**HTML Helpers in ASP.NET MVC**

**HTML Helpers in ASP.NET MVC Application**

1. **Why HTML Helper in MVC?**
2. **What are HTML Helpers in ASP.NET MVC?**
3. **Is it possible to create our own custom HTML Helpers in ASP.NET MVC**
4. **Is it mandatory to use HTML Helpers in ASP.NET MVC Applications?**
5. **Types of HTML Helpers in MVC.**

**Why HTML Helper in ASP.NET MVC?**

In our traditional ASP.NET web forms application, as a developer, we generally use the toolbox for adding controls on any particular web page. However, coming to the ASP.NET MVC application there is no such toolbox available to drag and drop HTML controls onto the view. So those developers, who are coming from ASP.NET Web Forms backgrounds, find it a little difficult to create views in the ASP.NET MVC application.

So, to overcome the above difficulty, the ASP.NET MVC Framework provides HTML Helper classes that contain different extension methods to create different HTML Controls in a view. We can use those extension methods to create HTML controls programmatically within a view. All the HtmlHelper methods that are present within the HtmlHelper class generate HTML controls and return the result as an HTML string. If this is not clear at the moment then don’t worry we will discuss this in great detail.

**What are HTML Helpers in ASP.NET MVC?**

An HTML Helper in ASP.NET MVC is an extension method of the HTML Helper class which is used to generate HTML content in a view.  For example, if you want to generate a textbox with id=”firstname” and name=”firstname” then you can type all the required HTML in a view as shown below  
**<input type=”text” name=”firtsname” id=”firstname” />**

But in ASP.NET MVC, you can use the following “TextBox” HTML helper method in a view to generating a text box.   
**@Html.TextBox(“firstname”)**

The Point that you need to keep in mind is there are several overloaded versions available for the above TextBox HTML helper method. To set the value along with the name, you can use the following overloaded version of the TextBox helper method.  
**@Html.TextBox(“firstname”, “Pranaya”)**

At runtime, the above TextBox HTML helper method generates the following HTML  
**<input id=”firstname” name=”firstname” type=”text” value=”Pranaya” />**

It is also possible to set the HTML attributes of a text box. If you want to do so then you need to use the following overloaded version of the TextBox HTML helper method.   
**@Html.TextBox(“firstname”, “Pranaya”, new { style = “background-color:Red; color:White; font-weight:bold”, title=”Please enter your first name” })**

Notice here we are passing the HTML attributes title and style as an anonymous type to the TextBox helper method. Some of the HTML attributes are reserved keywords. For example, readonly, class, etc. If you want to use these attributes within a Helper method, then you need to prefix them with “**@**” symbol as shown in the below example.  
**@Html.TextBox(“firstname”, “Pranaya”, new { @class = “redtextbox”, @readonly=”true” })**

If you want to generate a label for “Name” using HTML helper method, then use the following HTML Helper method  
**@Html.Label(“Name”, “Name”)**

If you want to generate a textbox to enter a password then use the following HTML Helper method.  
**@Html.Password(“Password”)**

If you want to generate a multi-line textbox using Helper methods with 6 rows and 30 columns, then use the following HTML Helper method  
**@Html.TextArea(“Comments”, “”, 6, 30, null)**

If you want to generate a hidden field then use the following HTML Helper method  
**@Html.Hidden(“id”)**

The hidden field is used to store the hidden values which we don’t want to show to the end-users on a page but we need these values to update the data when the form is submitted to the server.

**Is it possible to create our own custom HTML Helpers in ASP.NET MVC?**

Yes, it is possible. We can create our custom HTML Helpers in ASP.NET MVC Application, and we will discuss this in our [**Custom HTML Helpers**](https://dotnettutorials.net/lesson/custom-html-helpers-mvc/) article.

**Is it mandatory to use HTML Helpers in ASP.NET MVC?**

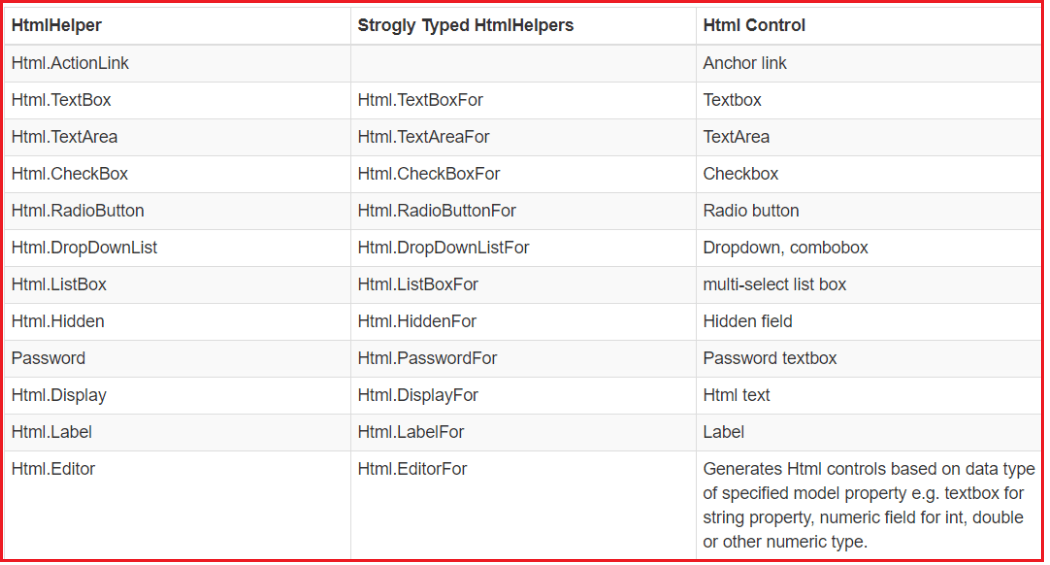
No, it is not mandatory to use HTML Helpers in ASP.NET MVC views. You can write the required HTML within a view but using HTML Helper extension methods greatly reduces the amount of HTML code that you write within a view. As per ASP.NET MVC documentation, the Views should be as simple as possible. All the complicated logic to generate the HTML Controls should be encapsulated into the HTML helper methods to keep views simple.

**Types of HTML Helpers Methods in ASP.NET MVC Application**

In ASP.NET MVC, there are two types of HTML Helper methods

1. **Simple HTML helper methods**
2. **Strongly type HTML helper Methods**

We will discuss what these are and the difference between them in a later article. The following table shows the lists of Html Helper methods and the corresponding HTML control they generate.



**TextBox HTML Helper in ASP.NET MVC**

**TextBox HTML Helper in ASP.NET MVC Application**

1. **How to Create TextBox using HTML Helper in MVC?**
2. **TextBox() HTML Helper Method in ASP.NET MVC.**
3. **TextBoxFor() HTML Helper Method in ASP.NET MVC application**
4. **What are the Differences between Html.TextBox and Html.TextBoxFor in MVC Application?**
5. **How to Create TexArea using HTML Helper in MVC?**
6. **Understanding TextArea() and TextAreaFor() HTML Helper Method in MVC.**

**How to Create TextBox using HTML Helper in MVC?**

To create a TextBox using HTML Helper Method in the ASP.NET MVC application, we need to use the TextBox Helper method. In the ASP.NET MVC application, we can use two different types of TextBox Helper methods to generates a textbox in a view. Those two extension methods are TextBox() and TextBoxFor(). The TextBox() HTML Helper method is a loosely typed method whereas the TextBoxFor() HTML Helper method is a strongly typed method.

**Let’s understand How to Create TextBox using HTML Helper in MVC with one example:**

Let’s create an ASP.NET MVC 5 application with the name **HTML\_HELPER**. Then Create **Employee.cs** class file within the **Models**folder and then copy and paste the following code into it.

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

**namespace** *HTML\_HELPER.Models*

**{**

**public** **class** Employee

**{**

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

**[**Display**(**Name = "Name"**)]**

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

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

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

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

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

**}**

**}**

We are going to use the above Employee model with **TextBox()** and **TextBoxFor()** HTML Helper methods. Once you create the Employee model next we are going to create an **MVC 5** **Empty Controller** with the name **EmployeeController** within the Controllers Folder.

**namespace** *HTML\_HELPER.Controllers*

**{**

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

**{**

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

**{**

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

**}**

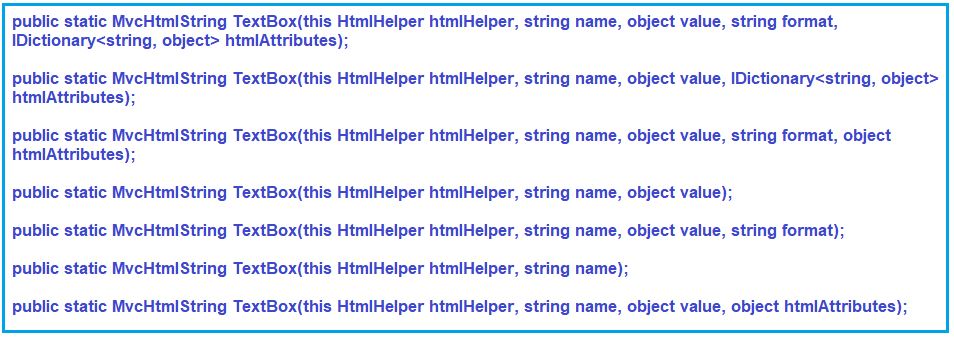
**}**

**}**

The **EmployeeController** is created with one action method i.e. the Index action method. So, create the respective Index view with empty HTML.

**TextBox() HTML Helper Method in ASP.NET MVC:**

The **Html.TextBox()** Helper method creates an element of **<input type=”text”>** with specified name, value and HTML attributes. There 7 overloaded versions of this **Html.TextBox()** Helper method is available as shown in the below image. The following method are loosely typed method



**Parameters:**

1. **htmlHelper**: The HTML helper instance that this method extends. It indicates that it is an extension method that belongs to HtmlHelper class.
2. **name**: The name of the form field and the System.Web.Mvc.ViewDataDictionary key that is used to look up the value.
3. **value**: The value of the text input element. The value is retrieved in this order – the System.Web.Mvc.ModelStateDictionary object, the value of this parameter, the System.Web.Mvc.ViewDataDictionary object, and lastly, a value attribute in the HTML attributes.
4. **format**: A string that is used to format the input.
5. **htmlAttributes**: An object that contains the HTML attributes to set for the element.

**Returns:** This method returns an input element whose type attribute is set to “text”.

**Note:** Replace **Home** with **Employee** in **RouteConfig** class.

**Modify the Index View:**

Please modify the Index View as shown below to use the TextBox Helper method.   
**@model HTML\_HELPER.Models.Employee**  
**@Html.TextBox(“EmployeeName”, null, new { @class = “form-control” })**

Run the application and inspect the Html and you will see that it will produce the following HTML for the textbox.  
**<input class=”form-control” id=”EmployeeName” name=”EmployeeName” type=”text” value=””>**

In the above example, the first parameter is “EmployeeName” which is a property of the Employee model which will be set as the name and id of the textbox. The second parameter is the value that we need to display in the textbox, which is null in the above example because the TextBox() method will automatically display the value of the EmployeeName property of the Employee model. The third parameter will be set as the class attribute. The HTML attributes parameter is an object type, so it can be an anonymous object, and the attributes name will be its properties starting with @ symbol.

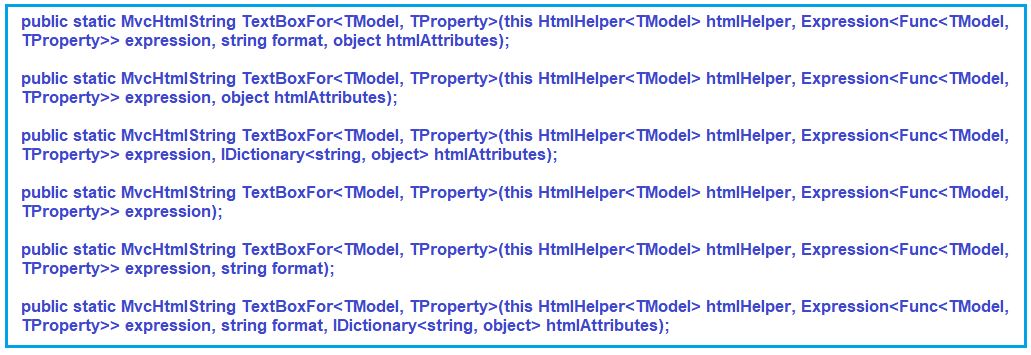
You can also specify any name for the textbox. However, it will not be bind to a model.  
**@Html.TextBox(“myTextBox”, “This is value”, new { @class = “form-control” })**

It will produce the following HTML  
**<input class=”form-control” id=”myTextBox” name=”myTextBox” type=”text” value=”This is value”>**

**TextBoxFor() HTML Helper Method in ASP.NET MVC application:**

The TextBoxFor() HTML Helper method is a strongly typed extension method. It generates an element of <input type=”text”> for the model property which needs to be specified using a lambda expression. The TextBoxFor() HTML Helper method binds the specified model object property to the input element. So it automatically displays that model property value within the textbox. The TextBoxFor() HTML Helper Method has 6 overloaded versions available in

ASP.NET MVC Framework are shown in the below image.



**Parameters:**

1. **htmlHelper**: The HTML helper instance that this method extends.
2. **expression**: An expression that identifies the object that contains the properties to display.
3. **format**: A string that is used to format the input.
4. **htmlAttributes**: An object that contains the HTML attributes to set for the element.

**Type parameters:**

1. **TModel**: The type of model.
2. **TProperty**: The type of the value.

**Returns**: An input element whose type attribute is set to “text”.

**Example to understand the TextBoxFor() HTML Helper Method in MVC:**

Copy and Paste the following code in Index.cshtml view  
**@model HTML\_HELPER.Models.Employee**  
**@Html.TextBoxFor(m => m.EmployeeName, new { @class = “form-control” })**

Run the application and insepct the element and you will see that, it will produce the following Html  
**<input class=”form-control” id=”EmployeeName” name=”EmployeeName” type=”text” value=””>**

In the above example, the first parameter in TextBoxFor() HTML Helper Method is a lambda expression which specifies the EmployeeName property of the Model object to bind with the textbox. It generates an input type text element with id and name property and both the properties value are set to EmployeeName. The value attribute will be set to the value of the EmployeeName property of the Model Object.

**What are the Differences between Html.TextBox and Html.TextBoxFor in ASP.NET MVC application?**

As we already discussed that the **@Html.TextBox()** is a loosely typed helper method whereas the **@Html.TextBoxFor()** is a strongly typed helper method.

The Html.TextBox() Helper method is not strongly typed and hence they don’t require a strongly typed view. This means that we can hardcode whatever name we want. On the other hand the Html.TextBoxFor() HTML Helper method is a strongly typed method and hence it requires a strongly typed view and the name should be given using the lambda expression. What is a strongly typed view that we will discuss in our upcoming articles?

The Strongly typed HTML helper methods also provide compile-time error checking. In real-time applications, we mostly prefer to use strongly typed HTML Helper methods.

But the most important point that we need to keep in mind is whether we use **Html.TextBox** Helper method or **Html.TextBoxFor()** Helper method the end result is the same that is they generate the same HTML.

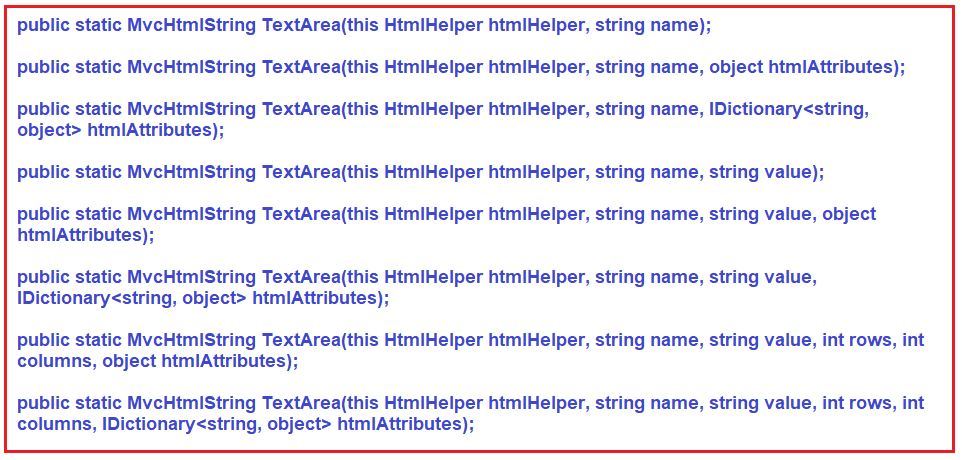
**Note**: The Strongly typed HTML helpers are introduced in MVC2.

**How to Create TexArea using HTML Helper in ASP.NET MVC Application?**

To create TextArea using HTML Helper in ASP.NET MVC application, we need to use the TextArea Helper methods.  The HtmlHelper class provides two extension methods to generate the textarea i.e. multiline textbox in a razor view: They are TextArea() and TextAreaFor(). By default, the textarea is created with rows=2 and cols=20 size.

**TextArea() HTML Helper Method in MVC:**

The **Html.TextArea()** method creates an input HTML element of **<textarea rows=”2″ cols=”20″>** with a specified name, value and HTML attributes. The **TextArea()** HTML Helper method is a loosely typed helper method because the name parameter is a string. The name parameter can also be the name of a property of a model object. It binds the specified property with the textarea. As a result, it automatically displays that property value within the textarea. There are also 8 overloaded versions available for TextArea() 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. **name**: The name of the form field to return.
3. **value**: The text content.
4. **rows**: The number of rows.
5. **columns**: The number of columns.
6. **htmlAttributes**: An object that contains the HTML attributes to set for the element.

**Returns**: Returns the specified textarea element by using the specified HTML helper, the name of the form field, the text content, the number of rows and columns, and the specified HTML attributes.

**Example: TextArea HTML Helper Method in ASP.NET MVC**

**Let’s modify our Employee Model as shown below**

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

**namespace** *HTML\_HELPER.Models*

**{**

**public** **class** Employee

**{**

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

**[**Display**(**Name = "Name"**)]**

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

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

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

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

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

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

**}**

**}**

**Modify the Employee Controller as shown below:**

**namespace** *HTML\_HELPER.Controllers*

**{**

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

**{**

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

**{**

Employee emp = new Employee**()**

**{**

EmployeeId = 1,

Address = "Andheri, Sakinaka, Mumbai, 400097, Maharashtra, India"

**}**;

**return** View**(**emp**)**;

**}**

**}**

**}**

**Modify the Index View:**

Copy and Paste the following code in the Index view  
**@model HTML\_HELPER.Models.Employee**  
**@Html.TextArea(“Address”, null, new { @class = “form-control” })**

If you inspect the text area, then you will see that it will produce the following HTML  
**<textarea class=”form-control” cols=”20″ id=”Address” name=”Address” rows=”2″></textarea>**

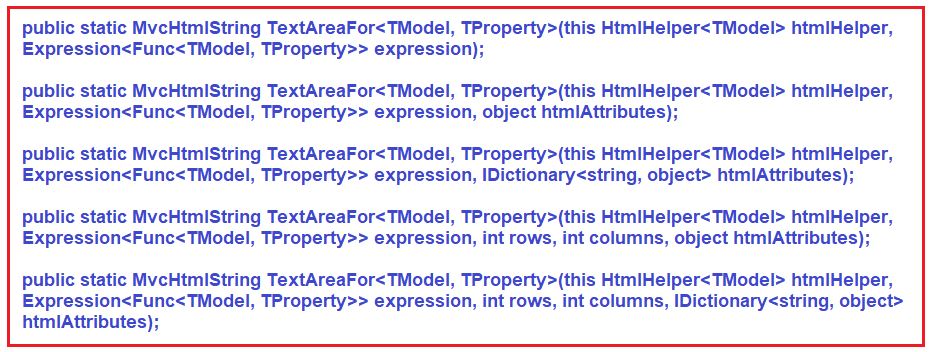
In the above example, the first parameter is the “**Address**” property of the **Employee** model class which will be set as name and id of the textarea. The second parameter is the **value** to display in the textarea, which is null in the above example because **TextArea()** method will automatically display a value of the Address property in the textarea. The third parameter will be set as a class attribute. The **HtmlAttributes** parameter is an object type, so it can be an anonymous object, and attributes name will be its properties starting with @ symbol.

We can also specify any name for the textarea. However, it will not be bound to a model.  
**@Html.TextArea(“myTextArea”, “This is value”, new { @class = “form-control” })**

It will produce the following html  
**<textarea class=”form-control” cols=”20″ id=”myTextArea” name=”myTextArea” rows=”2″>This is value</textarea>**

**TextAreaFor() HTML Helper Method in ASP.NET MVC:**

The **TextAreaFor()** HTML Helper method is a strongly typed extension method. It generates a text area input element. We need to specify the name using a lambda expression. The **TextAreaFor()** HTML Helper Method binds that specified model object property to the textarea element. So it automatically displays that model property value within the textarea. There are also 5 overloaded versions available for TextAreaFor() 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. **expression**: An expression that identifies the object that contains the properties to render.
3. **rows**: The number of rows.
4. **columns**: The number of columns.
5. **htmlAttributes**: A dictionary that contains the HTML attributes to set for the element.

**Type parameters:**

1. **TModel**: The type of model.
2. **TProperty**: The type of property.

**Returns**: Returns an HTML textarea element for each property in the object that is represented by the specified expression using the specified HTML attributes and the number of rows and columns.

**Example: TextAreaFor HTML Helper Method in ASP.NET MVC:**

Copy and paste the following code in the index view  
**@model HTML\_HELPER.Models.Employee**  
**@Html.TextAreaFor(m => m.Address, new { @class = “form-control” })**

It will produce the following HTML  
**<textarea class=”form-control” cols=”20″ id=”Address” name=”Address” rows=”2″>Andheri, Sakinaka, Mumbai, 400097, Maharashtra, India</textarea>**

In the above example, the first parameter in TextAreaFor() method is a lambda expression that specifies the model property to be bound with the textarea element. In our example, we have specified the Address property. So, it generates the input type <textarea> element with id and name property set to the property name – Address. The value of textarea will be set to the value of Address property.

**DropDownList HTML Helper in ASP.NET MVC**

**DropDownList HTML Helper in ASP.NET MVC**

In this article, I am going to discuss **DropDownList HTML Helper in ASP.NET MVC** Application. Please read our previous article before proceeding to this article where we discussed [**How to Create TextBox and TextArea using HTML Helper Methods**](https://dotnettutorials.net/lesson/textbox-html-helper-mvc/) in MVC Application. In order to generate a DropDownList in ASP.NET MVC Application, we need to use the DropDownListHTML Helper Method. As part of this article, we are going to discuss the following pointers.

1. **What is a DropDownList?**
2. **Understanding DropDownList() HTML Helper Method in ASP.NET MVC.**
3. **How to set the Dropdown list values from the database in the ASP.NET MVC Application?**
4. **How to use Enum to set the Dropdown list values in MVC?**
5. **Understanding DropDownListFor HTML Helper in ASP.NET MVC Application.**

**What is a DropDownList?**

A DropDownList in ASP.NET MVC application is nothing but a collection of SelectListItem objects. Depending on your business requirement you may either hard code the values or you may retrieve the values from a database table. In this article, I am going to discuss both approaches. First, we will discuss creating the DropDownList using the hard-coded value then we will see how to create the DropDownList with the values coming from a database.

The HtmlHelper class in MVC provides two extension methods to generate the <select> element in a view. They are

1. **DropDownList() Helper Method**
2. **DropDownListFor() Helper Method**

**DropDownList() HTML Helper Method in ASP.NET MVC:**

There are 8 overloaded versions available for the DropDownList HTML Helper method as shown in the below image.



**Parameters:**

1. **htmlHelper**: The HTML helper instance that this method extends.
2. **name**: The name of the form field to return.
3. **selectList**: A collection of System.Web.Mvc.SelectListItem objects that are used to populate the drop-down list.
4. **optionLabel**: The text for a default empty item. This parameter can be null.
5. **htmlAttributes**: An object that contains the HTML attributes to set for the element.

**Returns**: Returns a single-selection select element using the specified HTML helper, the name of the form field, the specified list items, an option label, and the specified HTML attributes.

**Example: DropDownList HTML Helper Method in ASP.NET MVC Application**

The following code will generate a department dropdown list. The first item in the drop-down list will be **“Select Department”**.

@Html.DropDownList("Departments", new List**<SelectListItem>**

{

new SelectListItem { Text = "IT", Value = "1", Selected=true},

new SelectListItem { Text = "HR", Value = "2"},

new SelectListItem { Text = "Payroll", Value = "3"}

}, "Select Department")

**The downside of hard-coding the DropDownList value within the code itself is that if we have to add or remove departments from the DropDownList then the code needs to be modified each and every time.**

**How to set the Dropdown list values from the database in the ASP.NET MVC Application?**

Most of the time or in real-time applications, we generally get the data from a database. To understand this let’s create a Department Class and then populate the values within the controller as shown below. Add Department.cs class file in Models folder as shown below.

**namespace** *HTML\_HELPER.Models*

**{**

**public** **class** Department

**{**

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

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

**}**

**}**

**To pass the list of Departments from the controller store them in a “ViewBag” as shown below**

**namespace** *HTML\_HELPER.Controllers*

**{**

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

**{**

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

**{**

//Get the data from the database

//Here we are creating Department list

List**<**Department**>** ListDepartments = new List**<**Department**>()**

**{**

new Department**()** **{**Id = 1, Name="IT" **}**,

new Department**()** **{**Id = 2, Name="HR" **}**,

new Department**()** **{**Id = 3, Name="Payroll" **}**,

**}**;

// Retrieve departments and build SelectList

ViewBag.Departments = new SelectList**(**ListDepartments, "Id", "Name"**)**;

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

**}**

**}**

**}**

**Or you can also do the same thing in the following way**

**namespace** *HTML\_HELPER.Controllers*

**{**

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

**{**

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

**{**

List**<**SelectListItem**>** items = new List**<**SelectListItem**>()**;

items.Add**(**new SelectListItem **{** Text = "IT", Value = "1" **})**;

items.Add**(**new SelectListItem **{** Text = "HR", Value = "2" **})**;

items.Add**(**new SelectListItem **{** Text = "Payroll", Value = "2" **})**;

ViewBag.Departments = items;

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

**}**

**}**

**}**

**Now in the “Index” view access the Department list from “ViewBag” as shown below**

**@Html.DropDownList(“Departments”, @ViewBag.Departments as List<SelectListItem>, “Select Department”,new { @class = “form-control”})**

**If you inspect the dropdown list then it will generate the below code**

**<select** class="form-control" id="Departments" name="Departments"**>**

**<option** value=""**>**Select Department**</option>**

**<option** value="1"**>**IT**</option>**

**<option** value="2"**>**HR**</option>**

**<option** value="2"**>**Payroll**</option>**

**</select>**

In the above example, the first parameter is the property name for which we want to display the list of items. The second parameter is the list of values which are going to be displayed within the DropDownList. Here we have used the ViewBag mechanism to get the department values. The third parameter is the label which is nothing but the first item in the drop-down list and the fourth parameter is for the Html attributes like CSS to be applied on the dropdown list.

**How to use Enum to set the Dropdown list values in ASP.NET MVC Application?**

Let’s see how to use Enum to set the Dropdown list values. In this example, we are going to set the Gender Values from the enum.

**In Department.cs file add the following enum**

**public** enum Gender

**{**

Male,

Female

**}**

**Copy and paste the following code in the index view**

@using HTML\_HELPER.Models

@Html.DropDownList("EmployeeGender",

new SelectList(Enum.GetValues(typeof(Gender))),

"Select Gender",

new { @class = "form-control" })

**When we run the application, it will generate the following HTML**

**<select** class="form-control" id="EmployeeGender" name="EmployeeGender"**>**

**<option** value=""**>**Select Gender**</option>**

**<option>**Male**</option>**

**<option>**Female**</option>**

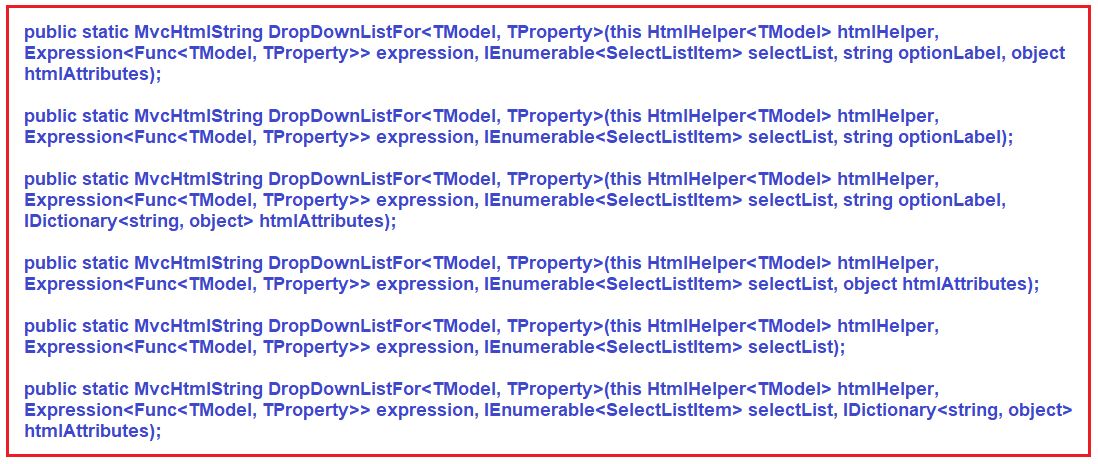
**</select>**

In the above example, the first parameter is the property name for which we want to display the list items. The second parameter is the list of values which is going to be displayed in the drop-down list. We have used Enum methods to get Gender enum values. The third parameter is the label which will be the first list item in the drop-down list and the fourth parameter is for the Html attributes like CSS to be applied on the dropdown list.

Please note that you can add **@using HTML\_HELPER.Models** namespace into the **<namespaces> section in web.config** which is present within the Views folder instead of using **@using t**o include the namespace in all the views.

**DropDownListFor HTML Helper in ASP.NET MVC Application:**

The DropDownListFor() HTML Helper method is a strongly typed extension method. This helper method is used to generate an <select> element for the property which is needed to be specified by using a lambda expression. The DropDownListFor() HTML Helper method will bind a specified property of a model object to the drop-down list control. As a result, it automatically displays the list items in the drop-down list. There are 6 overloaded versions available for the DropDownListFor 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. **expression**: An expression that identifies the object that contains the properties to display.
3. **selectList**: A collection of System.Web.Mvc.SelectListItem objects that are used to populate the drop-down list.
4. **optionLabel**: The text for a default empty item. This parameter can be null.
5. **htmlAttributes**: An object that contains the HTML attributes to set for the element.

**Type parameters:**

1. **TModel**: The type of model.
2. **TProperty**: The type of value.

**Returns:** Returns an HTML select element for each property in the object that is represented by the specified expression using the specified list items, option label, and HTML attributes.

**Example: DropDownListFor HTML Helper Method in ASP.NET MVC Application**

We are going to use the following models to understand the DropDownListFor HTML Helper.

**Employee.cs**

**namespace** *HTML\_HELPER.Models*

**{**

**public** **class** Employee

**{**

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

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

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

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

**}**

**}**

**Department.cs**

**namespace** *HTML\_HELPER.Models*

**{**

**public** **class** Department

**{**

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

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

**}**

**public** enum Gender

**{**

Male,

Female

**}**

**}**

**Modify the EmployeeController as shown below.**

**namespace** *HTML\_HELPER.Controllers*

**{**

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

**{**

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

**{**

//Lets create list department for dropdownlist

List**<**Department**>** ListDepartments = new List**<**Department**>()**

**{**

new Department**()** **{**Id = 1, Name="IT" **}**,

new Department**()** **{**Id = 2, Name="HR" **}**,

new Department**()** **{**Id = 3, Name="Payroll" **}**,

**}**;

ViewBag.Departments = ListDepartments;

//lets create one employee

Employee emp = new Employee**()**

**{**

EmployeeId = 1,

EmployeeName = "Pranaya",

Gender = "Male",

DepartmentID = 1

**}**;

//Pass that employee to the view

**return** View**(**emp**)**;

**}**

**}**

**}**

**Modify the Index.cshtml file as shown below**

@using HTML\_HELPER.Models

@model Employee

@Html.DropDownListFor(emp => emp.Gender,

new SelectList(Enum.GetValues(typeof(Gender))),

"Select Gender",

new { @class = "form-control" })

@Html.DropDownList("Department",

new SelectList(ViewBag.Departments, "Id", "Name"),

"Select Department",

new { @class = "form-control" })

**Or if you want to show the selected Department of that particular employee then copy and paste the following code**

@Html.DropDownListFor(emp => emp.DepartmentID,

new SelectList(ViewBag.Departments, "Id", "Name"),

"Select Department",

new { @class = "form-control" })

@Html.DropDownList("DepartmentID",

new SelectList(ViewBag.Departments, "Id", "Name", Model.DepartmentID),

"Select Department",

new { @class = "form-control" })

In the above example, the first parameter in **DropDownListFor() HTML Helper** method is a lambda expression that specifies the model property to be bind with the select element. We have specified Gender property of enum type and DepartmentID property. The second parameter specifies the items to show into the dropdown list using SelectList. The third parameter is the option Label which will be the first item of the drop-down list.

**RadioButton HTML Helper in MVC**

**RadioButton HTML Helper in ASP.NET MVC Application**

1. **What is a Radion Button in MVC?**
2. **How to generate radio button in ASP.NET MVC?**
3. **RadioButton HTML Helper Method in ASP.NET MVC**
4. **RadioButtonFor HTML Helper in ASP.NET MVC.**

**What is a Radion Button in ASP.NET MVC?**

A radio button is an element of the graphical user interface (GUI) and its main purpose is to allow a user to select a single item from a predefined list of items.

**How to generate radio button in ASP.NET MVC?**

The HtmlHelper class provides two extension methods which are used to generate a **<input type=”radio”>** element in a view. The two methods are

1. **RadioButton()**
2. **RadioButtonFor()**.

**RadioButton HTML Helper Method in ASP.NET MVC:**

The Html.RadioButton() HTML Helper method in ASP.NET MVC Application is used to create a radio button element with a specified name, isChecked boolean property, and the HTML attributes. The Html.RadioButton() HTML Helper method is a loosely type method. There are 6 overloaded versions of RadioButton HTML Helper methods are available as shown in the below image.



**Parameters:**

1. **htmlHelper**: The HTML helper instance that this method extends.
2. **name**: The name of the form field and the System.Web.Mvc.ViewDataDictionary key that is used to look up the value.
3. **value**: The value of the selected radio button. The value is retrieved in this order- the System.Web.Mvc.ModelStateDictionary object, the value of this parameter, the System.Web.Mvc.ViewDataDictionary object, and lastly, a value attribute in the html attributes.
4. **isChecked**: true to select the radio button; otherwise, false.
5. **htmlAttributes**: An object that contains the HTML attributes to set for the element.

**Returns:** Returns a radio button input element that is used to present mutually exclusive options

**Example to understand RadioButton Helper Method:**

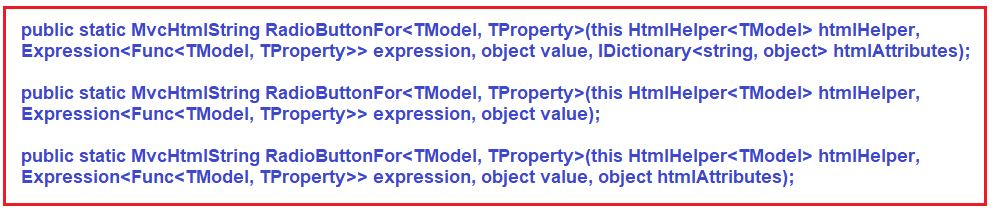
**Male: @Html.RadioButton(“Gender”, “Male”)**  
**Female: @Html.RadioButton(“Gender”, “Female”)**

If you inspect the HTML, then you will see the following HTML  
**Male: <input checked=”checked” id=”Gender” name=”Gender” type=”radio” value=”Male” />**  
**Female: <input id=”Gender” name=”Gender” type=”radio” value=”Female” />**

In the above example, we have created two radio buttons **Male** and **Female** for the “**Gender**” property. The first parameter is the group name. The point that you need to keep in mind is that you need to give the same name for both the radio button. The second parameter is the value of the radio button which will be sent to the server when the respective radio button is checked. That means if the **Male** radio button is selected, then the string value “Male” will be assigned to a model property with the name **Gender** and submitted to the server.

**RadioButtonFor HTML Helper in ASP.NET MVC:**

The RadioButtonFor HTML Helper method is the strongly typed extension method. It is also used to generate an <input type=”radio”> element in a view.  Here we need to specify the name for the Radion Button using a lambda expression. The RadioButtonFor HTML Helper method binds a specified model object property to RadioButton control. As a result, it automatically checked or unchecked the RadioButton based on the property value. There are 3 overloaded versions of RadioButtonFor HTML Helper methods are available in ASP.NET MVC as shown in the below image.



**Parameters:**

1. **htmlHelper**: The HTML helper instance that this method extends.
2. **expression**: An expression that identifies the object that contains the properties to render.
3. **value**: The value of the selected radio button. The value is retrieved in this order – the System.Web.Mvc.ModelStateDictionary object, the value of this parameter, the System.Web.Mvc.ViewDataDictionary object, and lastly, a value attribute in the HTML attributes.
4. **htmlAttributes**: A dictionary that contains the HTML attributes to set for the element.

**Type parameters:**

1. **TModel**: The type of model.
2. **TProperty**: The type of value.

**Returns**: An HTML input element whose type attribute is set to “radio” for each property in the object that is represented by the specified expression, using the specified HTML attributes.

**Example: RadioButtonFor HTML Helper in ASP.NET MVC**

To understand this let’s create a new project. Right-click on the “**Models**” folder and then add two class files with the name **Company.cs** and **Department.cs**. Once you create the class files then copy and paste the following code.

**Department.cs**

**namespace** *HTML\_HELPER.Models*

**{**

**public** **class** Department

**{**

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

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

**}**

**}**

**Company.cs**

**namespace** *HTML\_HELPER.Models*

**{**

**public** **class** Company

**{**

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

**public** List**<**Department**>** Departments

**{**

**get**

**{**

List**<**Department**>** ListDepartments = new List**<**Department**>()**

**{**

new Department**()** **{**Id = 1, Name="IT" **}**,

new Department**()** **{**Id = 2, Name="HR" **}**,

new Department**()** **{**Id = 3, Name="Manager" **}**,

**}**;

**return** ListDepartments;

**}**

**}**

**}**

**}**

**Creating Controller:**

Create a Controller with the name HomeController within the Controllers Folder and then copy and paste the following two action methods in it.

**namespace** *HTML\_HELPER.Controllers*

**{**

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

**{**

**[**HttpGet**]**

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

**{**

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

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

**}**

**[**HttpPost**]**

**public** string Index**(**Company company**)**

**{**

**if** **(**string.IsNullOrEmpty**(**company.SelectedDepartment**))**

**{**

**return** "You did not select any department";

**}**

**else**

**{**

**return** "You selected department with ID = " + company.SelectedDepartment;

**}**

**}**

**}**

**}**

**Creating Views:**

Right-click on the **“Index”** action method in **“HomeController”** and add a view with the name **“Index”**. Once you create the Index view then copy and paste the following code in it.

@model HTML\_HELPER.Models.Company

@{

ViewBag.Title = "Index";

}

**<h2>**Index**</h2>**

@using (Html.BeginForm())

{

foreach (var department in Model.Departments)

{

@Html.RadioButtonFor(m => m.SelectedDepartment, department.Id)@department.Name

}

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

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

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

}

**Testing the application:**

Run the application and inspect the html. Once you inspect the HTML you will see that the following HTML codes are generated by the RadionButton helper method.

**<input id=”SelectedDepartment” name=”SelectedDepartment” type=”radio” value=”1″>**  
**<input id=”SelectedDepartment” name=”SelectedDepartment” type=”radio” value=”2″>**  
**<input id=”SelectedDepartment” name=”SelectedDepartment” type=”radio” value=”3″>**

Now click on the **“Submit”** button without selecting any department. Notice that you are getting a message stating you have not selected any department. On the other hand, if you select a department and then click on the **“Submit”**button, then you will see the selected department ID.

In the above example, the first parameter in RadioButtonFor() HTML Helper method is a lambda expression that specifies the model property to be bound with the RadioButton element. We have created radio buttons for the SelectedDepartment property in the above example. So, it generates three **<input type=”Radio”>** element with id and name set to property name – SelectedDepartment. The second parameter is the value which will be sent to the server when the form will be submitted.

**CheckBox HTML Helper in ASP.NET MVC**

**Creating CheckBox using CheckBox HTML Helper in ASP.NET MVC**

1. **Why we need CheckBox?**
2. **Understanding the CheckBox HTML Helper in ASP.NET MVC Application**
3. **CheckBox HTML Helper in MVC Application**

**Why we need CheckBox?**

When we allow the users to select multiple options from the available options then we need to create a checkbox list. In order to create a checkbox list in ASP.NET MVC Application, we are provided with the CheckBox HTML Helper method.

**Understanding the CheckBox HTML Helper in ASP.NET MVC Application**

The HtmlHelper class provides two extension methods to generate a **<input type=”checkbox”>** element in an ASP.NET MVC view. They are as follows:

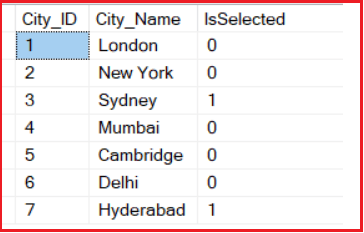
1. **CheckBox()**
2. **CheckBoxFor()**.

The **Html.CheckBox()** is a loosely typed method which generates a **<input type=”checkbox”>** with the specified name, isChecked boolean and html attributes.

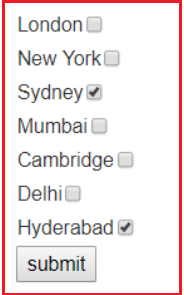
The **Html.CheckBoxFor()** HTML helper method is a strongly typed extension method. It also generates an **<input type=”checkbox”>** element for the model property which needs to be specified using a lambda expression. The **CheckBoxFor()** HTML Helper method binds a specified model object property to the checkbox element. As a result, it automatically checked or unchecked a checkbox based on that specified model property value.

**CheckBox HTML Helper in MVC Application:**

To understand the **CheckBox HTML Helper in MVC,** we are going to use the following “**City**” table.



 In the user interface, we need to create checkboxes for each city as shown in the below image.



Our requirement is when we select the checkboxes and click on the submit button then all the selected checkbox values should display as “**You selected – Checkbox values**” and if we don’t select any checkbox then the message should be “**You have not selected any City**” display.

**Please use the following SQL script to create table City and populate with the required test data**

**CREATE** **TABLE** City

(

City\_ID **INT** **IDENTITY** **PRIMARY** **KEY**,

City\_Name **NVARCHAR**(100) NOT **NULL**,

IsSelected **BIT** NOT **NULL**

)

**Insert** **into** City values ('London', 0)

**Insert** **into** City values ('New York', 0)

**Insert** **into** City values ('Sydney', 1)

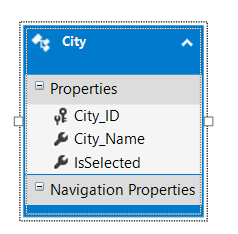
**Insert** **into** City values ('Mumbai', 0)

**Insert** **into** City values ('Cambridge', 0)

**Insert** **into** City values ('Delhi', 0)

**Insert** **into** City values ('Hyderabad', 1)

**Let’s add the ADO.NET data model to retrieve data from the database**



**Creating Controller:**

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

**namespace** *HTML\_HELPER.Controllers*

**{**

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

**{**

**[**HttpGet**]**

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

**{**

CityDBContext dbContext = new CityDBContext**()**;

**return** View**(**dbContext.Cities.ToList**())**;

**}**

**[**HttpPost**]**

**public** string Index**(**IEnumerable**<**City**>** cities**)**

**{**

**if** **(**cities.Count**(**x =**>** x.IsSelected**)** == 0**)**

**{**

**return** "You have not selected any City";

**}**

**else**

**{**

StringBuilder sb = new StringBuilder**()**;

sb.Append**(**"You selected - "**)**;

**foreach** **(**City city in cities**)**

**{**

**if** **(**city.IsSelected**)**

**{**

sb.Append**(**city.City\_Name + ", "**)**;

**}**

**}**

sb.Remove**(**sb.ToString**()**.LastIndexOf**(**","**)**, 1**)**;

**return** sb.ToString**()**;

**}**

**}**

**}**

**}**

**Note:**

1. **HiddenFor**: Hidden for maintains the ID.
2. **DisplayFor**: DisplayFor displays the checkbox name (text).
3. **CheckboxFor**: CheckboxFor displays the checkbox.

Create the Index View and then copy and paste the following code in it

@model List**<HTML**\_HELPER.Models.City**>**

@{

ViewBag.Title = "Index";

}

@using (Html.BeginForm())

{

for (var i = 0; i < Model.Count(); i++)

{

**<table>**

**<tr>**

**<td>**

@Html.HiddenFor(it => it[i].City\_ID)

@Html.HiddenFor(it => it[i].City\_Name)

@Html.DisplayFor(it => it[i].City\_Name)

**</td>**

**<td>**

@Html.CheckBoxFor(it => it[i].IsSelected, new { Style = "vertical-align:3px}" })

**</td>**

**</tr>**

**</table>**

}

**<input** id="Submit1" type="submit" value="submit" **/>**

}

That’s it. We are done with our implementation. Now run the application and test the requirement and you will see it is working as expected.

**ListBox HTML Helper in ASP.NET MVC**

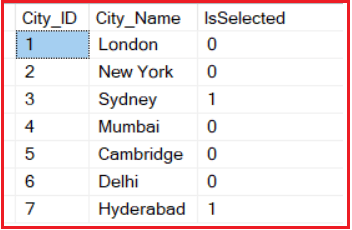
**Creating ListBox using ListBox HTML Helper in ASP.NET MVC**

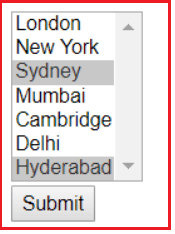
**What is a ListBox?**

A ListBox is a graphical control element that allows the user to select one or more items from the list box. The user clicks inside the box on an item to select it, sometimes in combination with the **Shift or Ctrl** in order to make multiple selections. Clicking an item that has already been selected, unselects it.

**Let us understand ListBox HTML Helper in MVC with an example.**

To understand the ListBox HTML Helper in the ASP.NET MVC application, we are going to use the following **City** Table.

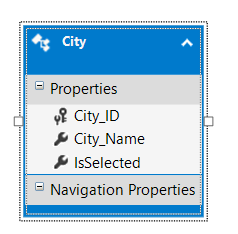


We need to generate the following list box. Notice that for each city in the **City** table, there is an entry in the list box as shown in the below image.   


**Business Requirements:**

We need to allows the user to select one or more cities from the ListBox. Once the user selects the cities and clicks on the **Submit** button,  then we need to display the CityIds of the selected cities separated by the comma. If the user doesn’t select any city and click on the **Submit** button, then a message of **No cities are selected**should be displayed.

For this demo, we are going to use the following **“City”** entity that we created using the **ADO.NET entity framework** in our previous article.



**Creating a View Model:**

First, we need to create a **View Model**. In ASP.NET MVC, the view models are nothing but a mechanism to shuttle data between the controller and the view. To create the View Model, right-click on the **Models** folder, and then add a new class file with the name **CitiesViewModel**. Once you create the **CitiesViewModel** then copy and paste the following code into it. This is the class that is going to be the Model for our view.

**namespace** *HTML\_HELPER.Models*

**{**

**public** **class** CitiesViewModel

**{**

**public** IEnumerable**<**string**>** SelectedCities **{** **get**; **set**; **}**

**public** IEnumerable**<**SelectListItem**>** Cities **{** **get**; **set**; **}**

**}**

**}**

Modify the Home Controller as shown below

**namespace** *HTML\_HELPER.Controllers*

**{**

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

**{**

**[**HttpGet**]**

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

**{**

CityDBContext dbContext = new CityDBContext**()**;

List**<**SelectListItem**>** citiesSelectListItems = new List**<**SelectListItem**>()**;

**foreach** **(**City city in dbContext.Cities.ToList**())**

**{**

SelectListItem selectList = new SelectListItem**()**

**{**

Text = city.City\_Name,

Value = city.City\_ID.ToString**()**,

Selected = city.IsSelected

**}**;

citiesSelectListItems.Add**(**selectList**)**;

**}**

CitiesViewModel citiesViewModel = new CitiesViewModel**()**

**{**

Cities = citiesSelectListItems

**}**;

**return** View**(**citiesViewModel**)**;

**}**

**[**HttpPost**]**

**public** string Index**(**IEnumerable**<**string**>** selectedCities**)**

**{**

**if** **(**selectedCities == **null)**

**{**

**return** "No cities selected";

**}**

**else**

**{**

StringBuilder sb = new StringBuilder**()**;

sb.Append**(**"You selected - " + string.Join**(**",", selectedCities**))**;

**return** sb.ToString**()**;

**}**

**}**

**}**

**}**

**Copy and paste the following code in the “Index.cshtml” view**

@model HTML\_HELPER.Models.CitiesViewModel

@{

ViewBag.Title = "Index";

}

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

**<h2>**Index**</h2>**

@using (Html.BeginForm())

{

@Html.ListBoxFor(m => m.SelectedCities, Model.Cities, new { size = 7 })

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

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

}

**</div>**

**Note:** in order to select multiple items from the list box, you need to hold down the CTRL Key. Run the application and see everything is working as expected.

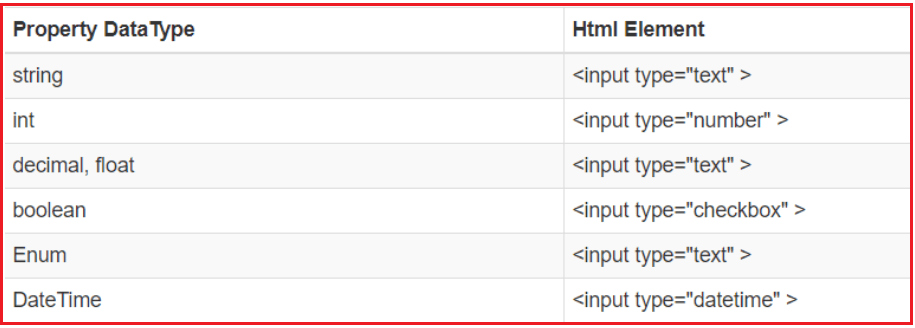
**Editor HTML Helper in ASP.NET MVC**

**Editor HTML Helper in ASP.NET MVC Application**

The ASP.NET MVC Framework provides Editor() HTML Helper method for simple type view and EditorFor() HTML Helper method for strongly type view to generate HTML elements based on the data type of the model object’s property.

**Data Types and Its Equivalent HTML elements:**

The following diagram lists the HTML element that is created for each data type by Editor() or EditorFor() method.



**How to use Editor HTML Helper in ASP.NET MVC Application?**

Let us understand how to use Editor HTML Helper in ASP.NET MVC Application with one example. To do so, first, create an empty ASP.NET MVC application. Then create the following models within the **Models** folder.

**namespace** *HTML\_HELPER.Models*

**{**

**public** **class** Employee

**{**

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

**[**Display**(**Name = "Name"**)]**

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

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

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

**public** **bool** isNewlyEnrolled **{** **get**; **set**; **}**

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

**}**

**}**

**Editor HTML Helper in ASP.NET MVC:**

The Editor HTML Helper method requires a string expression as a parameter to specify the property name. This Editor() extension method creates the input HTML element based on the data type of the specified property. The signature of the Editor() HTML Helper method is given below.

**MvcHtmlString Editor(string propertyname)**

**Creating MVC 5 Controller:**

Create a controller with the name as **EmployeeController** and then copy and paste the following code into it.

**namespace** *HTML\_HELPER.Controllers*

**{**

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

**{**

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

**{**

Employee emp = new Employee**()**

**{**

EmployeeId = 101,

EmployeeName = "Pranaya",

Gender = "Male",

Age = 30,

isNewlyEnrolled = **true**,

DoB =Convert.ToDateTime**(**"02-02-1988"**)**

**}**;

**return** View**(**emp**)**;

**}**

**}**

**}**

**Creating Views:**

Let’s create the index view and then copy and paste the below code into it.

@using HTML\_HELPER.Models

@model Employee

**<table>**

**<tr>**

**<td>**EmployeeId**</td>**

**<td>**@Html.Editor("EmployeeId")**</td>**

**</tr>**

**<tr>**

**<td>**EmployeeName**</td>**

**<td>**@Html.Editor("EmployeeName")**</td>**

**</tr>**

**<tr>**

**<td>**Gender**</td>**

**<td>**@Html.Editor("Gender")**</td>**

**</tr>**

**<tr>**

**<td>**Age**</td>**

**<td>**@Html.Editor("Age")**</td>**

**</tr>**

**<tr>**

**<td>**isNewlyEnrolled**</td>**

**<td>**@Html.Editor("isNewlyEnrolled")**</td>**

**</tr>**

**<tr>**

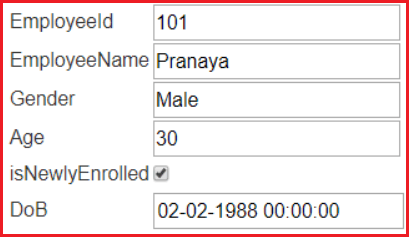
**<td>**DoB**</td>**

**<td>**@Html.Editor("DoB")**</td>**

**</tr>**

**</table>**

**Now, run the application, it will give you the following output.**



In the above example, we have specified the property names of the Employee model. So, the **Editor HTML Helper** method creates the appropriate input elements based on the datatype of Employee model properties as shown in the above image.

**EditorFor HTML Helper in ASP.NET MVC:**

The EditorFor HTML Helper method is a strongly typed helper method. As it is a strongly typed method we need to use a lambda expression to specify the name. The signature of the **EditorFor() HTML Helper**method is given below:

**MvcHtmlString EditorFor(<Expression<Func<TModel, TValue>> expression)**

**Modify the index view as shown below to use EditorFor extension method**

@using HTML\_HELPER.Models

@model Employee

**<br/>**

**<table>**

**<tr>**

**<td>**EmployeeId**</td>**

**<td>**@Html.EditorFor(emp => emp.EmployeeId)**</td>**

**</tr>**

**<tr>**

**<td>**EmployeeName**</td>**

**<td>**@Html.EditorFor(emp => emp.EmployeeName)**</td>**

**</tr>**

**<tr>**

**<td>**Gender**</td>**

**<td>**@Html.EditorFor(emp => emp.Gender)**</td>**

**</tr>**

**<tr>**

**<td>**Age**</td>**

**<td>**@Html.EditorFor(emp => emp.Age)**</td>**

**</tr>**

**<tr>**

**<td>**isNewlyEnrolled**</td>**

**<td>**@Html.EditorFor(emp => emp.isNewlyEnrolled)**</td>**

**</tr>**

**<tr>**

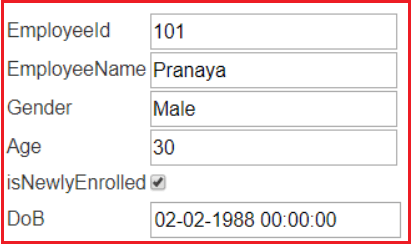
**<td>**DoB**</td>**

**<td>**@Html.EditorFor(emp => emp.DoB)**</td>**

**</tr>**

**</table>**

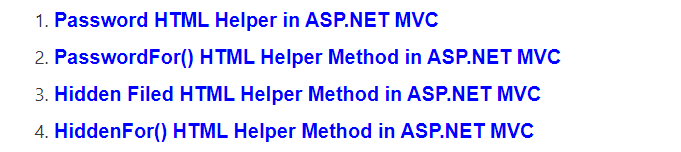
**The output is as shown below.**



In the above example, we specified the property name using the lambda expression. There is no difference in the result whether you use the **Editor()** or the **EditorFor()** extension method.

**Password Field and Hidden Field HTML Helper in ASP.NET MVC**

**Password Field and Hidden Field HTML Helper in ASP.NET MVC**



**Password HTML Helper in ASP.NET MVC:**

The HtmlHelper class provides two extension methods which we can use to generate a password field of **<input type=”password”>** in an MVC view. These two extension methods are **Password()** and **PasswordFor()**.

We are going to use the following Login model to understand the **Password()**and **PasswordFor()** HTML Helper method.

**Create LoginModel within in the Models folder as shown below**

**namespace** *HTML\_HELPER.Models*

**{**

**public** **class** LoginModel

**{**

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

**[**Display**(**Name = "Name"**)]**

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

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

**}**

**}**

**Create LoginController within Controllers Folder:**

Copy and Paste the following code in LoginController

**namespace** *HTML\_HELPER.Controllers*

**{**

**public** **class** LoginController : Controller

**{**

**public** ActionResult Login**()**

**{**

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

**}**

**}**

**}**

**The Password() HTML Helper method in MVC:**

The Html.Password() HTML Helper method in MVC is used to generates an input password element with a specified name, value, and HTML attributes. The signature of Password() Helper Method is shown as below

**MvcHtmlString Html.Password(string name, object value, object htmlAttributes)**

Change the **RouteConfig** File, set the Login Action method of Login Controller as default Route.

**namespace** *HTML\_HELPER*

**{**

**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 = "Login", action = "Login", id = UrlParameter.Optional **}**

**)**;

**}**

**}**

**}**

**Create Login View and Copy and Paste the Following code**

**@model HTML\_HELPER.Models.LoginModel**  
**@Html.Password(“LoginPassword”)**

Run the application and inspect the HTML for the password field text box. It will generate the following HTML for Password field.  
**<input id=”LoginPassword” name=”LoginPassword” type=”password”>**

**The PasswordFor() HTML Helper Method in ASP.NET MVC:**

The PasswordFor() HTML helper method is the strongly typed extension method. This method is used to generate an element of type <input type=”password”>. The name should be specified using a lambda expression.

The PasswordFor() HTML Helper method binds a specified model object property to the password element. As a result, it automatically sets the value of the model property to the password field. The PasswordFor() HTML Helper Method Signature is shown below:

**MvcHtmlString Html.PasswordFor(Expression<Func<dynamic, TProperty>> expression, object htmlAttributes)**

**Modify the Login View as shown below.**

**@model HTML\_HELPER.Models.LoginModel**  
**@Html.PasswordFor(m => m.LoginPassword)**

It will generate the following HTML Code.  
**<input id=”LoginPassword” name=”LoginPassword” type=”password”>**

In the above example, the first parameter of the PasswordFor() helper method is a lambda expression that specifies the model property to be bind with the password field textbox.

**The Hidden Filed HTML Helper Method in ASP.NET MVC:**

The hidden field HTML Helper method is used when we need to store the hidden values on a webpage. We need this when we don’t want to show the values to the end-users but we need these values to update the data when the form is submitted to the server. The HtmlHelper class in ASP.NET MVC provides two extension methods to generate a hidden field of type **(<input type=”hidden”>)** in an MVC razor view. The two methods are Hidden() and HiddenFor().

In this demo, we are going to use the following Student model to understand the Hidden() and HiddenFor() HTML Helper method. So, create the following Student Model within the Models folder of your application.

**namespace** *HTML\_HELPER.Models*

**{**

**public** **class** LoginModel

**{**

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

**[**Display**(**Name = "Name"**)]**

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

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

**}**

**}**

**The Hidden() HTML Helper method in MVC:**

The Hidden() HTML Helper method is used to generate an input hidden field element with a specified name, value, and HTML attributes. the Signature of the Hidden HTML Helper Method is shown below:

**MvcHtmlString Html.Hidden(string name, object value, object htmlAttributes)**

The following example creates a hidden field for LoginId property of the Login model. It binds the LoginId with the hidden field so that it can assign the value of LoginId to the hidden field and visa-versa is also true.

**Html.Hidden() in Razor View**

**@model HTML\_HELPER.Models.LoginModel**  
**@Html.Hidden(“LoginId”,1)**

**Html Result:**  
**<input data-val=”true” data-val-number=”The field LoginId must be a number.”**  
**data-val-required=”The LoginId field is required.” id=”LoginId”**  
**name=”LoginId” type=”hidden” value=”1″>**

**HiddenFor() HTML Helper Method in ASP.NET MVC:**

The HiddenFor() HTML Helper method is the strongly typed extension method which is used to generates an input element hidden for the model property which is specified by using a lambda expression. This HTML Helper method binds a specified model object property to **<input type=”hidden”>** element. So that it automatically sets the value of the model property to the hidden field and the visa-versa is also true. The **HiddenFor() HTML Helper Method Signature** is given below.

**MvcHtmlString Html.HiddenFor(Expression<Func<dynamic, TProperty>> expression)**

**HiddenFor() in Razor View**

**@model HTML\_HELPER.Models.LoginModel**  
**@Html.HiddenFor(m => m.LoginId)**

**Html Result:**  
**<input data-val=”true” data-val-number=”The field LoginId must be a number.”**  
**data-val-required=”The LoginId field is required.” id=”LoginId”**  
**name=”LoginId” type=”hidden” value=”1″>**

In the above example, the first parameter in **HiddenFor()** method is the lambda expression which specifies the model property to be bind with the hidden field. We have specified the LoginId property of the model object in our example.

**Templated Helpers in ASP.NET MVC**

**Templated Helpers in ASP.NET MVC Application**

As an ASP.NET MVC developer, we generally used to use the HTML Helper methods such as **LabelFor() and TextBoxFor()** to display the model properties on a view. This approach works fine in many situations, but it proves to be inadequate when we want to customize the data i.e. how the data should be presented to the end-user for the purpose of displaying and editing. If this is your requirement then you need to use the display template and editor template helpers. At the end of this article, you will understand the following pointers.

1. **Why do we need Templated Helpers in ASP.NET MVC?**
2. **Types of Templated Helpers in ASP.NET MVC**
3. **Types of Display and Editor Templated Helpers in ASP.NET MVC.**
4. **Examples to understand Display and Editor Templated Helpers**

**Why do we need Templated Helpers in ASP.NET MVC?**

When we use an HTML helper method, such as LabelFor() or TextBoxFor(), then it displays the model property in a fixed manner. For example, the LabelFor() extension method renders the model property name within a <label> tag. Similarly, the TextBoxFor() HTML Helper method renders a textbox in which the model property value is shown for editing.

But in some scenarios, we want more control over the data which is going to be present to the end-user either for displaying or editing purposes. For example, we have a model property that stores the currency value. Our requirement is that when displaying this currency property value to the end-user, along with the currency value we also want to show the currency symbol such as $. If this is your requirement then it is not possible with the HTML Helper methods such as LabelFor() extension method. Similarly, if you have a DateTime property and you want to display the DateTime in a specific format then you can’t do this customization using the HTML Helper methods.

To overcome the above problem means to customize the model data, the ASP.NET MVC Framework comes with templated helpers that can be used in such scenarios.

**Types of Templated Helpers in ASP.NET MVC**

The Templated helpers are introduced as part of MVC 2. The built-in template helpers in ASP.NET MVC are classified into 2 types are as follows.

1. **Display Templates**
2. **Editor Templates**

If you want then you can also create your own custom templated helpers which we will discuss in our next article.

**Types of Display Templated Helpers in ASP.NET MVC**

There are three types of Display Templated Helpers available in ASP.NET MVC. They are as follows.

1. **@Html.Display(“EmployeeData”)** – Used with a view that is not strongly typed. For example, if we have data stored in a ViewData or ViewBag, then we can use this templated helper using the key that was used to store data in ViewData or ViewBag.
2. **@Html.DisplayFor(model => model)** – Used with strongly typed views. If our model has properties that return complex objects, then this templated helper is very useful.
3. **@Html.DisplayForModel()** – Used with strongly typed views. It will work through each property of the model to display the object.

**Types of Edit Templated Helpers in ASP.NET MVC**

There are also 3 Edit Templated Helpers available in ASP.NET MVC. They are as follows

1. **@Html.Editor(“EmployeeData”)**
2. **@Html.EditorFor(model => model)**
3. **@Html.EditorForModel()**

When we are going to create views, we can use the **DisplayFor(), DisplayForModel(), EditorFor(), and EditorForModel()** helpers even if we don’t intend to have a customized user interface. Later, if we decide to have a customized interface, all you need to do is define the display and editor templates that we will discuss in a later article.

**Understanding Templated Helpers in ASP.NET MVC Application:**

Let us understand the Templated Helpers in ASP.NET MVC Application with an example. First, create an empty ASP.NET MVC application with the name **TemplateHelpersMVC**. Then add a class with the name **Employee** within the Models folder as shown below.

**namespace** *TemplateHelpersMVC.Models*

**{**

**public** **class** Employee

**{**

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

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

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

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

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

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

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

**}**

**}**

**Creating Controller:**

Next, add a controller with the name as **EmployeeController** within the Controller and then copy and paste the below codes in it.

**namespace** *TemplateHelpersMVC.Controllers*

**{**

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

**{**

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

**{**

//Here we are hardcoded the Employee Details

//In Realtime you will get the data from any data source

Employee employee = new Employee**()**

**{**

Id = 1,

FullName = "Pranaya Rout",

Gender = "Male",

HireDate = Convert.ToDateTime**(**"2017-01-02 17:53:46.833"**)**,

EmailAddress = "info@dotnettutorials.com",

Salary = 500000,

PersonalWebSite = "https://dotnettutorials.net/"

**}**;

ViewData**[**"EmployeeData"**]** = employee;

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

**}**

**}**

**}**

Notice that in the above code, we have added one action method with the name **Details** and then we stored the employee object in the ViewData using the **“EmployeeData”** key.

**Creating the Details view:**

Create the Details view and then copy and paste the below code in it.

@{

ViewBag.Title = "Details";

}

**<fieldset>**

**<legend>**Employee Details**</legend>**

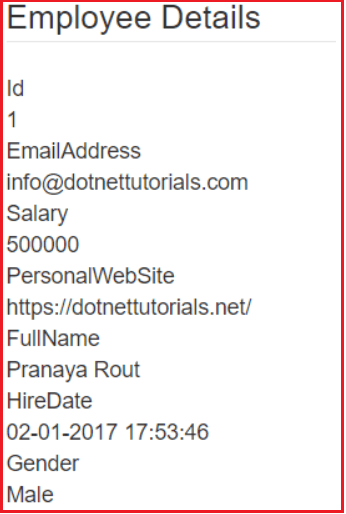
@Html.Display("EmployeeData")

**</fieldset>**

As you can see in the above code, we are using **@Html.Display(“EmployeeData”)** templated helper method to render the data. At the moment **“Details.cshtml”** view does not have a Model associated with it. So it is not a strongly-typed view. Now run the application and navigate to the following URL

**http://localhost:61629/Employee/Details**

It will display the following output as expected.



**Now, change the implementation of the “Details” action method within the Employee controller as shown below.**

**namespace** *TemplateHelpersMVC.Controllers*

**{**

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

**{**

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

**{**

//Here we are hardcoded the Employee Details

//In Realtime you will get the data from any data source

Employee employee = new Employee**()**

**{**

Id = 1,

FullName = "Pranaya Rout",

Gender = "Male",

HireDate = Convert.ToDateTime**(**"2017-01-02 17:53:46.833"**)**,

EmailAddress = "info@dotnettutorials.com",

Salary = 500000,

PersonalWebSite = "https://dotnettutorials.net/"

**}**;

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

**}**

**}**

**}**

You can see in the above code that, instead of storing the “Employee” object in the ViewData, we are passing it to the View.

**Now change the Details.cshtml view as below.**

@model TemplateHelpersMVC.Models.Employee

@{

ViewBag.Title = "Details";

}

**<fieldset>**

**<legend>**Employee Details**</legend>**

@Html.DisplayFor(model => model)

**</fieldset>**

In the above Details view, we have specified **“Employee”** as the model object. That means, here we are working with a strongly typed view, and hence we are using **@Html.DisplayFor (model => model)** templated helper method.

Since none of the properties of the Employee class return a complex object, the ideal choice here would be, to use **@Html.DisplayForModel()** templated helper as shown below.

@model TemplateHelpersMVC.Models.Employee

@{

ViewBag.Title = "Details";

}

**<fieldset>**

**<legend>**Employee Details**</legend>**

@Html.DisplayForModel()

**</fieldset>**

Whether we use DisplayFor() or DisplayForModel() templates helper, in both cases, the output will be the same. So Run the application and see everything is working as expected.

**Editor Templated Helpers in ASP.NET MVC:**

The way we work with Display Template Helpers, in the same way, we work with **Editor Template Helper** in ASP.NET MVC. So let’s understand this concept by adding the Edit Action method within the Employee Controller as shown below.

**namespace** *TemplateHelpersMVC.Controllers*

**{**

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

**{**

**public** ActionResult Edit**()**

**{**

//Here we are hardcoded the Employee Details

//In Realtime you will get the data from any data source

Employee employee = new Employee**()**

**{**

Id = 1,

FullName = "Pranaya Rout",

Gender = "Male",

HireDate = Convert.ToDateTime**(**"2017-01-02 17:53:46.833"**)**,

EmailAddress = "info@dotnettutorials.com",

Salary = 500000,

PersonalWebSite = "https://dotnettutorials.net/"

**}**;

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

**}**

**}**

**}**

**Now we need to add the Edit.cshtml view as shown below**

@model TemplateHelpersMVC.Models.Employee

@{

ViewBag.Title = "Edit";

}

**<h2>**Edit**</h2>**

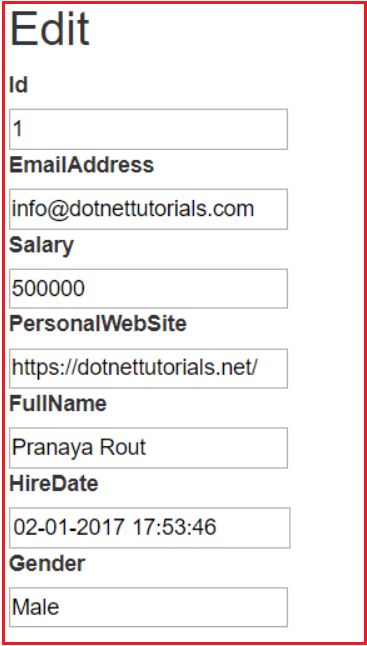
@using (@Html.BeginForm())

{

@Html.EditorForModel()

}

Run the application and navigates to the below URL and see everything is working as expected.  
**http://localhost:61629/Employee/Edit**  
it will display the following:

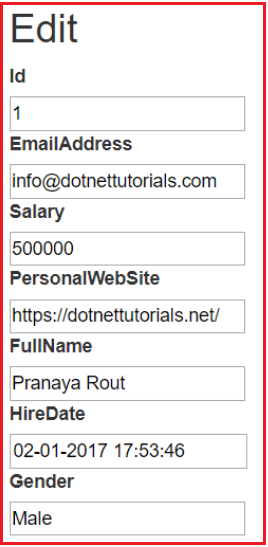


Here I just show you how to use **EditorForModel**, the rest two approaches you can test by yourself.

**Customizing Templated Helpers in ASP.NET MVC**

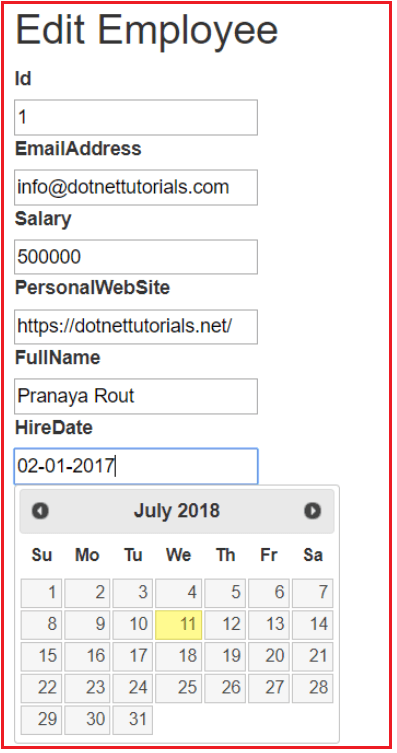
**Customizing Templated Helpers in ASP.NET MVC Application**

At the moment, when we navigate to **http://localhost:61629/Employee/Edit** it will display the following page.



Notice that in the above image, for the **HireDate**, the users have to type the date in the date text box. As we all know the date-time has different formats. For example, it may be **MM/DD/YYYY** or **DD/MM/YY,**etc. So, different users may enter the date value in a different manner. Also, from the user’s point of view, it is always better to display a **DateTime** picker from which the user can easily select the date.

The **built-in DateTime editor** template used by ASP.NET MVC Framework simply displays a textbox for editing the Dates. So, let’s customize the DateTime editor template, to use the jQuery calendar. We want the output as shown in the below image.



**The following convention is used by the MVC Framework to find the customized templates**

1. The customized display templates must be stored in a sub-folder with the name **DisplayTemplates**. Similarly, the customized Editor templates must be stored in a sub-folder with the name **EditorTemplates**.
2. These sub-folders can be created within the **“Shared”** folder or within a specific view folder. If these folders are present within the Shared folder, then the templates are available for all the views. If they are present within a specific views folder, then, they are only available the views which are present in that specific views folder.
3. The most important point that you need to keep in mind is the **name of the template** must match the **name of the data type**. For example, as we are going to customizing the DateTime template, the name of the template must be **DateTime.cshtml**.

**Adding a Custom DateTime Editor template**

**Step1:** If the “**Shared**” folder does not already exist in your project within the views folder, then right-click on the Views folder and add it.  
**Step2:** Right-click on the “**Shared**” folder, and “**EditorTemplates**” folder.  
**Step3:** Right-click on “**EditorTemplates**” folder and add a partial view with the name **DateTime**  
**Step4:** Copy and paste the following code in **DateTime.cshtml**partial view

@model DateTime?

@Html.TextBox("", (Model.HasValue ? Model.Value.ToString("dd/MM/yyyy") : string.Empty), new

{

@class = "date"

})

**Step5: Copy and paste the following code within the Edit.cshtml view**

@model TemplateHelpersMVC.Models.Employee

@{

ViewBag.Title = "Edit";

Layout = null;

}

**<script** src="~/Scripts/jquery-1.12.4.min.js"**></script>**

**<script** src="~/Scripts/jquery-ui-1.12.1.min.js"**></script>**

**<link** href="~/Content/bootstrap.min.css" rel="stylesheet" **/>**

**<link** href="~/Content/Site.css" rel="stylesheet" **/>**

**<link** href="~/Content/themes/base/jquery-ui.min.css" rel="stylesheet" **/>**

**<link** href="~/Content/themes/base/datepicker.css" rel="stylesheet" **/>**

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

$(function()

{

$("input:text.date").datepicker(

{

dateFormat: "dd/mm/yy"

});

});

**</script>**

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

**<div** class="row"**>**

**<div** class="col-md-offset-3 col-md-6"**>**

@using (@Html.BeginForm())

{

**<h2>**Edit Employee**</h2>**

@Html.EditorForModel()

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

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

}

**</div>**

**</div>**

**</div>**

Now run the application and you will see a jquery date picker is used to display the DateTime as expected. Please download the latest version of Jquery and Jquery UI from the NuGet in order to work it properly. If there is some version mismatch then it will not work.

**Custom HTML Helpers in ASP.NET MVC**

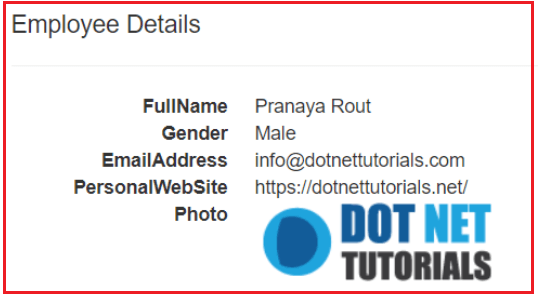
**Creating Custom HTML Helpers in ASP.NET MVC Application**

1. **How we can display Images in an ASP.NET MVC application?**
2. **How to create Custom HTML Helpers in MVC to display images?**

As we already discussed the HTML helper is a method that returns an HTML string. Then this HTML string is rendered in a view. ASP.NET MVC provides many built-in HTML helper methods that we can directly use in a view. The MVC framework also provides the facility to create **Custom HTML Helpers in ASP.NET MVC** application. Once you create your custom HTML helper method then you can reuse it many times.

**Let’s understand Custom HTML Helpers in MVC with an example.**

In this demo, we are going to display the employee details along with the Employeephoto as shown in the below image.



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

First, create an empty ASP.NET MVC application with the name **CustomHTMLHelper**. Then create one model class with the name **Employee** within the **Models** Folder and then copy and paste the following code in it.

**namespace** *CustomHTMLHelper.Models*

**{**

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

**{**

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

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

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

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

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

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

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

**}**

**}**

**Then add a folder with the Name Photos to the project.**

To do this, Right-click on the Project and select Add Folder and then rename the folder as “**Photos**“. Then download and add the following image within the **Photos** Folder. Rename the image name as **MyPhoto.png**.

**Creating Controller:**

Now add a controller with the name **EmployeeController** within the **Controllers** folder and then copy and paste the below codes.

**namespace** *CustomHTMLHelper.Controllers*

**{**

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

**{**

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

**{**

//Here we are hardcoded the Employee Details

//In Realtime you will get the data from any data source

Employee employee = new Employee**()**

**{**

Id = 1,

FullName = "Pranaya Rout",

Gender = "Male",

EmailAddress = "info@dotnettutorials.com",

PersonalWebSite = "https://dotnettutorials.net/",

Photo = "~/Photos/MyPhoto.png",

AlternateText = "Pranaya Rout Photo not available"

**}**;

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

**}**

**}**

**}**

Adding View:

Add the Details view and then copy and paste the following codes in it.

@model CustomHTMLHelper.Models.Employee

@{

ViewBag.Title = "Details";

}

**<div>**

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

**<hr** **/>**

**<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.EmailAddress)

**</dt>**

**<dd>**

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

**</dd>**

**<dt>**

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

**</dt>**

**<dd>**

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

**</dd>**

**<dt>**

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

**</dt>**

**<dd>**

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

**</dd>**

**<dt>**

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

**</dt>**

**<dd>**

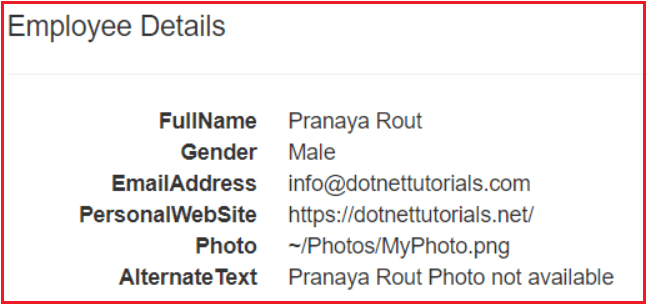
@Html.DisplayFor(model => model.AlternateText)

**</dd>**

**</dl>**

**</div>**

Now Run the application and navigates to the URL **http://localhost:61629/Employee/Details**It will produce the following output.



Notice instead of rendering the photo, the PhotoPath and AlternateText property values are displayed.

**How to display an image in ASP.NET MVC Application?**

**<dt>**

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

**</dt>**

**<dd>**

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

**</dd>**

**<dt>**

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

**</dt>**

**<dd>**

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

**</dd>**

**Replace the above code with the following code.**

**<dt>**

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

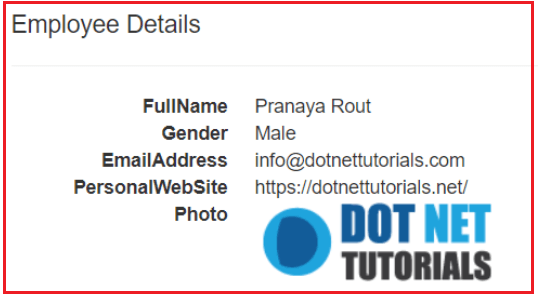
**</dt>**

**<dd>**

**<img** src="@Url.Content(@Model.Photo)" alt="@Model.AlternateText" **/>**

**</dd>**

Notice that, now we are using the **Url.Content()** HTML helper method. This method resolves a URL for a resource when we pass it the relative path. Now, run the application, and notice that the image is displayed as expected as shown in the below image.



We use the below code to render Image in ASP.NET MVC application. We are building the image tag, by passing the values for “src” and “alt” attributes.

**<img src=”@Url.Content(@Model.Photo)” alt=”@Model.AlternateText” />**

Though the above code is not very complex, it still makes sense to move this logic into its own helper method. We don’t want any complicated logic in our views. Views should be as simple as possible. Don’t you think, it would be very nice, if we can render the image, using Image() HTML helper method as shown below.

**@Html.Image(Model.Photo, Model.AlternateText)**

But, ASP.NET MVC does not provide any built-in Image() HTML helper. So, let’s build our own custom image HTML helper method. Let’s take a step back and understand HTML helper methods. The HTML helper method simply returns a string. To generate a textbox, we use the following code in our view.

**@Html.TextBox(“TextBox Name”)**

So, here TextBox() is an extension method defined in HtmlHelper class. In the above code, Html is the property of the View, which returns an instance of the HtmlHelper class.

**Creating Image() extension method, to HtmlHelper class.**

Right-click on the project and add the “**CustomHelpers**” folder.  Then Right-click on the “**CustomHelpers**” folder and add the “**CustomHelpers.cs**” class file. Copy and paste the following code in it. The code is commented and self-explanatory. **TagBuilder**class is in **System.Web.Mvc** namespace.

**using** *System.Web;*

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

**namespace** *CustomHTMLHelper.CustomHelpers*

**{**

**public** **static** **class** CustomHelpers

**{**

**public** **static** IHtmlString Image**(**this HtmlHelper helper, string src, string alt**)**

**{**

// Build <img> tag. The parameter name must be img

TagBuilder tb = new TagBuilder**(**"img"**)**;

// Add "src" attribute

tb.Attributes.Add**(**"src", VirtualPathUtility.ToAbsolute**(**src**))**;

// Add "alt" attribute

tb.Attributes.Add**(**"alt", alt**)**;

// return MvcHtmlString. This class implements IHtmlString

// interface. IHtmlStrings will not be html encoded.

**return** new MvcHtmlString**(**tb.ToString**(**TagRenderMode.SelfClosing**))**;

**}**

**}**

**}**

To use the custom Image() HTML helper in Details.cshtml view, please include the following using statement in Details.cshtml

**@using CustomHTMLHelper.CustomHelpers;**

As we intend to use this **Image() HTML helper**, in all our views, let’s include “**CustomHTMLHelper.CustomHelpers**” namespace in **web.config** that is present in the **Views** Folder. This eliminates the need to include the namespace, in each and every view.

**<system.web.webPages.razor>**

**<pages** pageBaseType="System.Web.Mvc.WebViewPage"**>**

**<namespaces>**

**<add** namespace="System.Web.Mvc" **/>**

**<add** namespace="System.Web.Mvc.Ajax" **/>**

**<add** namespace="System.Web.Mvc.Html" **/>**

**<add** namespace="System.Web.Routing" **/>**

**<add** namespace="TemplateHelpersMVC" **/>**

**<add** namespace="CustomHTMLHelper.CustomHelpers" **/>**

**</namespaces>**

**</pages>**

**<host** ......**/>**

**</system.web.webPages.razor>**

Now use the following code to display an Image

**@Html.Image(Model.Photo, Model.AlternateText)**

If you intend to use the Image() custom HTML helper, only with a set of views, then, include a web.config file in the specific views folder, and then specify the namespace in it.