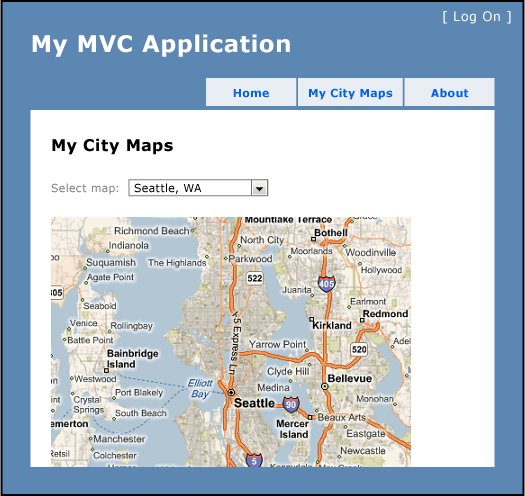
The results are displayed in the **Test Results** window. This time the tests pass.

1. In **Solution Explorer**, select the walkthrough project and press CTRL+F5 to run the application.

The Index.aspx page is displayed, which includes the tabs that are defined in the master page.

1. Click the **My City Maps** tab.

The **My City Maps** page is displayed. Select any map to see it displayed.



Description: http://i.msdn.microsoft.com/Global/Images/clear.gifNext Steps

This walkthrough gives you a first look at creating an MVC application and working with unit tests in ASP.NET MVC. From here, you might want to learn more about the ASP.NET MVC framework. The following list suggests topics for additional learning.

* For more information about MVC controllers, see [Controllers and Action Methods in ASP.NET MVC Applications](http://msdn.microsoft.com/en-us/library/dd410269.aspx).
* For more information about MVC views, see [Views and UI Rendering in ASP.NET MVC Applications](http://msdn.microsoft.com/en-us/library/dd410123.aspx).
* For more information about MVC models, see [Models and Validation in ASP.NET MVC](http://msdn.microsoft.com/en-us/library/dd410405.aspx).
* For more information about URL routing, see [ASP.NET Routing](http://msdn.microsoft.com/en-us/library/cc668201.aspx).