**Attributes in ASP.NET MVC (Part 1)**

**Attributes in ASP.NET MVC Application**

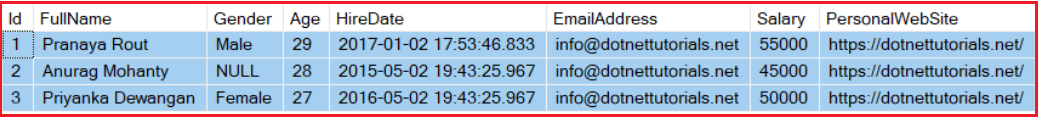
1. **Display**
2. **DisplayName**
3. **DisplayFormat**
4. **ScaffoldColumn**
5. **DataTypeAttribute**
6. **DisplayColumnAttribute**

And in the [**next article**](https://dotnettutorials.net/lesson/attributes-in-asp-dot-net-mvc/)**,** I am going to discuss the following attribute.

1. **UIHint**
2. **HiddenInput**
3. **ReadOnly**

**Creating the Database Table:**

We are going to use the following Employee table.



Please use below SQL Script to create and populate the Employee table that we are going to use in this article.

-- Create Employee Table

**Create** **table** Employee

(

Id int primary key identity,

FullName nvarchar(100),

Gender nvarchar(10),

Age int,

HireDate DateTime,

EmailAddress nvarchar(100),

Salary int,

PersonalWebSite nvarchar(100)

)

**GO**

-- Insert some test data into Employee Table

**Insert** **into** Employee values

('Pranaya Rout', 'Male', 29, '2017-01-02 17:53:46.833', 'info@dotnettutorials.net', 55000, 'https://dotnettutorials.net/')

**Insert** **into** Employee values

('Anurag Mohanty', **NULL**, 28, '2015-05-02 19:43:25.965', 'info@dotnettutorials.net', 45000,'https://dotnettutorials.net/')

**Insert** **into** Employee values

('Priyanka Dewangan', 'Female', 27, '2016-05-02 19:43:25.965', 'info@dotnettutorials.net', 50000,'https://dotnettutorials.net/')

**GO**

**Create an empty ASP.NET MVC application:**

Create an empty ASP.NET MVC Project with the name **AttributesInMVC**. Generate the **ADO.NET Entity Data model** for table Employee using the database first approach. Save and build the project. It will create the following Employee Model

**namespace** *AttributesInMVC.Models*

**{**

**using** System;

**using** System.Collections.Generic;

**public** **partial** **class** Employee

**{**

**public** **int** Id **{** **get**; **set**; **}**

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

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

**public** Nullable**<int>** Age **{** **get**; **set**; **}**

**public** Nullable**<**System.DateTime**>** HireDate **{** **get**; **set**; **}**

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

**public** Nullable**<int>** Salary **{** **get**; **set**; **}**

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

**}**

**}**

**Creating Employee Controller:**

Right-click on the “**Controllers**” folder and add a controller with the name “**EmployeeController**” and then copy and paste the following code in it.

**using** *AttributesInMVC.Models;*

**namespace** *AttributesInMVC.Controllers*

**{**

**public** **class** EmployeeController : Controller

**{**

**public** ActionResult Details**(int** id**)**

**{**

EmployeeDBContext dbContext = new EmployeeDBContext**()**;

Employee employee = dbContext.Employees.Single**(**x =**>** x.Id == id**)**;

**return** View**(**employee**)**;

**}**

**}**

**}**

**Creating Details View:**

Add the details view and then copy and paste the below code in it**.**

@model AttributesInMVC.Models.Employee

@{

ViewBag.Title = "Details";

}

**<div>**

**<h4>**Employee Details**</h4>**

**<dl** class="dl-horizontal"**>**

**<dt>**

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

**</dt>**

**<dd>**

@Html.DisplayFor(model => model.FullName)

**</dd>**

**<dt>**

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

**</dt>**

**<dd>**

@Html.DisplayFor(model => model.Gender)

**</dd>**

**<dt>**

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

**</dt>**

**<dd>**

@Html.DisplayFor(model => model.Age)

**</dd>**

**<dt>**

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

**</dt>**

**<dd>**

@Html.DisplayFor(model => model.HireDate)

**</dd>**

**<dt>**

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

**</dt>**

**<dd>**

@Html.DisplayFor(model => model.EmailAddress)

**</dd>**

**<dt>**

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

**</dt>**

**<dd>**

@Html.DisplayFor(model => model.Salary)

**</dd>**

**<dt>**

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

**</dt>**

**<dd>**

@Html.DisplayFor(model => model.PersonalWebSite)

**</dd>**

**</dl>**

**</div>**

Now, run the application and navigate to the URL **http://localhost:61449/Employee/Details/1**and It will display the employee information as shown below.



Notice that the output is not that pretty. We can control the display of data in a view using display attributes that are found in the **System.ComponentModel.DataAnnotations** namespace. It is not a good idea to add display attributes to the properties of the auto-generated “Employee” class as our changes will be lost if the class is auto-generated again.

**Creating Employee Partial Class:**

We need to create a partial “**Employee**” class and we need to decorate that class with the display attributes.  So, Right-click on the “**Models**” folder and then add a class file with the name **ModifyEmployee.cs**. Once you created the **ModifyEmployee.cs** then copy and paste the following code in it. The code is self-explained. So, please go through the comment lines.

**namespace** *AttributesInMVC.Models*

**{**

**[**MetadataType**(**typeof**(**EmployeeMetaData**))]**

**public** **partial** **class** Employee

**{**

**}**

**public** **class** EmployeeMetaData

**{**

//If you want "FullName" to be displayed as "Full Name",

//use DisplayAttribute or DisplayName attribute.

//DisplayName attribute is in System.ComponentModel namespace.

//[DisplayAttribute(Name="Full Name")]

//[Display(Name = "Full Name")]

**[**DisplayName**(**"Full Name"**)]**

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

//To get only the date part in a datetime data type

//[DisplayFormat(DataFormatString = "{0:d}")]

//[DisplayFormatAttribute(DataFormatString="{0:d}")]

//To get time in 24 hour notation

//[DisplayFormat(DataFormatString = "{0:dd/MM/yyyyHH:mm:ss}")]

//To get time in 12 hour notation with AM PM

//[DisplayFormat(DataFormatString = "{0:dd/MM/yyyyhh:mm:sstt}")]

// public DateTime? HireDate { get; set; }

// Display only Time Part

// [DataType(DataType.Time)]

// Display only Date Part

**[**DataType**(**DataType.Date**)]**

**public** DateTime? HireDate **{** **get**; **set**; **}**

// If gender is NULL, "Gender not specified" text will be displayed.

**[**DisplayFormat**(**NullDisplayText = "Gender not specified"**)]**

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

//If you don't want to display a column use ScaffoldColumn attribute.

//This only works when you use @Html.DisplayForModel() helper

//[ScaffoldColumn(false)]

//public int? Salary { get; set; }

// Display currency symbol. For country specific currency, set culture using globalization element in

//web.config. For Great Britain Pound symbol<globalization culture="en-gb"/>

**[**DataType**(**DataType.Currency**)]**

**public** **int**? Salary **{** **get**; **set**; **}**

// Display mailto hyperlink

**[**DataType**(**DataType.EmailAddress**)]**

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

// Generate a hyperlink

**[**DataType**(**DataType.Url**)]**

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

**}**

**}**

Make sure to include the following using statements:

**using System.ComponentModel.DataAnnotations;**  
**using System.ComponentModel;**

**Now Run the application and see the output as shown below.**



The **DisplayColumn**attribute is useful when a class has a property of the complex type, and we want to pick only one property of that complex object for display purposes.

**Let’s understand this with an example.**

Right-click on the “**Models**” folder and add a class file with the name **Company.cs** and then copy and paste the below code.

**namespace** *AttributesInMVC.Models*

**{**

**public** **class** Company

**{**

**public** Employee CompanyDirector

**{**

**get**

**{**

EmployeeDBContext dbContext = new EmployeeDBContext**()**;

**return** dbContext.Employees.Single**(**x =**>** x.Id == 1**)**;

**}**

**}**

**}**

**}**

Notice that, this class has **CompanyDirector**property which returns an **Employee** object. The employee is a complex type so; Employee object has got several properties. If we want FullName to be used for display purposes, then make the following changes.

**Modifying the ModifyEmployee.cs**

Modify the **ModifyEmployee.cs** file as shown below where we decorate the “**Employee**” partial class with the **DisplayColumn** attribute.

**namespace** *AttributesInMVC.Models*

**{**

**[**MetadataType**(**typeof**(**EmployeeMetaData**))]**

**[**DisplayColumn**(**"FullName"**)]**

**public** **partial** **class** Employee

**{**

**}**

**public** **class** EmployeeMetaData

**{**

**[**DisplayName**(**"Full Name"**)]**

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

**[**DataType**(**DataType.Date**)]**

**public** DateTime? HireDate **{** **get**; **set**; **}**

**[**DisplayFormat**(**NullDisplayText = "Gender not specified"**)]**

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

**[**DataType**(**DataType.Currency**)]**

**public** **int**? Salary **{** **get**; **set**; **}**

**[**DataType**(**DataType.EmailAddress**)]**

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

**[**DataType**(**DataType.Url**)]**

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

**}**

**}**

**Change the “Details” action method in the “Employee” controller class as shown below.**

**namespace** *AttributesInMVC.Controllers*

**{**

**public** **class** EmployeeController : Controller

**{**

**public** ActionResult Details**()**

**{**

Company company = new Company**()**;

**return** View**(**company**)**;

**}**

**}**

**}**

**Copy and paste the following code in Details.**cshtml**view**

@model AttributesInMVC.Models.Company

@{

ViewBag.Title = "Details";

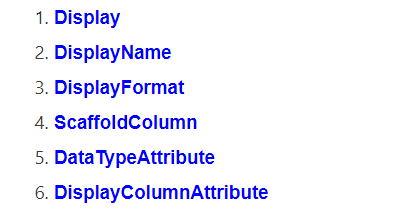
}

@Html.DisplayTextFor(x => x.CompanyDirector)

Navigate to **http://localhost:61449/Employee/Details** and you should see the FullName of the employee.

**Attributes in MVC (Part 2)**

**Attributes in ASP.NET MVC Application**



**UIHint Attribute in ASP.NET MVC Application:**

Let us understand how to open a page in the new browser window in asp.net MVC application. Along the way, we will also see how to use the UIHint attribute in ASP.NET MVC.

Modify the **ModifyEmployee.cs** class file in the Models folder as shown below.

**namespace** *AttributesInMVC.Models*

**{**

**[**MetadataType**(**typeof**(**EmployeeMetaData**))]**

**public** **partial** **class** Employee

**{**

**}**

**public** **class** EmployeeMetaData

**{**

**[**DataType**(**DataType.Url**)]**

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

**}**

**}**

**Modify the Details action method of EmployeeController as shown below**

**namespace** *AttributesInMVC.Controllers*

**{**

**public** **class** EmployeeController : Controller

**{**

**public** ActionResult Details**(int** id**)**

**{**

EmployeeDBContext dbContext = new EmployeeDBContext**()**;

Employee employee = dbContext.Employees.Single**(**x =**>** x.Id == id**)**;

**return** View**(**employee**)**;

**}**

**}**

**}**

**Modify the Details.cshtml view as shown below.**

@model AttributesInMVC.Models.Employee

@{

ViewBag.Title = "Details";

}

@Html.DisplayForModel()

At this point, build the application and navigate to **http://localhost:61449/Employee/Details/1**. When you click on the personal website link, the target page will open in the same window.

**How to open the page in a new window?**

If you want the page to be open in a new window, then please follow the below steps.

Right-click on the “**Views**” folder, and add the “**Shared**” folder if it does not exists. Then again, Right-click on the “**Shared**” folder and add a folder with the name “**DisplayTemplates**“. Next, Right-click on the “**DisplayTemplates”** folder, and add a view with the name “**Url.cshtml**“. Once you created the **Uri.cshtml** view, then please copy and paste the following code in it.

**<a href=”@ViewData.Model” target=”\_blank”>@ViewData.Model</a>**

That’s it. Build the application and click on the link. Now you will see that the page is opened in a new window. The downside of this approach is that from now on all the links, will open in a new window. To overcome this, please follow the below steps.

1. Rename **Url.cshtml** to **OpenInNewWindow.cshtml**
2. Decorate the “**PersonalWebSite**” property in **EmployeeMetaData**class with **UIHint** attribute and specify the name of the template to use. In our case, the name of the template is “**OpenInNewWindow**” as shown below.

**namespace** *AttributesInMVC.Models*

**{**

**[**MetadataType**(**typeof**(**EmployeeMetaData**))]**

**public** **partial** **class** Employee

**{**

**}**

**public** **class** EmployeeMetaData

**{**

**[**DataType**(**DataType.Url**)]**

**[**UIHint**(**"OpenInNewWindow"**)]**

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

**}**

**}**

So, the **UIHint**Attribute in ASP.NET MVC is used to specify the name of the template to use to display the data field on the model properties.

**Hidden Input Attribute in ASP.NET MVC:**

The HiddenInput attribute in ASP.NET MVC is used to generate an HTML element with the input type=hidden. This attribute is extremely useful when you don’t want the user to see or edit the property, but you need to post the property value to the server when the form is submitted, so the correct record can be updated. The Hidden Input attribute is present in the **System.Web.Mvc** namespace.

**Readonly Attribute in ASP.NET MVC:**

The ReadOnlyattribute is present in **System.ComponentModel** namespace. As the name suggests, this attribute is used to make a property read-only. Please note that we will still be able to change the property value on the view, but once we post the form the model binder will respect the read-only attribute and will not move the value to the property.  We can also, make the property of a class readonly simply, by removing the SET accessor

Let us understand the use of HiddenInput and Readonly Attribute with an example.

**Modifying the ModifyEmployee.cs file:**

Modify the ModifyEmployee.cs file as shown below. Please notice that the Id property is decorated with **HiddenInput** attribute and **EmailAddress** is decorated with the **ReadOnly** attribute.

**namespace** *AttributesInMVC.Models*

**{**

**[**MetadataType**(**typeof**(**EmployeeMetaData**))]**

**public** **partial** **class** Employee

**{**

**}**

**public** **class** EmployeeMetaData

**{**

// Id property is hidden and cannot be changed

**[**HiddenInput**(**DisplayValue = **false)]**

**public** **int** Id **{** **get**; **set**; **}**

// EmailAddress is read only

**[**ReadOnly**(true)]**

**[**DataType**(**DataType.EmailAddress**)]**

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

**[**ScaffoldColumn**(true)]**

**[**DataType**(**DataType.Currency**)]**

**public** **int**? Salary **{** **get**; **set**; **}**

**[**DataType**(**DataType.Url**)]**

**[**UIHint**(**"OpenInNewWindow"**)]**

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

**[**DisplayAttribute**(**Name = "Full Name"**)]**

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

**[**DisplayFormat**(**DataFormatString = "{0:d}"**)]**

**public** DateTime? HireDate **{** **get**; **set**; **}**

**[**DisplayFormat**(**NullDisplayText = "Gender not specified"**)]**

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

**}**

**}**

**Changes to EmployeeController.cs file**

**using** *System.Web.Mvc;*

**using** *AttributesInMVC.Models;*

**using** *System.Data.Entity;*

**namespace** *AttributesInMVC.Controllers*

**{**

**public** **class** EmployeeController : Controller

**{**

**public** ActionResult Details**(int** id**)**

**{**

EmployeeDBContext dbContext = new EmployeeDBContext**()**;

Employee employee = dbContext.Employees.Single**(**x =**>** x.Id == id**)**;

**return** View**(**employee**)**;

**}**

**public** ActionResult Edit**(int** id**)**

**{**

EmployeeDBContext dbContext = new EmployeeDBContext**()**;

Employee employee = dbContext.Employees.Single**(**x =**>** x.Id == id**)**;

**return** View**(**employee**)**;

**}**

**[**HttpPost**]**

**public** ActionResult Edit**(**Employee employee**)**

**{**

**if** **(**ModelState.IsValid**)**

**{**

EmployeeDBContext dbContext = new EmployeeDBContext**()**;

Employee employeeFromDB = dbContext.Employees.Single**(**x =**>** x.Id == employee.Id**)**;

// Populate all the properties except EmailAddrees

employeeFromDB.FullName = employee.FullName;

employeeFromDB.Gender = employee.Gender;

employeeFromDB.Age = employee.Age;

employeeFromDB.HireDate = employee.HireDate;

employeeFromDB.Salary = employee.Salary;

employeeFromDB.PersonalWebSite = employee.PersonalWebSite;

dbContext.Entry**(**employeeFromDB**)**.State = EntityState.Modified;

dbContext.SaveChanges**()**;

**return** RedirectToAction**(**"Details", new **{** id = employee.Id **})**;

**}**

**return** View**(**employee**)**;

**}**

**}**

**}**

**Create Edit.cshtml view and copy and paste the following code**

@model AttributesInMVC.Models.Employee

@{

ViewBag.Title = "Edit";

}

**<div** style="font-family:Arial"**>**

@using (Html.BeginForm())

{

@Html.EditorForModel()

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

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

**<input** type="submit" value="Save" **/>**

}

**</div>**

Run the application and navigate to **http://localhost:61449/Employee/Edit/1** and see everything is working as expected

**Action Selectors in ASP.NET MVC**

**Action Selectors in ASP.NET MVC Application**

1. **What are Action Selectors in ASP.NET MVC?**
2. **Types of Action Selectors in ASP.NET MVC**
3. **Understanding ActionName Action Selector in ASP.NET MVC**

**What are Action Selectors in ASP.NET MVC?**

The Actions are the public methods of a controller in ASP.NET MVC Application that responds to incoming HTTP Requests. The Action Selectors in ASP.NET MVC are the attributes that can be applied to the action methods of a controller and are used to control which action method gets invoked in response to a particular request. That means Action Selectors in ASP.NET MVC Framework help the routing engine to select the correct action method to handle a particular request.

**Types of Action Selectors in ASP.NET MVC.**

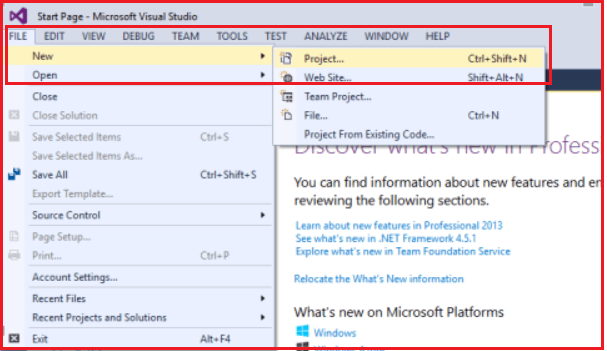
The ASP.NET MVC Framework provides the following three action selector attributes:

1. **ActionName**
2. **ActionVerbs**
3. **NonAction**

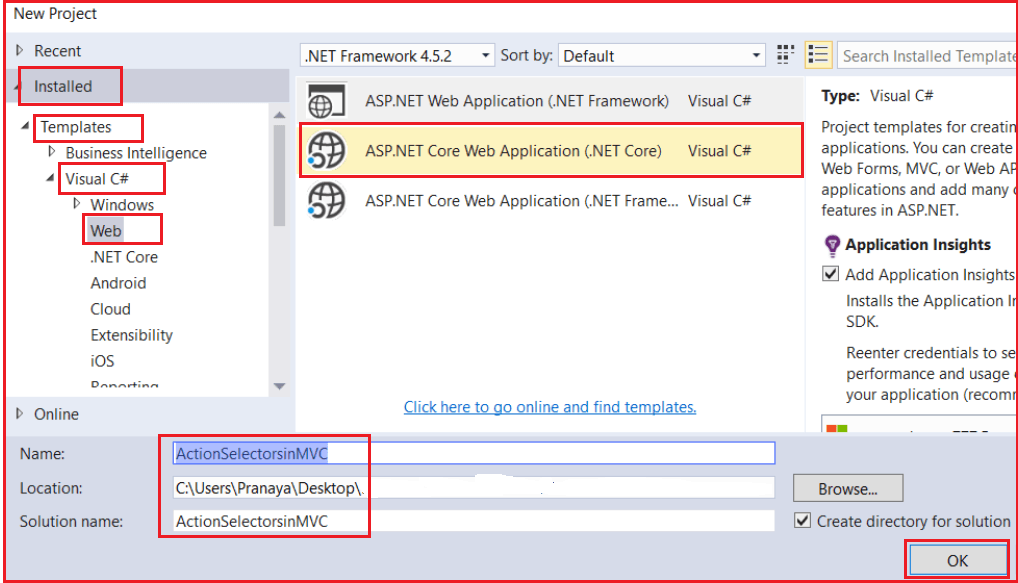
Let us understand the above Action Selectors with examples. To understand this let’s create an empty ASP.NET MVC application.

**Create an empty ASP.NET MVC application.**

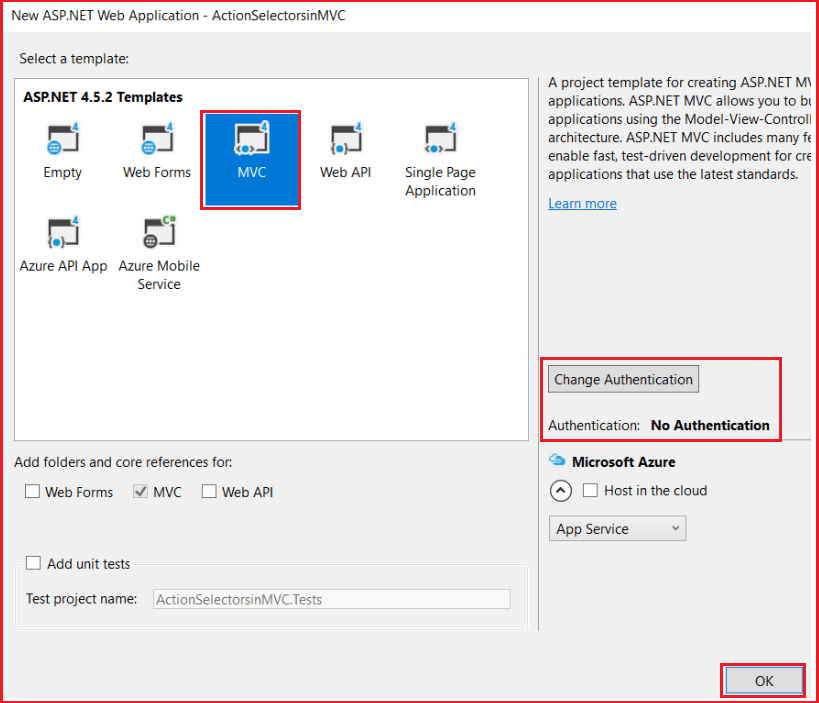
Open Visual Studio and create a new project by selecting **File => New => Project**option from the context menu as shown in the below image.



After clicking on the “**Project**” link a new dialog window will open. In that, we are going to select “**Web”** templates from the left pane, and from the middle pane we need to select the web template as “**ASP.NET Web Application**“. Provide the name and location and finally click on the **OK** button as shown in the below image.



Once you click on the **OK** button a new dialog will pop up with the name “**New ASP.NET Web Application**” for selecting the project Templates. From this dialog, we are going to choose the **MVC** project template and then we are going to choose the Authentication type for doing that just click on the **Change Authentication** button, a new dialog will pop up with the name “**Change Authentication**” here we are going to choose **No Authentication** and finally click on the OK button as shown below.



Once you click on the **OK** button, it will take some time to create the project for us.

**Adding Model classes in the Models folder:**

Right-click on the Models folder and click on **Add => Class** from the context menu and give the class name as **Product.cs.**Once you created the **Product.cs** class file, then copy and paste the following code into it.

**public** **class** Product

**{**

**public** **int** ProductId **{** **get**; **set**; **}**

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

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

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

**}**

**public** **class** DataAccessElectronics

**{**

**public** List**<**Product**>** GetDataElectronics**()**

**{**

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

**{**

new Product**()** **{** ProductId = 1, ProductName = "Desktop", Price = 34000, Category = "Electronics" **}**,

new Product**()** **{** ProductId = 2, ProductName = "Laptop", Price = 34000, Category = "Electronics" **}**,

new Product**()** **{** ProductId = 3, ProductName = "Router", Price = 34000, Category = "Electronics" **}**,

new Product**()** **{** ProductId = 4, ProductName = "Mouse", Price = 34000, Category = "Electronics" **}**,

new Product**()** **{** ProductId = 5, ProductName = "USB HDD", Price = 34000, Category = "Electronics" **}**,

new Product**()** **{** ProductId = 6, ProductName = "LCD", Price = 34000, Category = "Electronics" **}**

**}**;

**return** ElectronicsProductList;

**}**

**}**

The above code contains the following classes

1. **Product** – This is an entity class containing properties for Product information.
2. **DataAccessElectronics** – This class contains the method for returning all Electronics products.

**Note:** In a real-world example, the **Entity framework** can be used here to map with the database server.

**Understanding ActionName Action Selector in ASP.NET MVC:**

The ActionName action selector in the ASP.NET MVC Application is used when we want to invoke an action method with a different name, than what is already given to the action method. To understand the need and use of the ActionName selector, let’s modify the HomeController as shown below.

**public** **class** HomeController : Controller

**{**

**public** string Index**()**

**{**

**return** "Index action method invoked";

**}**

**}**

The URL “**/Home/Index**“ would invoke the Index() action method in HomeController. If you want to invoke the**Index()** action method, with the URL “**/Home/List”,** then you need to decorate the action method with the ActionName attribute as shown below.

**public** **class** HomeController : Controller

**{**

**[**ActionName**(**"List"**)]**

**public** string Index**()**

**{**

**return** "Index action method invoked";

**}**

**}**

Now, if you navigate to **/Home/Index**, you will get an error – “**The resource cannot be found**“. At the moment, the **Index**() action method is returning a string, but if it returns a view, should the view be named – **Index or List**? The List should be the view name.

**Let’s modify the Home Controller as shown below.**

Now, the Index Action Method returning a view.

**public** **class** HomeController : Controller

**{**

**[**ActionName**(**"List"**)]**

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

**{**

DataAccessElectronics DAE = new DataAccessElectronics**()**;

List**<**Product**>** ElectronicsList = DAE.GetDataElectronics**()**;

**return** View**(**ElectronicsList**)**;

**}**

**}**

Let’s add **Index.cshtml** within the Home folder and then copy and paste the following code into it.

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

@{

ViewBag.Title = "Index";

}

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

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

**<tr>**

**<th>**

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

**</th>**

**<th>**

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

**</th>**

**<th>**

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

**</th>**

**</tr>**

@foreach (var item in Model)

{

**<tr>**

**<td>**

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

**</td>**

**<td>**

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

**</td>**

**<td>**

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

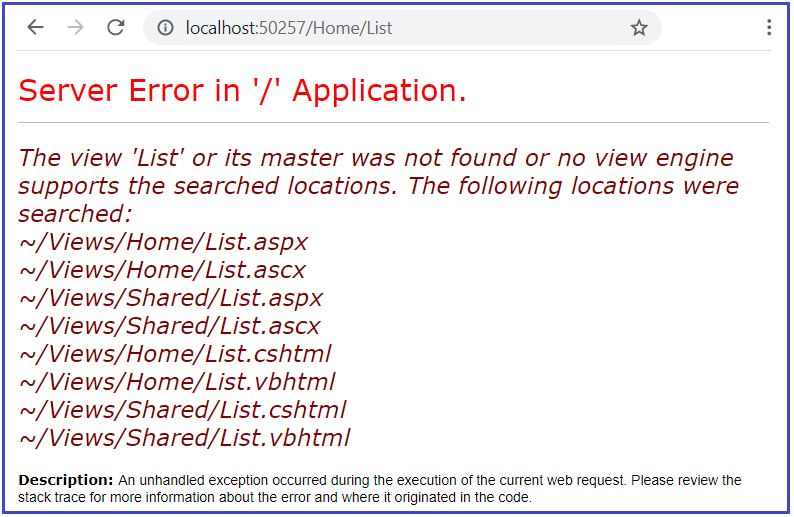
**</td>**

**</tr>**

}

**</table>**

Now run the application and navigate to Home/List and you will get the following error which proves that it will search for a view with the name List.cshtml, not Index.cshtml.



Now add a view with the name **List.cshtml** within the Home folder and copy and paste the following code into it.

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

@{

ViewBag.Title = "Index";

}

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

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

**<tr>**

**<th>**

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

**</th>**

**<th>**

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

**</th>**

**<th>**

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

**</th>**

**</tr>**

@foreach (var item in Model) {

**<tr>**

**<td>**

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

**</td>**

**<td>**

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

**</td>**

**<td>**

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

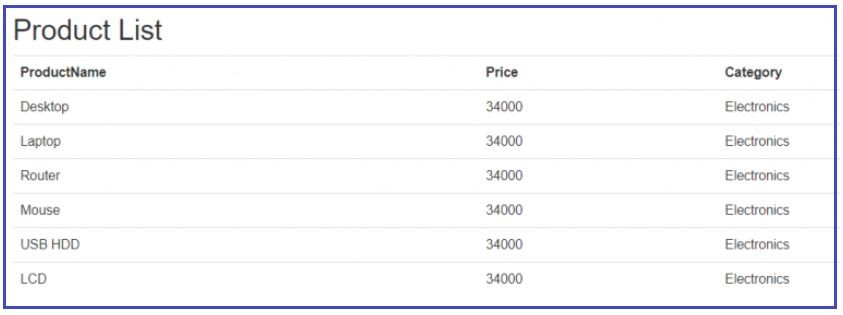
**</td>**

**</tr>**

}

**</table>**

**Now, run the application and navigate to /Home/List and it should display the following output as the response.**



Now for some reason, if you want to use **“Index”** as the view name then we need to use the other overloaded version of the View Extension method which takes the view name as a parameter. So, modify the controller action method as shown below. Here, you can notice, we have passed the first parameter as Index which is nothing but the view name.

**public** **class** HomeController : Controller

**{**

**[**ActionName**(**"List"**)]**

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

**{**

DataAccessElectronics DAE = new DataAccessElectronics**()**;

List**<**Product**>** ElectronicsList = DAE.GetDataElectronics**()**;

**return** View**(**"Index", ElectronicsList**)**;

**}**

**}**

With the above changes in place, now when you run the application and navigate to **/Home/List**, it will display the results by using the Index view.

**Action Verb Selector in ASP.NET MVC**

**Action Verb Action Selector in ASP.NET MVC Application**

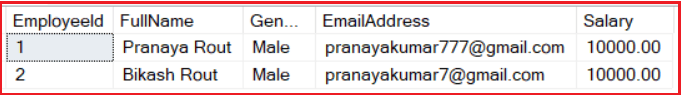
In this article, I am going to discuss the **Action Verb Action Selector in ASP.NET MVC** Application. Please read our previous article before proceeding to this article as we are going to work with the same example. In our previous article, we discussed the [**Action Name Action Selector in the ASP.NET MVC**](https://dotnettutorials.net/lesson/action-selectors-mvc/) Application. The Action Verb Selector is another selector that we can apply to the Action methods of a Controller.

**Action Verb Action Selector in ASP.NET MVC Application:**

We need to use the **Action Verb Selector**when we want to control the invocation of an action method based on the request type in the ASP.NET MVC Application. We can define two different action methods with the same name but one action method responds to an HTTP Get request while the other action method responds to an HTTP Post request.

**Database Required:**

We are going to use the following Employee table in this demo.



Please use the below SQL Script to create the Employee table with the required test data.

CREATE TABLE Employee**(**

**[**EmployeeId**]** **[int]** PRIMARY KEY IDENTITY**(**1,1**)**,

**[**FullName**]** **[**nvarchar**](**100**)**,

**[**Gender**]** **[**nvarchar**](**10**)**,

**[**EmailAddress**]** **[**nvarchar**](**100**)**,

**[**Salary**]** DECIMAL**(**18, 2**)**

**)**

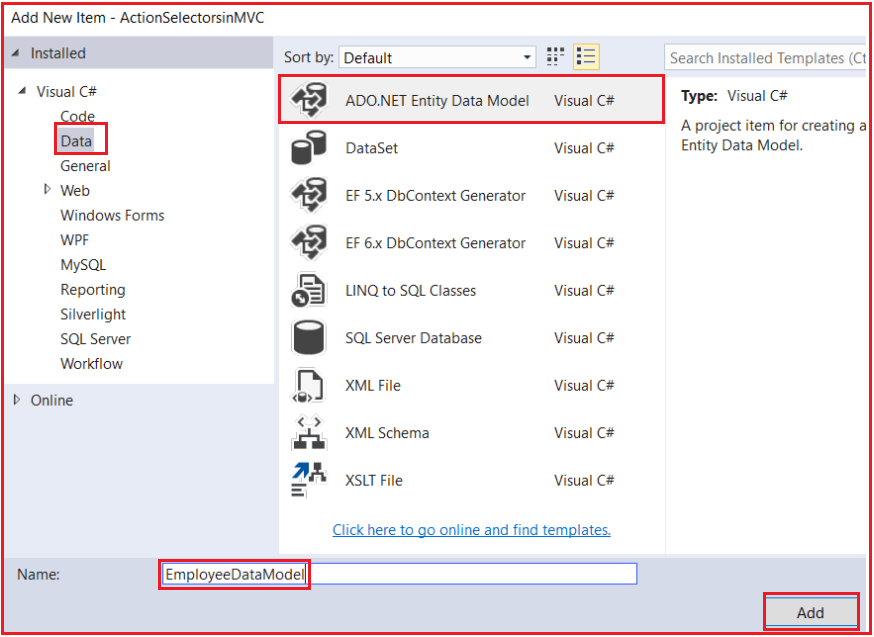
-- Populate the following test data

Insert **into** Employee values**(**'Pranaya Rout', 'Male', 'pranayakumar777@gmail.com', 10000**)**

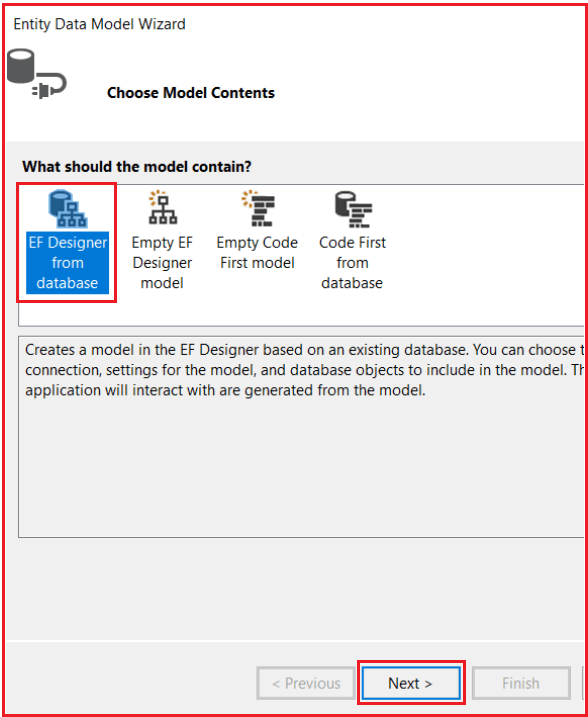
Insert **into** Employee values**(**'Bikash Rout', 'Male', 'pranayakumar7@gmail.com', 10000**)**

**Adding ADO.NET Entity Data Model:**

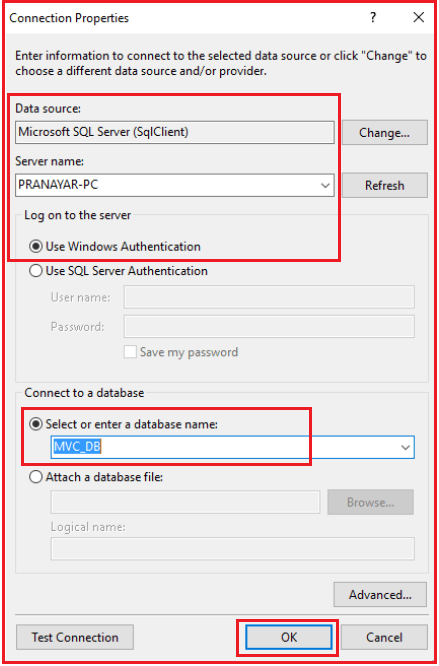
Right-click on the Models folder then select **Add => New Item** from the context menu. Then select **ADO.NET Entity Data Model**, provide a meaningful name such as **EmployeeDataModel** and click on the **Add** button as shown in the image below.



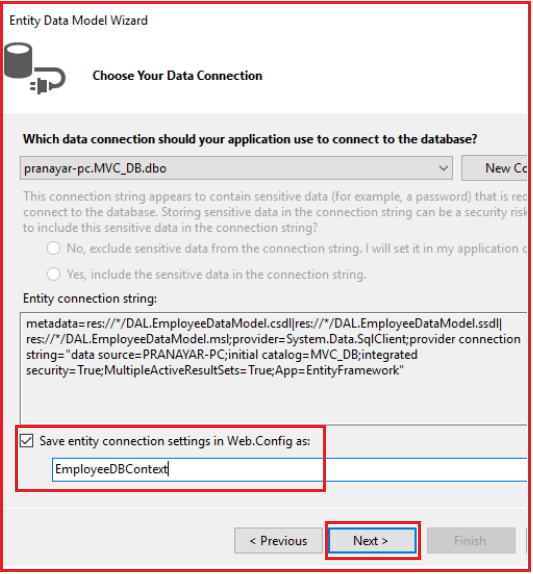
Here, we are going to use **Entity Framework Database First Approach** to interact with the database. So, from the **Choose Entity Model Wizard** window, select **Generate From Database** and click on the **Next** button as shown below.



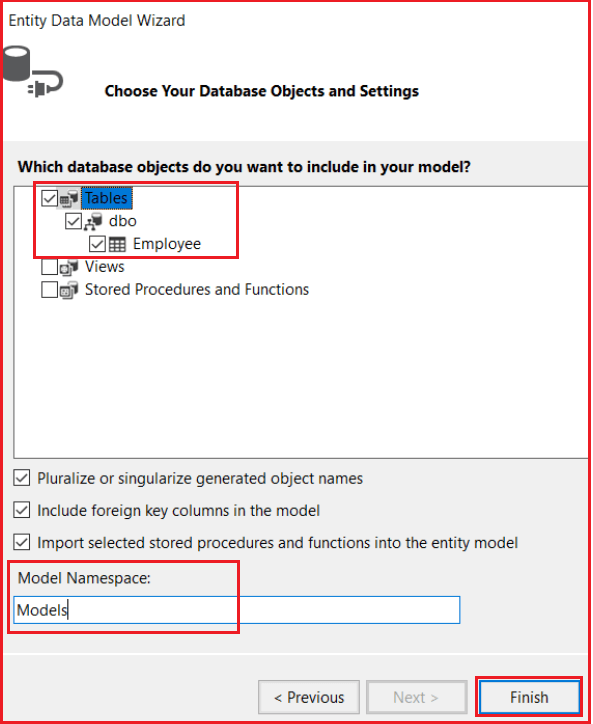
On the next screen, click on the **New Connection** tab and provide the necessary details to interact with the database. Then select the database and click on the **OK** button as shown below.



In the next step provide a meaningful name such as “**EmployeeDBContext**” for the Connection String that is going to create in the **Web.config** file and click on the **Next** button as shown below.



In the next step From Choose your database objects screen, choose the **Employee** table, provide the namespace name and click on the **Finish** button as shown below.



**Adding Employee Controller:**

Let’s Add Employee Controller to the Controller Folder and copy and paste the following code. In the below code, the “Edit” action method that is decorated with the **[AcceptVerbs(HttpVerbs.Get)]**accept verb responds to the GET request, whereas the “Edit” action method which is decorated with **[AcceptVerbs(HttpVerbs.Post)]**accept verb responds to POST request. The default is GET. So, if we don’t decorate an action method with any accept verb, then, by default, the method responds to the GET request. In our example Index is not decorated with any action verb so it by default responds to the GET attribute.

**namespace** *ActionSelectorsinMVC.Controllers*

**{**

**public** **class** EmployeeController : Controller

**{**

EmployeeDBContext dbContext = new EmployeeDBContext**()**;

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

**{**

List**<**Employee**>** ListEmployees = dbContext.Employees.ToList**()**;

**return** View**(**ListEmployees**)**;

**}**

**[**AcceptVerbs**(**HttpVerbs.Get**)]**

**public** ActionResult Edit**(int** id**)**

**{**

Employee employee = dbContext.Employees.Where**(**x =**>** x.EmployeeId == id**)**.FirstOrDefault**()**;

**return** View**(**employee**)**;

**}**

**[**AcceptVerbs**(**HttpVerbs.Post**)]**

**public** ActionResult Edit**(**Employee employee**)**

**{**

**if** **(**ModelState.IsValid**)**

**{**

dbContext.Entry**(**employee**)**.State = EntityState.Modified;

dbContext.SaveChanges**()**;

**return** RedirectToAction**(**"Index"**)**;

**}**

**return** View**(**employee**)**;

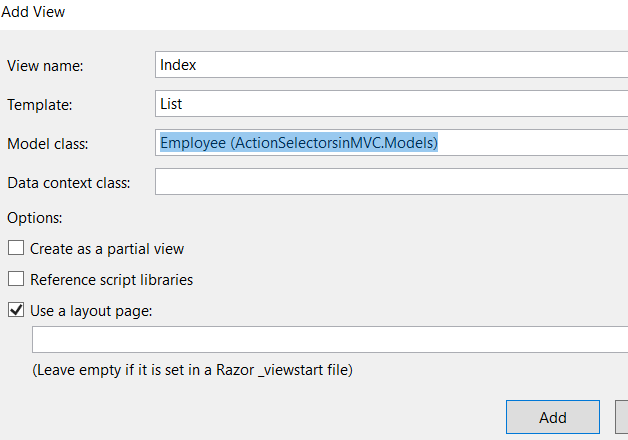
**}**

**}**

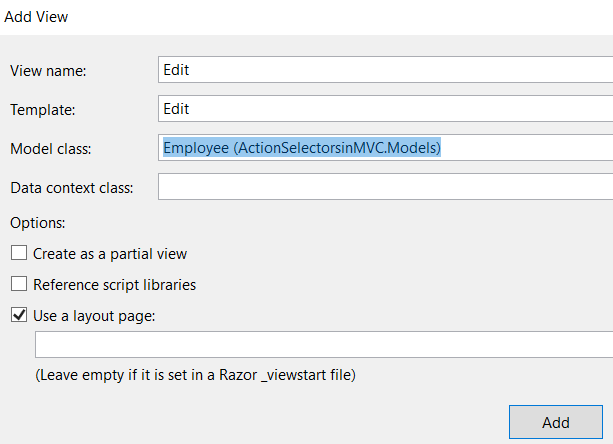
**}**

**Add Index and Edit view.**

Right-click on Index view and select Add View and provide the following details and click on the Add button as shown below.



Right-click on the Edit action method and click on Add View and provide the following details and click on the Add button as shown below.



**HttpGet and HttpPost Attribute in ASP.NET MVC:**

Instead of using **[AcceptVerbs(HttpVerbs.Get)]** and **[AcceptVerbs(HttpVerbs.Post)]** attribute, we can also use HttpGet and HttpPost attributes as shown in the below code. This is an alternative approach to using the AcceptVerbs attribute and the behavior is going to be the same i.e. the action method which decorates with the HttpGet attribute will only respond to GET Request whereas the action method which decorates with HttpPost attribute will only respond to the Post Request.

**public** **class** EmployeeController : Controller

**{**

EmployeeDBContext dbContext = new EmployeeDBContext**()**;

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

**{**

List**<**Employee**>** ListEmployees = dbContext.Employees.ToList**()**;

**return** View**(**ListEmployees**)**;

**}**

**[**HttpGet**]**

**public** ActionResult Edit**(int** id**)**

**{**

Employee employee = dbContext.Employees.Where**(**x =**>** x.EmployeeId == id**)**.FirstOrDefault**()**;

**return** View**(**employee**)**;

**}**

**[**HttpPost**]**

**public** ActionResult Edit**(**Employee employee**)**

**{**

**if** **(**ModelState.IsValid**)**

**{**

dbContext.Entry**(**employee**)**.State = EntityState.Modified;

dbContext.SaveChanges**()**;

**return** RedirectToAction**(**"Index"**)**;

**}**

**return** View**(**employee**)**;

**}**

**}**

**Non-Action Selector in ASP.NET MVC**

**Non-Action Selector in ASP.NET MVC Application**

**Why do we need a Non-Action Selector in the ASP.NET MVC Application?**

By default, in ASP.NET MVC, an action method is a public method in a controller that can be invoked using a URL request**.** But in some scenarios, we need to restrict the public methods of a controller to be invoked by a URL i.e. we don’t want the action method to be invoked using URL request.

To restrict access to the public methods in a controller, we need to decorate the action method with the Non-Action attribute. The Non-Action is another built-in Action Selector Attribute, which indicates that a public method of a Controller is not an action method. It is used when we want that method shouldn’t be treated as an action method.

**Understanding Non-Action Attribute in the ASP.NET MVC Application with an Example.**

**public** **class** HomeController : Controller

**{**

**[**HttpGet**]**

**public** string Method1**()**

**{**

**return** "<h1>Method 1 Invoked</h1>";

**}**

**[**HttpGet**]**

**public** string Method2**()**

**{**

**return** "<h1>Method 2 Invoked</h1>";

**}**

**}**

As you can see in the above code, we have 2 public methods i.e. Method1() and Method2() in HomeController. As the above two methods are public methods, so, we can access these two methods using the below URL.

Method1 can be invoked using URL /Home/Method1  
Method2 can be invoked using URL /Home/Method2

Let’s say **Method2**() is doing some internal work, and we don’t want it to be invoked using a URL request. To achieve this, we need to decorate **Method2**() with the **NonAction** attribute as shown in the below code.

**[**NonAction**]**

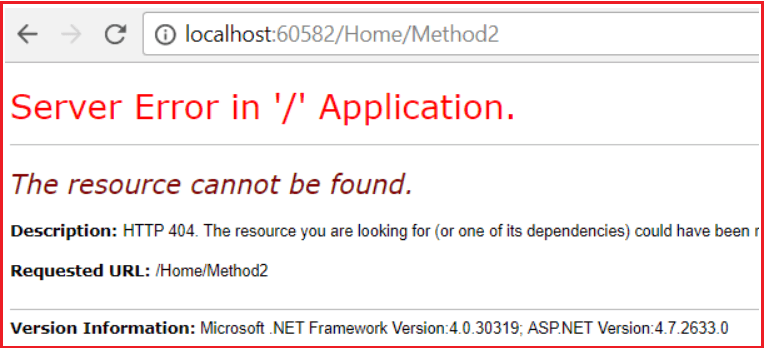
**public** string Method2**()**

**{**

**return** "<h1>Method 2 Invoked</h1>";

**}**

Now, if we navigate to URL **/Home/Method2**, we will get an error saying – **The resource cannot be found**as shown in the image below.



**Another way to restrict access to methods in a controller is by making them private.**

**private** string Method2**()**

**{**

**return** "<h1>Method 2 Invoked</h1>";

**}**

In general, it’s a bad design to have a public method in a controller that is not an action method. If we have any such method for performing business calculations, it should be somewhere in the model and not in the controller. However, if for some reason, if you want to have public methods in a controller and you don’t want to treat them as actions, then use the Non-Action Selector Attribute in ASP.NET MVC Application.