What is Layout View in ASP.NET MVC

An application may contain a specific UI portion that remains the same throughout the application, such as header, left navigation bar, right bar, or footer section. ASP.NET MVC introduced a Layout view which contains these common UI portions so that we don't have to write the same code in every page. The layout view is the same as the master page of the ASP.NET webform application.

For example, an application UI may contain a header, left menu bar, right bar, and footer section that remains the same on every page. Only the center section changes dynamically, as shown below.

[Chart, treemap chart

Description automatically generated](https://www.tutorialsteacher.com/Content/images/mvc/layout-view-1.png)Sample Application UI Parts



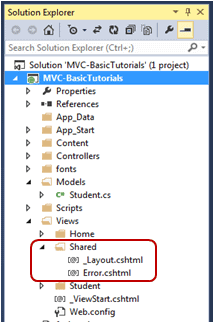
The layout view allows you to define a common site template, which can be inherited in multiple views to provide a consistent look and feel in multiple pages of an application. The layout view eliminates duplicate coding and enhances development speed and easy maintenance. The layout view for the above sample UI would contain a Header, Left Menu, Right bar, and Footer sections. It has a placeholder for the center section that changes dynamically, as shown below.



[Chart, treemap chart

Description automatically generated](https://www.tutorialsteacher.com/Content/images/mvc/layout-view-2.png)Layout View

The layout view has the same extension as other views, .cshtml or .vbhtml. Layout views are shared with multiple views, so it must be stored in the Shared folder. By default, a layout view \_Layout.cshtml is created when you [Create MVC application](https://www.tutorialsteacher.com/mvc/create-first-asp.net-mvc-application) using Visual Studio, as shown below.

[](https://www.tutorialsteacher.com/Content/images/mvc/layout-view-3.png)Layout Views in Shared Folder

The following is the default \_Layout.cshtml.

\_Layout.cshtml:

 Copy

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>@ViewBag.Title - My ASP.NET Application</title>

@Styles.Render("~/Content/css")

@Scripts.Render("~/bundles/modernizr")

</head>

<body>

<div class="navbar navbar-inverse navbar-fixed-top">

<div class="container">

<div class="navbar-header">

<button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

@Html.ActionLink("Application name", "Index", "Home", new { area = "" }, new { @class = "navbar-brand" })

</div>

<div class="navbar-collapse collapse">

<ul class="nav navbar-nav">

<li>@Html.ActionLink("Home", "Index", "Home")</li>

<li>@Html.ActionLink("About", "About", "Home")</li>

<li>@Html.ActionLink("Contact", "Contact", "Home")</li>

</ul>

</div>

</div>

</div>

<div class="container body-content">

@RenderBody()

<hr />

<footer>

<p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>

</footer>

</div>

@Scripts.Render("~/bundles/jquery")

@Scripts.Render("~/bundles/bootstrap")

@RenderSection("scripts", required: false)

</body>

</html>

As you can see, the layout view contains HTML Doctype, head, and body tags. The only difference is a call to RenderBody() and RenderSection() methods. The child views will be displayed where the RenderBody() is called.

ADVERTISEMENT

Using Layout View

The views which will be displayed in a placeholder RenderBody() are called child views. There are multiple ways to specify which layout view will be used with which child views. You can specify it in a common \_ViewStart.cshtml, in a child view, or in an action method.



ViewStart

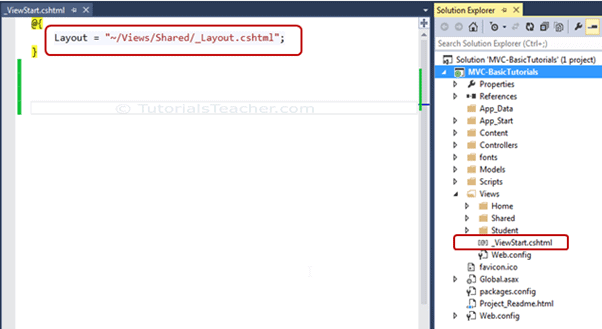
The default \_ViewStart.cshtml is included in the Views folder. It can also be created in all other Views sub-folders. It is used to specify common settings for all the views under a folder and sub-folders where it is created.



Set the Layout property to a particular layout view will be applicable to all the child views under that folder and its sub-folders.



For example, the following \_ViewStart.cshtml in the **Views** folder sets the Layout property to "~/Views/Shared/\_Layout.cshtml". So, the \_layout.cshtml would be a layout view of all the views included in Views and its subfolders.

[](https://www.tutorialsteacher.com/Content/images/mvc/viewstart.png)Setting Layout View in \_ViewStart.cshtml

The \_ViewStart.cshtml can also be created in the sub-folders of the View folder to set the default layout page for all the views included in that particular subfolder.

For example, the following \_ViewStart.cshtml in the Home folder sets the Layout property to \_myLayoutPage.cshtml. So now, Index.cshtml, About.cshtml and Contact.cshtml will display in the \_myLayoutPage.cshtml instead of default \_Layout.cshml.

[Graphical user interface, text, application

Description automatically generated](https://www.tutorialsteacher.com/Content/images/mvc/viewstart-home.png)Layout View in Sub-folders

Specify Layout View in a Child View

You can also override the default layout view setting of \_ViewStart.cshtml by setting the Layout property in each child view. For example, the following Index.cshtml view uses the \_myLayoutPage.cshtml even if \_ViewStart.cshtml sets the \_Layout.cshtml.

Index.cshtml

 Copy

@{

ViewBag.Title = "Home Page";

Layout = "~/Views/Shared/\_myLayoutPage.cshtml";

}

<div class="jumbotron">

<h1>ASP.NET</h1>

<p class="lead">ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.</p>

<p><a href="http://asp.net" class="btn btn-primary btn-lg">Learn more &raquo;</a></p>

</div>

<div class="row">

<div class="col-md-4">

<h2>Getting started</h2>

<p>

ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that

enables a clean separation of concerns and gives you full control over markup

for enjoyable, agile development.

</p>

<p><a class="btn btn-default" href="http://go.microsoft.com/fwlink/?LinkId=301865">Learn more &raquo;</a></p>

</div>

<div class="col-md-4">

<h2>Get more libraries</h2>

<p>NuGet is a free Visual Studio extension that makes it easy to add, remove, and update libraries and tools in Visual Studio projects.</p>

<p><a class="btn btn-default" href="http://go.microsoft.com/fwlink/?LinkId=301866">Learn more &raquo;</a></p>

</div>

<div class="col-md-4">

<h2>Web Hosting</h2>

<p>You can easily find a web hosting company that offers the right mix of features and price for your applications.</p>

<p><a class="btn btn-default" href="http://go.microsoft.com/fwlink/?LinkId=301867">Learn more &raquo;</a></p>

</div>

</div>

Specify Layout Page in Action Method

Specify the layout view name as a second parameter in the View() method, as shown below. By default, layout view will be searched in the Shared folder.

Example: Specify Layout View in Action Method

 Copy

public class HomeController : Controller

{

public ActionResult Index()

{

return View("Index", "\_myLayoutPage"); //set "\_myLayoutView" as layout view

}

}

Rendering Methods

ASP.NET MVC layout view renders child views using the following methods.

| Method | Description |
| --- | --- |
| RenderBody() | Renders the portion of the child view that is not within a named section. Layout view must  include the RenderBody() method. |
| RenderSection(string name) | Renders a content of named section and specifies whether the section is required. |

The following figure illustrates the use of the RenderBody() and RenderSection() methods.

[Diagram, treemap chart

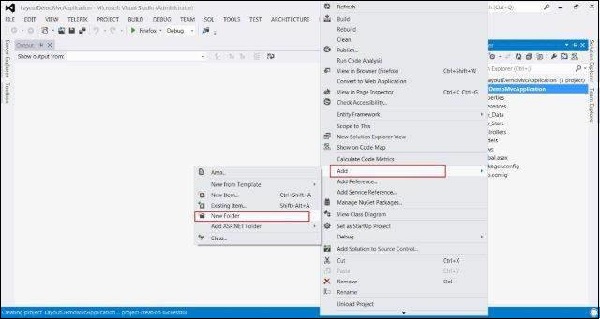
Description automatically generated](https://www.tutorialsteacher.com/Content/images/mvc/layout-view-rendering.png)Rendering Methods

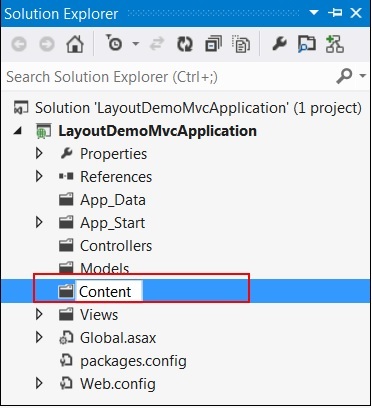
As you can see in the above figure, the \_Layout.cshtml includes the RenderBody() method and RenderSection() method. Above, Index.cshtml contains the named sections using @section where the name of each section matches the name specified in the RenderSection() method of a layout view \_Layout.cshtml, e.g. @Section RightSection. At run time, the named sections of Index.cshtml, such as LeftSection, RightSection, and MiddleSection will be rendered at appropriate place where the RenderSection() method is called. The rest of the Index.cshtml view, which is not in any of the named section, will be rendered in the RenderBody() is called.

Layouts are used in MVC to provide a consistent look and feel on all the pages of our application. It is the same as defining the Master Pages but MVC provides some more functionalities.

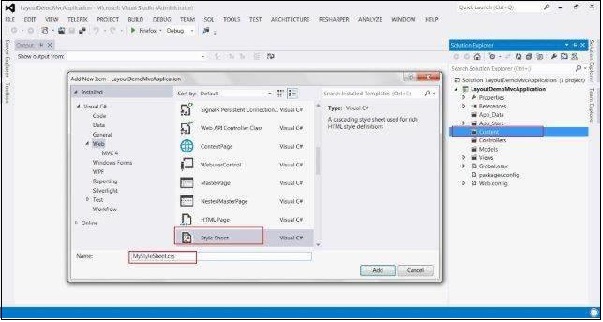
## Create MVC Layouts

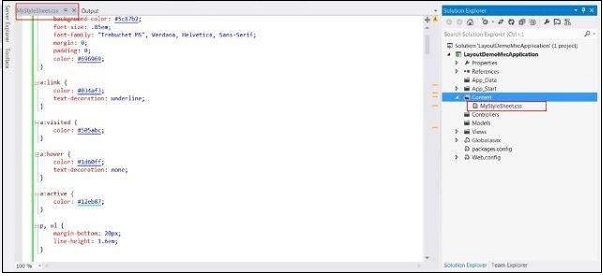
**Step 1** − Create a sample MVC application with Internet application as Template and create a Content folder in the root directory of the web application.



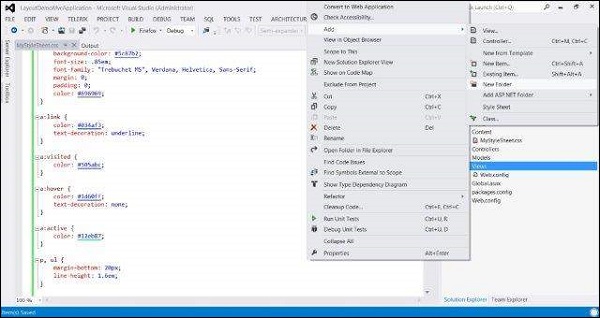


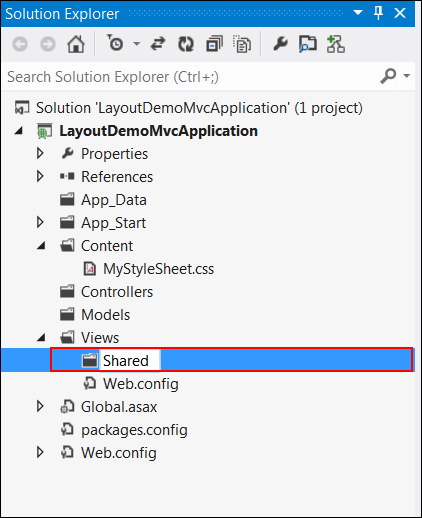
**Step 2** − Create a Style Sheet file named MyStyleSheet.css under the CONTENT folder. This CSS file will contain all the CSS classes necessary for a consistent web application page design.



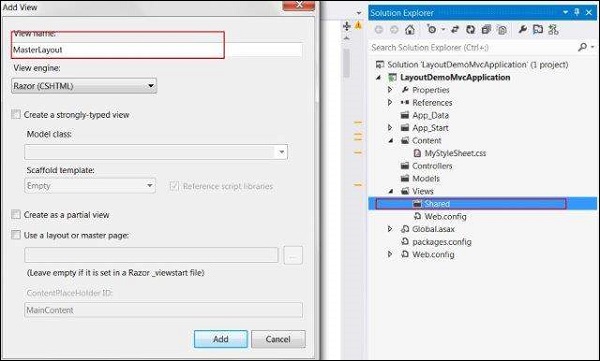


**Step 3** − Create a Shared folder under the View folder.





**Step 4** − Create a MasterLayout.cshtml file under the Shared folder. The file MasterLayout.cshtml represents the layout of each page in the application. Right-click on the Shared folder in the Solution Explorer, then go to Add item and click View. Copy the following layout code.



### Layout Code

<!DOCTYPE html>

<html lang = "en">

<head>

<meta charset = "utf-8" />

<title>@ViewBag.Title - Tutorial Point</title>

<link href = "~/favicon.ico" rel = "shortcut icon" type = "image/x-icon" />

<link rel = "stylesheet" href = "@Url.Content("~/Content/MyStyleSheet.css")" />

</head>

<body>

<header>

<div class = "content-wrapper">

<div class = "float-left">

<p class = "site-title">

@Html.ActionLink("Tutorial Point", "Index", "Home")

</p>

</div>

<div class = "float-right">

<nav>

<ul id = "menu">

<li>@Html.ActionLink("Home", "Index", "Home")</li>

<li>@Html.ActionLink("About", "About", "Home")</li>

</ul>

</nav>

</div>

</div>

</header>

<div id = "body">

@RenderSection("featured", required: false)

<section class = "content-wrapper main-content clear-fix">

@RenderBody()

</section>

</div>

<footer>

<div class = "content-wrapper">

<div class = "float-left">

<p>© @DateTime.Now.Year - Tutorial Point</p>

</div>

</div>

</footer>

</body>

</html>

In this layout, we are using an HTML helper method and some other system-defined methods, hence let's look at these methods one by one.

* **Url.Content()** − This method specifies the path of any file that we are using in our View code. It takes the virtual path as input and returns the absolute path.
* **Html.ActionLink()** − This method renders HTML links that links to action of some controller. The first parameter specifies the display name, the second parameter specifies the Action name, and the third parameter specifies the Controller name.
* **RenderSection()** − Specifies the name of the section that we want to display at that location in the template.
* **RenderBody()** − Renders the actual body of the associated View.

**Step 5** − Finally, open the \_ViewStart.cshtml file inside Views folder and add the following code −

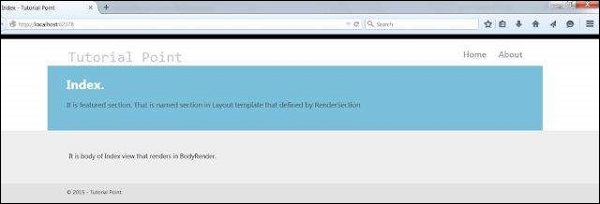
@{

Layout = "~/Views/Shared/\_Layout.cshtml";

}

If the file is not present, you can create the file with this name.

**Step 6** − Run the application now to see the modified home page.



Suppose you are developing an ASP.NET web application and you want to maintain a consistent look and feel across all the pages within you web application. You then have two options, the first one is to design the head, body and footer sections on each page. In this approach you need to write more code on each page so ASP.NET 2.0 introduced "Master Pages" that helps enable this when using .aspx based pages or templates. It is your second option. Razor also supports this concept with a feature called "layouts" that allow you to define a common site template, and then inherit its look and feel across all the views/pages on your web application.

Create an MVC Application

I will create a MVC application using Visual Studio 2012. So let's see step-by-step how to create a MVC application.  
  
**Step 1:** Go to "File" -> "New" -> "Project...".  
  
**Step 2:** Choose "ASP.NET MVC 4 Web Application" from the list then give the application name "LayoutMvcApplication" and set the path in the location input where you want to create the application.  
  
**Step 3:**Now choose the Project Template "Empty" and select "Razor" as the view engine from the dropdown list.

Create Layout for Application

A layout is used to provide a consistent look and feel of all pages of a web application so we create a layout for the web application. Let's see the procedure for that.  
  
**Step 1:** Create a "Content" folder in the root directory of the web application.  
  
**Step 2:**Create a Style Sheet "Site.css" under the content folder. That CSS file contains all CSS classes necessary for a consistent web application page design.  
  
**Step 3:**Create a "Shared" folder under the "View" folder.  
  
**Step 4:** Create a "\_Layout.cshtml" file under the "Shared" folder. The file "\_Layout.cshtml" represents the layout of each page in the application. Right-click on the Shared folder in Solution Explorer then go to "Add" item and click on "View".  
  
Graphical user interface, application

Description automatically generated  
  
Now the View has been created. Write the following code in the "\_Layout.cshtnl" file.

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="utf-8" />

    <title>@ViewBag.Title - Code Express</title>

    <link href="~/favicon.ico" rel="shortcut icon" type="image/x-icon" />

    <link rel="stylesheet" href="@Url.Content("~/Content/Site.css")" />

</head>

<body>

    <header>

            <div **class**="content-wrapper">

                <div **class**="float-left">

                    <p **class**="site-title">

                        @Html.ActionLink("Code Express", "Index", "Home")

                    </p>

                </div>

                <div **class**="float-right">

                    <nav>

                        <ul id="menu">

                            <li>@Html.ActionLink("Home", "Index", "Home")</li>

                            <li>@Html.ActionLink("About", "About", "Home")</li>

                        </ul>

                    </nav>

                </div>

            </div>

        </header>

    <div id="body">

        @RenderSection("featured", required: **false**)

        <section **class**="content-wrapper main-content clear-fix">

                @RenderBody()

            </section>

    </div>

    <footer>

            <div **class**="content-wrapper">

                <div **class**="float-left">

                    <p>© @DateTime.Now.Year – Code Express</p>

                </div>

            </div>

        </footer>

</body>

</html>

In this layout we are using a HTML helper method and some other system defined methods so let's see these methods one by one.

**Url.Content():** Content() method is a method of UrlHelper class. It converts a virtual (relative) path to an application absolute path. It has one parameter of string type that is a virtual path of the content. It returns an application's absolute path. If the specified content path (parameter of the method) does not start with the tilde (~) character then this method returns contentPath unchanged. Url.Content() ensures that all links work no matter if the site is in a virtual directory or in the web site root.  
  
**Html.ActionLink():** The easiest way to render an HTML link in is to use the HTML.ActionLink() helper. With MVC, the Html.ActionLink() does not link to a view. It creates a link to a controller action. ActionLink() is an extension method of the HtmlHelper class. It returns an anchor element (an element) that contains the virtual path of the specified action. When you use an ActionLink() method then you need to pass three string parameter. The parameters are linkText (the inner text of the anchor element), actionName (the name of the action) and controllerName (the name of the controller).  
  
**RenderSection():** RenderSection() is a method of the WebPageBase class. Scott wrote at one point, The first parameter to the "RenderSection()" helper method specifies the name of the section we want to render at that location in the layout template. The second parameter is optional and allows us to define whether the section we are rendering is required or not. If a section is "required", then Razor will throw an error at runtime if that section is not implemented within a view template that is based on the layout file (that can make it easier to track down content errors). It returns the HTML content to render.  
  
**RenderBody():** In layout pages, renders the portion of a content page that is not within a named section. It returns the HTML content to render. RenderBody is required since it's what renders each view.  
  
**The \_ViewStart File**  
The "\_ViewStart" file in the Views folder contains the following content:

1. @{
2. Layout = "~/Views/Shared/\_Layout.cshtml";
3. }

This code is automatically added to all views displayed by the application. If you remove this file then you must add this line to all views.

Create Controller in MVC Application

You need to create a controller that contains an action method to render the view on the user interface. I will create a HomeController controller with two action methods, one is Index and the other is About. Both action methods execute when the request type is GET and renders a view on the browser. So use the following procedure.  
  
**Step 1:** Right-click on the Controllers folder under Solution Explorer then go to "Add" and click on "Controller".  
  
**Step 2:** Give the name HomeController for the Controller name input and select "Empty MVC controller" from the Template dropdown list. After that click on "Add".  
  
Now you need to write the following code in the "HomeController.cs" file:

1. **using** System.Web.Mvc;
2. **namespace** LayoutMvcApplication.Controllers
3. {
4. **public** **class** HomeController : Controller
5. {
6. **public** ActionResult Index()
7. {
8. **return** View();
9. }
10. **public** ActionResult Index()
11. {
12. **return** View();
13. }
14. }
15. }

Create View in MVC Application

You need to create two views (Index and About) that inherit the "\_Layout" view via the "\_ViewStart" view so you need to use the following procedure.  
  
**Step 1:** Right-click on the Index() action method under the "HomeController.cs" file then click on "Add View".  
  
**Step 2:** Leave Add New screen unchanged and click on "Add"  
  
Now you should write the following code in your web application index view:

1. @{
2. ViewBag.Title = "Index";
3. }
4. @section featured
5. {
6. <section **class**="featured">
7. <div **class**="content-wrapper">
8. <hgroup **class**="title">
9. <h1>@ViewBag.Title.</h1>
10. </hgroup>
11. <p>
12. It is featured section. That is named section **in** Layout template that defined by RenderSection
13. </p>
14. </div>
15. </section>
16. }
17. <p> It is body of Index view that renders **in** BodyRender.</p>

You can follow the same steps for the About view and create it.

Graphical user interface, text, application, email

Description automatically generated  
  
You can get the same design for the About view. You can download the source code from the attachment.  
  
With this code I want to share one more thing, that it is my maiden fifty of articles on C# Corner.