**Partial Views in ASP.NET MVC**

**Partial Views in ASP.NET MVC Application**

In this article, I am going to discuss **Partial Views in ASP.NET MVC** Application. Please read our previous section articles where we discussed [**Action Results in ASP.NET MVC**](https://dotnettutorials.net/lesson/action-result-overview-mvc/) Application. At the end of this article, you will understand what are Partial Views in MVC and Why do we need partial Views, and how to implement Partial Views in the ASP.NET MVC Application with Examples.

**Why do we need Partial Views in the ASP.NET MVC Application? or When Should I Use Partial Views?**

When we need a common part of the user interface at multiple pages in a web application then we develop a partial view. Hence partial view is a regular view that can be used multiple times in an application and has the file extension .cshtml.

Sometimes we also use a partial view to divide a web page into small parts such as header, footer, and menu on Layout. Other examples are comments on blogging site, shipping and billing address in the invoice in e-commerce site, etc.

If you are coming from an ASP.NET WebForms background, then you can compare the Partial views in the ASP.NET MVC Application with user controls in the ASP.NET WebForms application. That means a Partial View is like user control in Asp.Net Webforms that are used for code re-usability. Partial views help us to reduce code duplication. Hence partial views are reusable views like Header and Footer views.

**What are Partial Views in MVC Application?**

The Partial Views in ASP.NET MVC Application are the views that are rendered within another view. The HTML output generated by partial view is rendered into the calling (or parent) view. Like views, partial views use the *.*cshtml file extension.

**How can we Call/Display Partial View?**

We can call or display partial view within a view mainly in five ways. They are as follows:

1. **Html.RenderPartial**
2. **Html.Partial**
3. **Html.RenderAction**
4. **Html.Action**
5. **jQuery load function**

**Example: Partial Views in ASP.NET MVC Application.**

Let us understand Partial Views in ASP.NET MVC Application with an example. It is a common task in Web Applications to make use of the same code over and over again to display/render information and details for their domain objects. For example, an e-shop Web Application would probably render each product in the same way on all web pages. Consider the following product class.

**public** **class** Product

**{**

**public** **long** ProductID **{** **get**; **set**; **}**

**public** string Name **{** **get**; **set**; **}**

**public** string Category **{** **get**; **set**; **}**

**public** string Description **{** **get**; **set**; **}**

**public** **decimal** Price **{** **get**; **set**; **}**

**}**

The View for rendering the list of products in ASP.NET MVC View would be something like below

@foreach (var product in Model)

{

**<div>**

ID: **<span>**@product.ProductID**</span>**

Name: **<span>**@product.Name**</span>**

Category: **<span>**@product.Category**</span>**

Description: **<span>**@product.Description**</span>**

Price: **<span>**@product.Price**</span>**

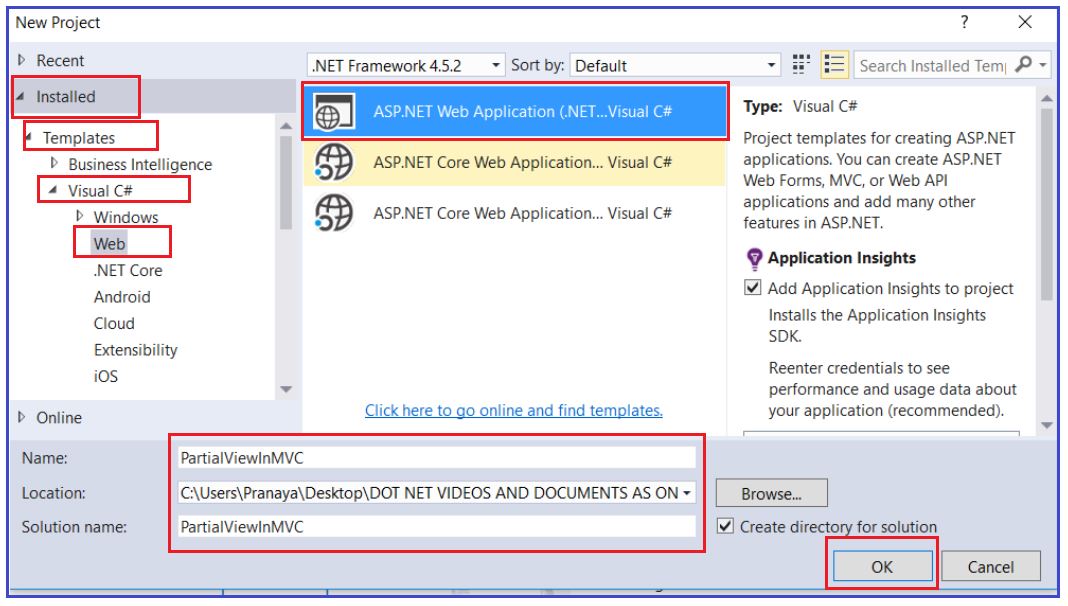
**</div>**

}

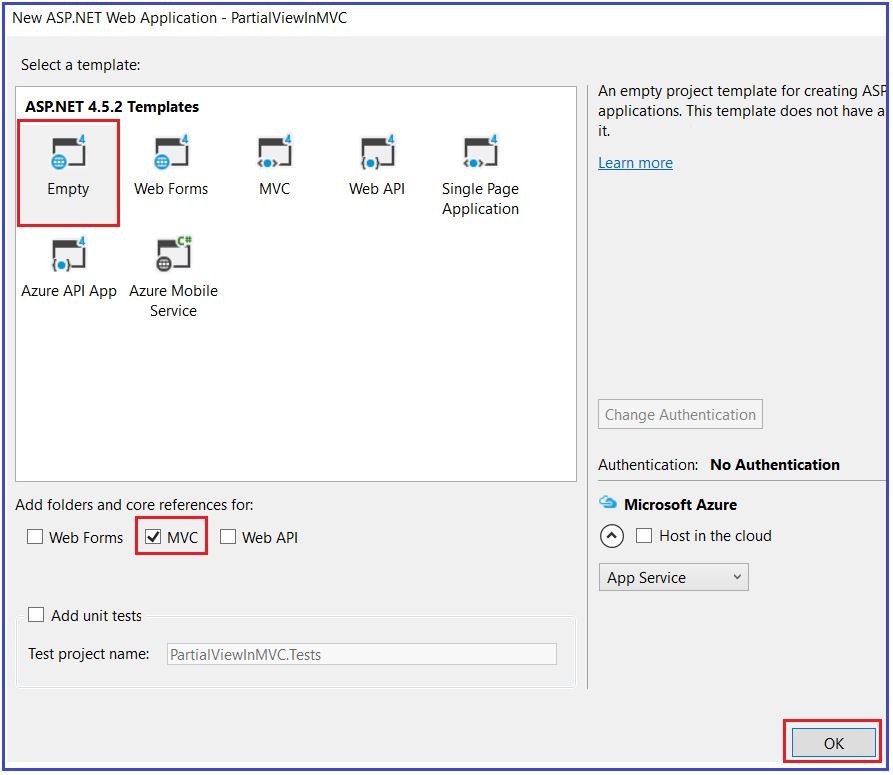
This is just a simple Product class for demonstration purposes. What if we wanted to display objects with twenty or even more properties? And what if we needed to display this information on many pages in our application? Writing the same code, again and again, would be time-consuming, error-prone, and maintenance becomes a headache as if we want to do any modification, then we need to do it at all the places. This is where Partial Views Comes into the picture in ASP.NET MVC Application.

**Create an ASP.NET MVC Application and understand the power of Partial Views.**

Create a new ASP.NET Web Application named “**PartialViewInMVC**” and click on the **OK** button as shown in the below image.



Once you click on the OK, it will open the “New ASP.NET Web Application” window to select the Project Template. In this window choose the **Empty** template and check the **MVC** checkbox, then click on the **OK** button as shown in the below image.



Once you click on the OK button, it will create an empty ASP.NET MVC 5 Project.

**Creating Model:**

Add a new class file with the name Product.cs within the Models folder and then copy and paste the following code into it.

**namespace** *PartialViewInMVC.Models*

**{**

**public** **class** Product

**{**

**public** **long** ProductID **{** **get**; **set**; **}**

**public** string Name **{** **get**; **set**; **}**

**public** string Category **{** **get**; **set**; **}**

**public** string Description **{** **get**; **set**; **}**

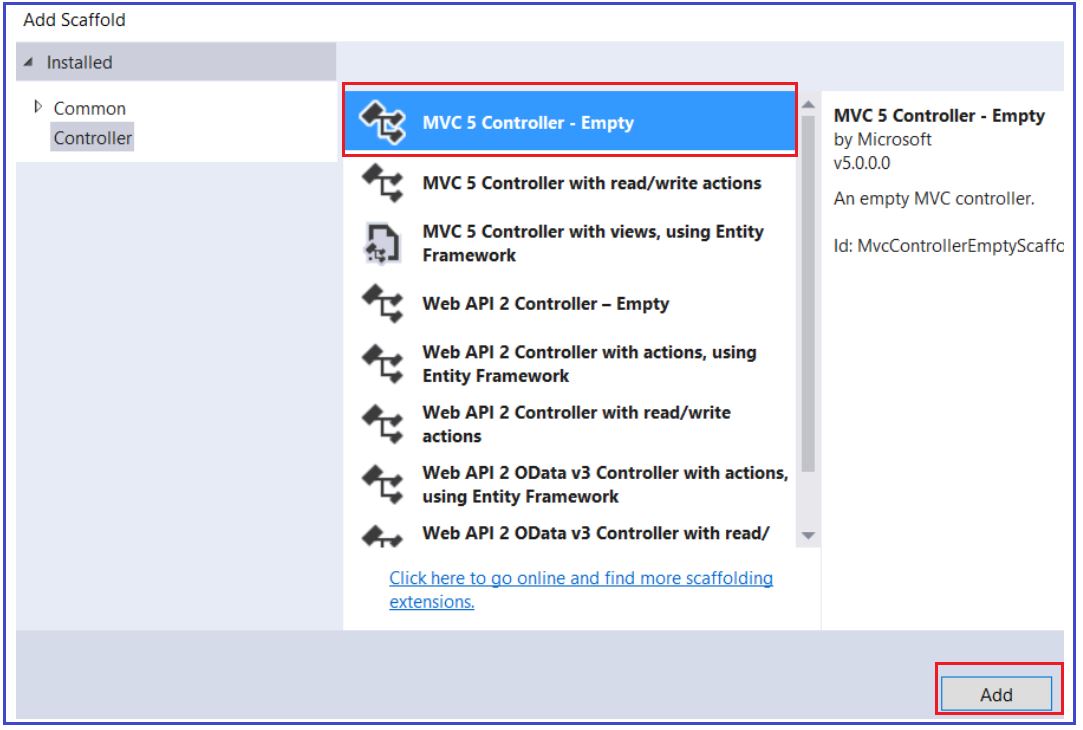
**public** **decimal** Price **{** **get**; **set**; **}**

**}**

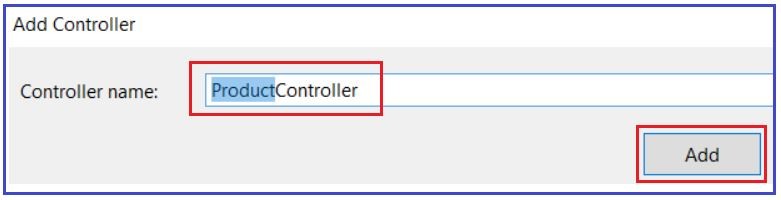
**}**

**Creating Controller:**

Add a new Controller named “**ProductController**” in the Controllers folder, choosing the **Empty MVC5 Controller** template, and click on **ADD** button as shown in the below image.



In the next screen provide the controller name as ProductController and click on Add button as shown in the below image.



Once the ProductController is created, then copy and paste the following code into it.

**public** **class** ProductController : Controller

**{**

**public** ActionResult Index**()**

**{**

List**<**Product**>** products = new List**<**Product**>()**

**{**

new Product **{** ProductID =1, Name ="Product 1", Category = "Category 1", Description ="Description 1", Price = 10m**}**,

new Product **{** ProductID =2, Name ="Product 2", Category = "Category 1", Description ="Description 2", Price = 20m**}**,

new Product **{** ProductID =3, Name ="Product 3", Category = "Category 1", Description ="Description 3", Price = 30m**}**,

new Product **{** ProductID =4, Name ="Product 4", Category = "Category 2", Description ="Description 4", Price = 40m**}**,

new Product **{** ProductID =5, Name ="Product 5", Category = "Category 2", Description ="Description 5", Price = 50m**}**,

new Product **{** ProductID =6, Name ="Product 6", Category = "Category 2", Description ="Description 6", Price = 50m**}**

**}**;

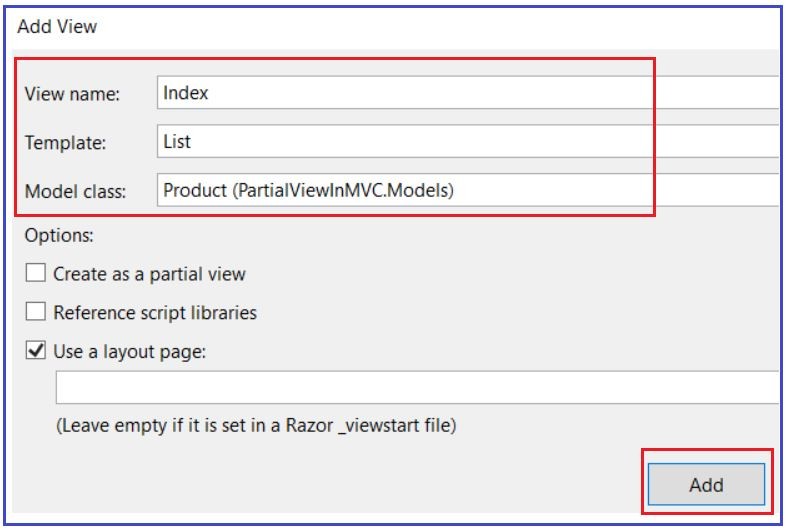
**return** View**(**products**)**;

**}**

**}**

**Creating View:**

Right-click inside the Index action method and then select the “**Add view**” option from the context menu and provide the following details and click on Add button as shown in the below image.



Once the Index View is created, then copy and paste the following code into it.

@model IEnumerable**<PartialViewInMVC.Models.Product>**

@{

ViewBag.Title = "Index";

}

**<h2>**Product List**</h2>**

**<table** class="table"**>**

**<tr>**

**<th>**

@Html.DisplayNameFor(model => model.ProductID)

**</th>**

**<th>**

@Html.DisplayNameFor(model => model.Name)

**</th>**

**<th>**

@Html.DisplayNameFor(model => model.Category)

**</th>**

**<th>**

@Html.DisplayNameFor(model => model.Description)

**</th>**

**<th>**

@Html.DisplayNameFor(model => model.Price)

**</th>**

**</tr>**

@foreach (var item in Model) {

**<tr>**

**<td>**

@Html.DisplayFor(modelItem => item.ProductID)

**</td>**

**<td>**

@Html.DisplayFor(modelItem => item.Name)

**</td>**

**<td>**

@Html.DisplayFor(modelItem => item.Category)

**</td>**

**<td>**

@Html.DisplayFor(modelItem => item.Description)

**</td>**

**<td>**

@Html.DisplayFor(modelItem => item.Price)

**</td>**

**</tr>**

}

**</table>**

Change the default controller in the RouteConfig.cs file, from “Home” to “Product” as shown below.

**public** **class** RouteConfig

**{**

**public** **static** **void** RegisterRoutes**(**RouteCollection routes**)**

**{**

routes.IgnoreRoute**(**"{resource}.axd/{\*pathInfo}"**)**;

routes.MapRoute**(**

name: "Default",

url: "{controller}/{action}/{id}",

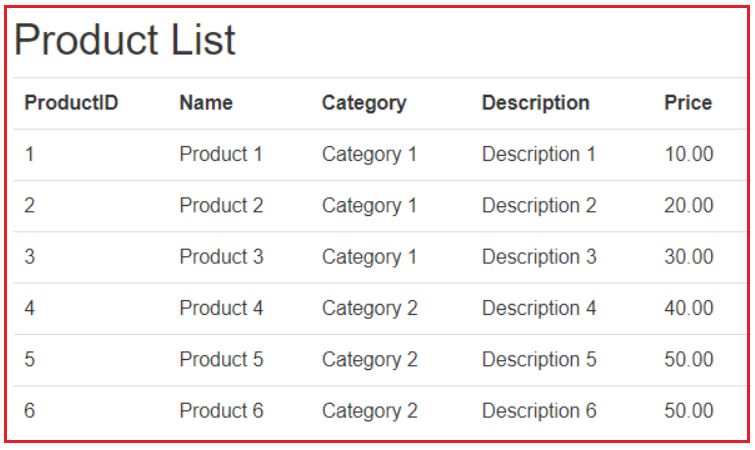
defaults: new **{** controller = "Product", action = "Index", id = UrlParameter.Optional **}**

**)**;

**}**

**}**

Now, build and run your application and navigates to Product/Index and you should get the following output.

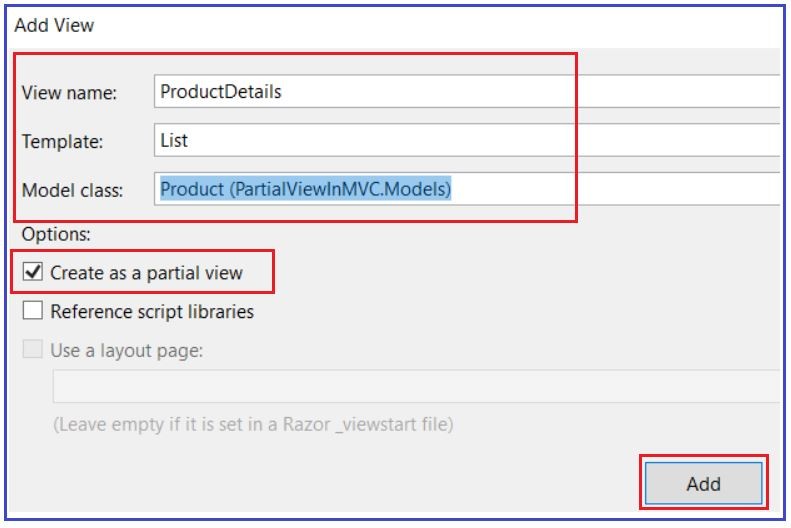


When we need a section of a web page (both the Razor tags and HTML markup) in several different places, we create and use them as Partial Views.

**How Partial Views are Created in ASP.NET MVC Application?**

Right-click on the **/Views/Shared** folder and Select **Add -> View** option from the context menu and then provide the following details

1. View Name = ProductDetails
2. Template = List
3. Model Class = Product (PartialViewInMVC.Models)
4. Check the Create a partial View check box and click on Add button as shown in the below image



Once the ProductDetails Partial View is created, then copy and paste the following code into it.

@model IEnumerable**<PartialViewInMVC.Models.Product>**

**<div** class="container"**>**

**<div** class="col-md-6"**>**

**<h2>**Product List**</h2>**

**<table** class="table"**>**

**<tr>**

**<th>**

@Html.DisplayNameFor(model => model.ProductID)

**</th>**

**<th>**

@Html.DisplayNameFor(model => model.Name)

**</th>**

**<th>**

@Html.DisplayNameFor(model => model.Category)

**</th>**

**<th>**

@Html.DisplayNameFor(model => model.Description)

**</th>**

**<th>**

@Html.DisplayNameFor(model => model.Price)

**</th>**

**</tr>**

@foreach (var item in Model)

{

**<tr>**

**<td>**

@Html.DisplayFor(modelItem => item.ProductID)

**</td>**

**<td>**

@Html.DisplayFor(modelItem => item.Name)

**</td>**

**<td>**

@Html.DisplayFor(modelItem => item.Category)

**</td>**

**<td>**

@Html.DisplayFor(modelItem => item.Description)

**</td>**

**<td>**

@Html.DisplayFor(modelItem => item.Price)

**</td>**

**</tr>**

}

**</table>**

**</div>**

**</div>**

**How to use the Partial Views in ASP.NET MVC Application?**

To use this Partial view, remove the respective code in the Index View and replace it with**Html.Partial** helper method as shown in the below code.

@model IEnumerable**<**PartialViewInMVC.Models.Product**>**

@**{**

ViewBag.Title = "Index";

**}**

@Html.Partial**(**"ProductDetails", Model**)**

Now, build and run your application and see that everything is working as expected. But this time you can re-use this partial view wherever you want and moreover if you decide to change how product objects are rendered, the only View you need to change is the ProductDetails partial view. The above @Html.Partial helper method passed a List<Product> object in the “ProductDetails” partial view. The partial view was dynamically rendered.

# Different Ways to Render Partial View in ASP.NET MVC

## ****Different Ways to Render Partial View in ASP.NET MVC****

In this article, I will explain **Different Ways to Render a Partial view in the ASP.NET MVC** application. Please read **our previous article** before proceeding to this article as we are going to use the same example. In our previous article, we discussed [**What is Partial View?** **Why do we need a Partial View? How we can use Partial View in the ASP.NET MVC Application?**](https://dotnettutorials.net/lesson/partial-views-in-mvc/) At the end of this article, you will understand the different methods to render a partial view in the ASP.NET MVC application.

##### ****Different Ways to Render Partial Views in ASP.NET MVC Application****

We can render Partial Views in our main views in 5 ways. They are as follows:

1. **Html.RenderPartial**
2. **Html.Partial**
3. **Html.RenderAction**
4. **Html.Action**
5. **jQuery Load function**

##### ****Rendering Partial Views using Html.RenderPartial Helper Method in ASP.NET MVC Application****

There are 4 overloaded versions available for the RenderPartial method in ASP.NET MVC Framework as shown in the below image.



###### **Parameters:**

1. **htmlHelper**: The HTML helper instance that this method extends
2. **partialViewName**: The name of the partial view.
3. **viewData**: The view data for the partial view.
4. **model**: The model for the partial view.

##### ****Example: How to use RenderPartial Helper method to call Partial Views in MVC****

Let us see how to call a Partial view from the main view using the **Html.RenderPartial**helper method in ASP.NET MVC Application. Modify the Index action method of Product Controller to call Partial view using Html.RenderPartial() method as shown below. Here, we are using the overloaded version of the RenderPartial method which takes the view name and object model as input.

@model IEnumerable**<PartialViewInMVC.Models.Product>**

@{

ViewBag.Title = "Index";

}

@{Html.RenderPartial("ProductDetails", Model);}

It works when you have the partial view located in the Shared folder. If your partial view is located in a different folder then you will have to mention the full path of view as shown below.

**@{Html.RenderPartial(“~/Views/Home/ProductDetails.cshtml”, Model);}**

##### ****Points to Remember while working with Html.RenderPartial:****

1. RenderPartial() is a void method that writes the output to the response stream.  The “void” method in C# needs a”;” and hence must be enclosed by { }.
2. This method result will be directly written to the HTTP response stream. That means this method generates the response as part of the same HTTP response of the main view. It uses the same TextWriter object used by the current web page.
3. This method returns void.
4. Simple to use and no need to create any action.
5. This method is faster as its result is directly written to the response stream which makes it fast.
6. If you have a model associated with the View and the model required for the partial view is part of ViewModel then the RenderPartial method is ideal to use.

##### ****Html.Partial Helper Method in ASP.NET MVC Application****

There are 4 overloaded versions available for the Partial method in ASP.NET MVC Framework as shown in the below image.



##### ****Parameters:****

1. **htmlHelper**: The HTML helper instance that this method extends
2. **partialViewName**: The name of the partial view to render
3. **viewData**: The view data dictionary for the partial view.
4. **model**: The model for the partial view.

**Returns:** The partial view that is rendered as an HTML-encoded string.

##### ****Example: How to use RenderPartial Helper method to call Partial Views in MVC****

Let us see how to call a Partial view from the main view using the Html.Partial helper method in ASP.NET MVC Application. Modify the Index action method of Product Controller to use Partial view using **Html.Partial()** method as shown below. Here, we are using the overloaded version of the Partial method which takes the view name and object model as input.

@model IEnumerable**<PartialViewInMVC.Models.Product>**

@{

ViewBag.Title = "Index";

}

@Html.Partial("ProductDetails", Model)

It works when you have a partial view located in the Shared folder. If your partial view is located in a different folder then you will have to mention the full path of the view as shown below.

**@Html.Partial(“~/Views/Shared/ProductDetails.cshtml”, Model)**

##### ****Points to Remember while working with Html.Partial in MVC:****

1. The Partial() Helper method in MVC is a method that returns a MvcHtmlString. In Razor, You can call a property or a method that returns such a string with just a @ prefix to distinguish it from plain HTML.
2. Renders the partial view as an HTML-encoded string.
3. This method result can be stored in a variable since it returns string type value.
4. Simple to use and no need to create any action.
5. Like the RenderPartial method, the Partial method is also useful when displaying data in the partial view is already in the corresponding view model.

##### ****Html.RenderAction in ASP.NET MVC Application****

For rendering Partial view using Html.RenderAction help method, we required the Controller Action method which returns PartialViewResult. Add the following GetProducts method in the Product Controller class. Notice, this action method is going to return a Partial View.

**public** PartialViewResult GetProducts**()**

**{**

List**<**Product**>** products = new List**<**Product**>()**

**{**

new Product **{** ProductID =1, Name ="Product 1", Category = "Category 1", Description ="Description 1", Price = 10m**}**,

new Product **{** ProductID =2, Name ="Product 2", Category = "Category 1", Description ="Description 2", Price = 20m**}**,

new Product **{** ProductID =3, Name ="Product 3", Category = "Category 1", Description ="Description 3", Price = 30m**}**,

new Product **{** ProductID =4, Name ="Product 4", Category = "Category 2", Description ="Description 4", Price = 40m**}**,

new Product **{** ProductID =5, Name ="Product 5", Category = "Category 2", Description ="Description 5", Price = 50m**}**,

new Product **{** ProductID =6, Name ="Product 6", Category = "Category 2", Description ="Description 6", Price = 50m**}**

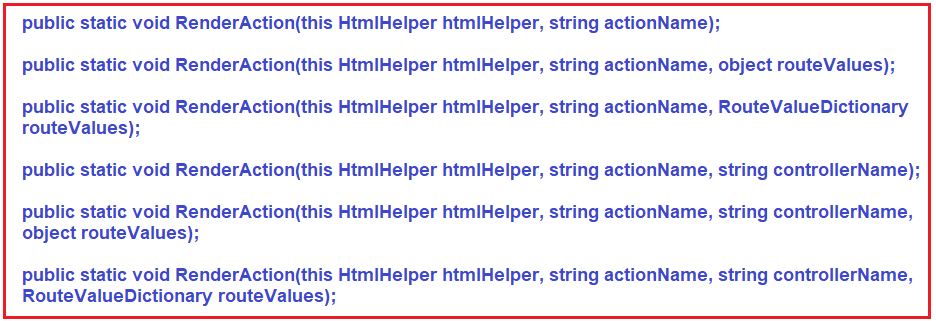
**}**;

**return** PartialView**(**"ProductDetails", products**)**;

**}**

##### ****RenderAction Helper Method in ASP.NET MVC****

There are 6 overloaded versions available for the RenderAction HTML Helper method in ASP.NET MVC Framework as shown in the below image.



##### ****Parameters:****

1. **htmlHelper**: The HTML helper instance that this method extends.
2. **actionName**: The name of the child action method to invoke.
3. **controllerName**: The name of the controller that contains the action method.
4. **object routeValues**: An object that contains the parameters for a route. You can use routeValues to provide the parameters that are bound to the action method parameters. The routeValues parameter is merged with the original route values and overrides them.
5. **RouteValueDictionary routeValues**: A dictionary that contains the parameters for a route. You can use routeValues to provide the parameters that are bound to the action method parameters. The routeValues parameter is merged with the original route values and overrides them.

##### ****Example: How to use RenderAction Helper method to call Partial Views in MVC****

Let us see how to call a Partial view from the main view using the Html.RenderAction helper method in ASP.NET MVC Application. Modify the Index action method of Product Controller to use Partial view using **Html.RenderAction()** method as shown below. Here, we are using the overloaded version of the RenderAction method which takes the action name and controller name as input.

@{

ViewBag.Title = "Index";

}

@{Html.RenderAction("GetProducts", "Product");}

Build the solution and run the application and see everything is working as expected.

##### ****Points to Remember while working with Html.RenderAction in MVC:****

1. This method result will be directly written to the HTTP response stream of the parent web page like Html.RenderPartial. That means it uses the same TextWriter object as used in the current webpage/template.
2. For this method, we need to create a child action for rendering the partial view.
3. The RenderAction method is useful when the displaying data in the partial view is independent of the corresponding view model.
4. This method is the best choice when you want to cache a partial view.
5. This method is faster HTML.Action Helper Method as its result is directly written to the HTTP response stream which makes it fast.

##### ****Html.Action Helper Method in ASP.NET MVC Application****

For rendering a partial view using Html.Action, we required the Controller Action method which returns **PartialViewResult**like Html.RenderAction. There are 6 overloaded versions available for the Action HTML Helper method in ASP.NET MVC Framework as shown in the below image.



##### ****Parameters:****

1. **htmlHelper**: The HTML helper instance that this method extends.
2. **actionName**: The name of the action method to invoke.
3. **controllerName**: The name of the controller that contains the action method.
4. **object routeValues**: An object that contains the parameters for a route. You can use routeValues to provide the parameters that are bound to the action method parameters. The routeValues parameter is merged with the original route values and overrides them.
5. **RouteValueDictionary routeValues**: A dictionary that contains the parameters for a route. You can use routeValues to provide the parameters that are bound to the action method parameters. The routeValues parameter is merged with the original route values and overrides them.

**Returns:** The child action result as an HTML string.

##### ****Example: How to use Action Helper method to call Partial Views in MVC****

Let us see how to call a Partial view from the main view using the Html.Action helper method in ASP.NET MVC Application. Modify the Index action method of Product Controller to use Partial view using **Html.Action()** method as shown below. Here, we are using the overloaded version of the Action method which takes the action name and controller name as input.

@**{**

ViewBag.Title = "Index";

**}**

@Html.Action**(**"GetProducts", "Product"**)**

Build the solution and run the application and see everything is working as expected.

###### **Points to Remember while working with Html.Action Method in MVC:**

1. Renders the partial view as an HtmlString.
2. For this method, we need to create a child action for rendering the partial view.
3. This method result can be stored in a variable since it returns string type value.
4. The action method is useful when displaying data in the partial view is independent of the corresponding view model.
5. This method is also the best choice when you want to cache a partial view.

##### ****Render Partial View Using jQuery in ASP.NET MVC****

We can load our partial view using the jQuery load method. It makes ajax requests to controller action method and load output in HTML control like div. Add div in the index.cshtml file as shown below and add a script to load output of action method GetProducts.

@{

ViewBag.Title = "Index";

}

**<div** id="partialviews"**>**

**</div>**

**<script** src="https://code.jquery.com/jquery-1.10.2.js"**></script>**

**<script** type="text/jscript"**>**

$(document).ready(function () {

$("#partialviews").load('/Product/GetProducts');

});

**</script>**

##### ****When would you use Partial() over RenderPartial() and vice versa in ASP.NET MVC?****

The main difference is that **“RenderPartial()”** returns void and the output will be written directly to the output stream, whereas the **“Partial()”** method returns **MvcHtmlString**, which can be assigned to a variable and manipulate if required. So, when there is a need to assign the output to a variable for manipulating it, then use **Partial**(), else use **RenderPartial**().

##### ****Which one is better for performance?****

From a performance perspective, rendering directly to the output stream is better. RenderPartial() does exactly the same thing and is better for performance than Partial().

##### ****When would you use Action() over RenderAction() and vice versa in ASP.NET MVC?****

The main difference is that **“RenderAction()”** returns void and the output will be written directly to the output stream, whereas the **“Action()”** method returns **MvcHtmlString**, which can be assigned to a variable and manipulate if required. So, when there is a need to assign the output to a variable for manipulating it, then use **Action**(), else use **RenderAction**().

# Razer View Syntax in MVC

## ****Razer View Syntax in MVC****

Use @ symbol to switch between C# code and HTML.

@for (int i = 1; i <= 10; i++)

{

**<b>**@i**</b>**

}

**Output:**  
**1 2 3 4 5 6 7 8 9 10**

Use @{ } to define a code block. If we want to define some variables and perform calculations, then use code block. The following code block defines 2 variables and computes the sum of the first 10 even and odd numbers.

@{

int SumOfEvenNumbers = 0;

int SumOfOddNumbers = 0;

for (int i = 1; i <= 10; i++)

{

if (i % 2 == 0)

{

SumOfEvenNumbers = SumOfEvenNumbers + i;

}

else

{

SumOfOddNumbers = SumOfOddNumbers + i;

}

}

}

**<h3>**Sum of Even Numbers = @SumOfEvenNumbers**</h3>**

**<h3>**Sum of Odd Numbers = @SumOfOddNumbers**</h3>**

**Output:**  
**Sum of Even Numbers = 30**

**Sum of Odd Numbers = 25**

**Use <text> element or @: to switch between c# code and literal text**

@for (int i = 1; i <= 10; i++)

{

**<b>**@i**</b>**

if (i % 2 == 0)

{

**<text>** - Even **</text>**

}

else

{

**<text>** - Odd **</text>**

}

**<br** **/>**

}

###### **The above program can be re-written using @: as shown below.**

@for (int i = 1; i <= 10; i++)

{

**<b>**@i**</b>**

if (i % 2 == 0)

{

@: - Even

}

else

{

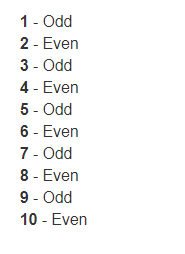
@: - Odd

}

**<br** **/>**

}

**Output:**



###### **Use @\* \*@ to comment in razor views**

**@\*This is a comment in razor views\*@**

**The transition between c# expressions and literal text**

@{

int day = 31;

int month = 12;

int year = 2013;

}

Date is @day-@month-@year

**Output:**  
Date is 31-12-2013

The @ symbol is used as a code delimiter in razor views. However, the razor is smart enough to recognize the format of internet email address and not to treat the @ symbol as a code delimiter.

This is my email address<br />

<b>info@dotnettutorials.net</b>  
Use @ symbol to escape @

I will meet you @@ office

**Output:**  
I will meet you @ office