Tag Helpers in ASP.NET Core MVC

In this article, I will discuss **Tag Helpers in ASP.NET Core MVC** Applications with Examples. Tag helpers are one of the new features introduced in ASP.NET Core MVC. Please read our previous article discussing [**How to Create Custom HTML Helpers in ASP.NET Core MVC**](https://dotnettutorials.net/lesson/custom-image-html-helper-in-asp-net-core-mvc/) Applications. As part of this article, we will discuss the following pointers.

Advertisements

1. **What are Tag Helpers in ASP.NET Core MVC?**
2. **Types of Tag Helpers in ASP.NET Core MVC**
3. **Built-in Tag Helpers Type in ASP.NET Core MVC**
4. **How to Use Tag Helpers in ASP.NET Core MVC?**
5. **Example to Understand Built-in Tag Helpers in ASP.NET Core MVC**
6. **Generating Links using Tag Helpers in ASP.NET Core MVC Application**
7. **Why Tag Helpers over HTML Helpers in ASP.NET Core MVC?**

**What are Tag Helpers in ASP.NET Core MVC?**

Tag Helpers in ASP.NET Core MVC allow server-side code to participate in creating and rendering HTML elements in Razor views. They provide a way to add server-side functionality to HTML elements in a more readable way than traditional HTML helpers.

Tag Helpers enhance HTML elements with attributes that generate dynamic HTML and allow developers to create cleaner and more maintainable markup.

The following are some of the Key Characteristics of Tag Helpers in ASP.NET Core MVC:

* **HTML-Like Syntax:** Tag Helpers use HTML-like syntax, which allows HTML developers to create and edit Razor views easily.
* **Model Binding:** Tag Helpers supports model binding.
* **Reusable:** They encapsulate the rendering logic and can be reused across different views, reducing code duplication.
* **Extensible:** Developers can create custom Tag Helpers to meet specific needs beyond what’s provided by built-in Tag Helpers.
* **Easy to Understand:** Tag Helpers use HTML-like syntax, making it easier for developers to understand and work with than HTML helpers, which rely on method calls.
* **IntelliSense Support:** Provides strong IntelliSense support within Visual Studio, making development faster and reducing errors.

**Types of Tag Helpers in ASP.NET Core MVC:**

Tag Helpers in ASP.NET Core MVC are classified into two Types. They are as follows:

* **Built-In Tag Helpers:** The ASP.NET Core framework provides these. They include functionality for common tasks such as form handling, link generation, and showing validation messages.
* **Custom Tag Helpers:** Developers can create their own Tag Helpers to encapsulate specific functionality or rendering logic required by their application.

**Note:** In this article, I will discuss **Built-in Tag Helpers**, and in our upcoming articles, I will discuss [**How to Create Custom Tag Helpers**](https://dotnettutorials.net/lesson/creating-custom-tag-helper-in-as-net-core-mvc/) in ASP.NET Core MVC Application.

**Built-in Tag Helpers Type in ASP.NET Core MVC:**

The ASP.NET Core MVC Framework provides several built-in HTML Tag Helpers that generate the most common HTML elements and scenarios.

The following are some of the built-in HTML Tag Helpers available in ASP.NET Core MVC:

* **Form Tag Helpers:** <form> Tag Helper generates HTML forms. They help create forms that post data to actions in your controllers.

For example: <form asp-controller=”Account” asp-action=”Login” method=”post”> <!– form fields –> </form>

* **Input Tag Helpers:** <input>, <label>, <textarea>, <select>, etc, to facilitate form data management and validation.

For example, <input asp-for=”Name” /> and <input asp-for=”IsAdmin” type=”checkbox” />.

* **Anchor Tag Helper:** <a> Tag Helper for generating links using routing information.

For example, <a asp-controller=”Home” asp-action=”Details” asp-route-id=”1″>Details</a>

* **Image Tag Helper:** <img> Tag Helper for rendering images with dynamic sources.

For example, <img src=”~/images/pic.jpg” alt=”Picture”>

* **Environment Tag Helper:** <environment> Tag Helper for rendering content conditionally based on the current environment, such as Development, Staging, or Production.
* **Cache Tag Helper:** <cache> Tag Helper for output caching. This can help cache portions of your views to improve performance by reducing the need to recompute content on every request. For example, <cache> <!– Content to be cached –> </cache>
* **Partial Tag Helper:** <partial> Tag Helper for rendering partial views. This tag helper renders a partial view within another view.
* **View Component Tag Helper:** <view-component> tag helper renders a view component within a view. View components are similar to partial views but offer more encapsulation and flexibility.

**How to Use Tag Helpers in ASP.NET Core MVC?**

We need to follow the below two Steps to Use Tag Helpers in ASP.NET Core MVC:

**Enable Tag Helpers:** To enable tag helpers, we first need to add the @addTagHelper directive to our Razor view or layout file. You can also add the @addTagHelper directive in the **\_ViewImports.cshtml** file.

**@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers**

**Use Tag Helpers:** Use Tag Helpers directly (HTML-like syntax) in the Razor views by adding them as HTML attributes. They will be processed on the server side and rendered as HTML. For example, use asp-action, asp-controller, or asp-for attributes to generate links or bind data to input fields.

**Example to Understand Built-in Tag Helpers in ASP.NET Core MVC:**

Let us understand how to use built-in tag helpers in an ASP.NET Core MVC application. So, create a new ASP.NET Core Application named **TagHelpersDemo** using the Model-View-Controller Project template. To make the Built-in Tag Helpers available for all views of our application, we need to import **@addTagHelper \*,**

**Microsoft.AspNetCore.Mvc.TagHelpers**directive inside the **\_ViewImports.cshtml** file. If you are creating the ASP.NET Core project using Model-View-Controller, then by default, this directive is included inside the **\_ViewImports.cshtml.**If you open the **\_ViewImports.cshtml**file, you will see the following code:

**@using TagHelpersDemo  
@using TagHelpersDemo.Models  
@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers**

**What is @addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers Directive in ASP.NET Core MVC?**

The **@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers** directive in an ASP.NET Core MVC application enables the built-in Tag Helpers provided by the ASP.NET Core MVC framework to be used across all the Razor views within the application. This directive is typically added to the **\_ViewImports.cshtml** file, a common place for importing directives that should be shared across multiple views.

Here,  
**@addTagHelper:** This directive includes specific Tag Helpers in the Razor view.

**\* (wildcard):** The \* symbol acts as a wildcard, loading all available Tag Helpers from the specified assembly or namespace.

In this case, it loads all Tag Helpers from Microsoft.AspNetCore.Mvc.TagHelpers namespace.

**Microsoft.AspNetCore.Mvc.TagHelpers:** This is the namespace where the built-in ASP.NET Core MVC Tag Helpers are defined. It includes Tag Helpers such as:

* AnchorTagHelper for generating HTML links
* FormTagHelper for form management
* InputTagHelper for binding inputs to model properties
* LabelTagHelper for generating labels
* And many more…

**Generating Links using Tag Helpers in ASP.NET Core MVC Application:**

Let’s understand how to generate links using the Tag helper in ASP.NET Core MVC. First, create a class file named **Student.cs** within the **Models** folder, and then please copy and paste the following code. This Student Model has only five properties: ID, Name, Branch, Section, and Gender.

Advertisements

**namespace** *TagHelpersDemo.Models*

**{**

**public** **class** Student

**{**

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

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

**public** **string**? Branch **{** **get**; **set**; **}**

**public** **string**? Section **{** **get**; **set**; **}**

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

**}**

**}**

**Modifying Home Controller:**

Next, modify the HomeController as shown below. Here, we have created two action methods. To simplify things and keep the focus on Tag helpers, we have hard-coded the required student data within the action method. The Index action method returns a list of students to the view, whereas the Details action method takes the Student ID as a parameter and returns that student information to the view.

**using** *TagHelpersDemo.Models;*

**using** *Microsoft.AspNetCore.Mvc;*

**namespace** *TagHelpersDemo.Controllers*

**{**

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

**{**

**private** List**<**Student**>** listStudents;

**public** HomeController**()**

**{**

//Create a List of Students

//In Real-Time, you will get the data from the database

listStudents = new List**<**Student**>()**

**{**

new Student**()** **{** StudentId = 101, Name = "James", Branch = "CSE", Section = "A", Gender = "Male" **}**,

new Student**()** **{** StudentId = 102, Name = "Smith", Branch = "ETC", Section = "B", Gender = "Male" **}**,

new Student**()** **{** StudentId = 103, Name = "David", Branch = "CSE", Section = "A", Gender = "Male" **}**,

new Student**()** **{** StudentId = 104, Name = "Sara", Branch = "CSE", Section = "A", Gender = "Female" **}**,

new Student**()** **{** StudentId = 105, Name = "Pam", Branch = "ETC", Section = "B", Gender = "Female" **}**

**}**;

**}**

**public** ViewResult Index**()**

**{**

//Pass the Student List to the View to make the view as a Strongly Typed View

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

**}**

**public** ViewResult Details**(int** Id**)**

**{**

//Fetch the Student Details

var studentDetails = listStudents.FirstOrDefault**(**std =**>** std.StudentId == Id**)**;

//Pass the Student model to the View to make the view as a Strongly Typed View

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

**}**

**}**

**}**

**Creating Details View:**

Create a view named **Details.cshtml** within the **Views=>Home** folder, then copy and paste the following code. The model for this view is Student, and this view renders the Student information.

@model TagHelpersDemo.Models.Student

@{

ViewBag.Title = "Student Details";

}

**<div** class="row m-3"**>**

**<div** class="col-sm-8"**>**

**<div** class="card"**>**

**<div** class="card-header text-center"**>**

**<h1>**@Model?.Name**</h1>**

**</div>**

**<div** class="card-body text-center"**>**

**<h4>**Studnet ID : @Model?.StudentId**</h4>**

**<h4>**Branch : @Model?.Branch**</h4>**

**<h4>**Section : @Model?.Section**</h4>**

**<h4>**Gender : @Model?.Gender**</h4>**

**</div>**

**<div** class="card-footer text-center"**>**

**<a** asp-action="Index" asp-controller="Home" class="btn btn-primary"**>**

Back

**</a>**

**</div>**

**</div>**

**</div>**

**</div>**

**Modifying Index View:**

Next, we need to display all the limited Student information within the Index View and a link to the Details view. When we click the link, we need to display the Student Details. For example, we need to generate the following hyperlink. The number 101 is the ID of the student whose details we want to view.

**/home/details/101**

Many different methods are available for generating a link in the ASP.NET Core MVC Application. Let’s discuss all the possible options and why we should use Tag Helper over others.

**Method 1: Using HTML Element:**

In this case, we need to use the anchor tag, and in the href attribute, we need to specify the path of the details action method (i.e., the Controller Name and Action Name) along with the student id like **<a href=”/home/details/@student.StudentId”>View</a>**. So, modify the Index action method of the Home Controller as follows.

@model List**<TagHelpersDemo.Models.Student>**

@{

ViewBag.Title = "Student List";

}

**<div** class="table-responsive"**>**

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

**<thead>**

**<tr>**

**<th>**ID**</th>**

**<th>**Name**</th>**

**<th>**View**</th>**

**</tr>**

**</thead>**

**<tbody>**

@foreach (var student in Model)

{

**<tr>**

**<td>**@student.StudentId**</td>**

**<td>**@student.Name**</td>**

**<td><a** href="/home/details/@student.StudentId" class="btn btn-primary"**>**View**</a></td>**

**</tr>**

}

**</tbody>**

**</table>**

**</div>**

With the above changes, run the application and navigate to the Root or **Home/Index** URL. You should see the following page on the web browser.

Once you click the View button, the Home Controller’s Details action method will execute by passing the corresponding Student ID value. If you click the View button of Student ID 1, you will see the following Details view of the Home Controller showing the Student details.

**Method 2: Using HTML Helpers**

Using HTML Helper methods, we can also generate the link. In this case, we need to use the ActionLink Helper method to generate the link like **@Html.ActionLink(“View”, “Details”, “Home”, new { id = student.StudentId }, new { @class = “btn btn-primary” }).**

Here, the parameter **View** is nothing but the link text; the second parameter **Details** is the Action method name; the third parameter **Home** is the controller’s name. The fourth parameter is object routeValues, which specifies the id parameter value. The fifth parameter is the HTML attributes, and here, we are applying the BootStarp class. So, modify the **Index.cshtml** View of the Home Controller as follows to use the HTML Helper Method to generate the link.

@model List**<TagHelpersDemo.Models.Student>**

@{

ViewBag.Title = "Student List";

}

**<div** class="table-responsive"**>**

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

**<thead>**

**<tr>**

**<th>**ID**</th>**

**<th>**Name**</th>**

**<th>**View**</th>**

**</tr>**

**</thead>**

**<tbody>**

@foreach (var student in Model)

{

**<tr>**

**<td>**@student.StudentId**</td>**

**<td>**@student.Name**</td>**

**<td>**@Html.ActionLink("View", "Details", "Home", new { id = student.StudentId }, new { @class = "btn btn-primary" })**</td>**

**</tr>**

}

**</tbody>**

**</table>**

**</div>**

With the above changes, run the application and see if everything works as expected.

**Method 3: Using Tag Helpers:**

To use Tag Helpers, please modify the Index view of the Home Controller as shown below.

@model List**<TagHelpersDemo.Models.Student>**

@{

ViewBag.Title = "Student List";

}

**<div** class="table-responsive"**>**

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

**<thead>**

**<tr>**

**<th>**ID**</th>**

**<th>**Name**</th>**

**<th>**View**</th>**

**</tr>**

**</thead>**

**<tbody>**

@foreach (var student in Model)

{

**<tr>**

**<td>**@student.StudentId**</td>**

**<td>**@student.Name**</td>**

**<td>**

**<a** asp-action="Details" asp-controller="Home" asp-route-id="@student.StudentId" class="btn btn-primary"**>**

View

**</a>**

**</td>**

**</tr>**

}

**</tbody>**

**</table>**

**</div>**

Now, run the application, and it should work as expected.

**Understanding Anchor Tag Helper in ASP.NET Core MVC:**

The Anchor Tag Helper in ASP.NET Core creates the standard HTML anchor (<a … ></a>) tag by adding attributes such as:

* **asp-controller:** It is used to specify the controller to target based on the routing system. For example, **asp-controller=”Home”** Specifies the controller name (“Home”). If you omitted this, then the controller of the current view is used by default.
* **asp-action:** It is used to specify the Action method to target based on the routing system. For example, asp-action=”Details” Specifies the action method name (“Details”). If you omit this attribute, the action rendering the current view is the default.
* **asp-route-{value}:** It specifies the value required by the action method. For example, asp-route-id=”@student.StudentId” specifies the route parameter (in this case, the id) dynamically from the model, i.e., @student.StudentId.

The value of the rendered anchor element’s “href” attribute is determined by the values of these “asp-” attributes. As the name says, asp-controller specifies the controller’s name, whereas asp-action specifies the action name. Similarly, the asp-route-{value} attribute includes route data in the generated href attribute value. {value} can be replaced with route parameters such as id, name, etc. Let us have a look at the following image for a better understanding.

As you can see in the above image, manually generating the links is much easier than using HTML Helpers or Tag Helpers. So why should we use HTML helpers or Tag Helpers instead of manually generating these links in ASP.NET Core?

**Why should we use Tag Helpers over manually generating these links in ASP.NET Core?**

In ASP.NET Core MVC, Tag Helpers generate links based on the application routing configuration. That means if we change the routing configuration in the future, the links generated by tag helpers will automatically reflect those changes. So, the generated links work as expected without any trouble.

On the other hand, if we manually hard-code the URL paths, we need to change the code wherever we have hard-coded the path in our application when the application routing configuration changes. Let’s understand this with an example. The following is the program class code.

**namespace** *TagHelpersDemo*

**{**

**public** **class** Program

**{**

**public** **static** **void** Main**(string[]** args**)**

**{**

var builder = WebApplication.CreateBuilder**(**args**)**;

// Add services to the container.

builder.Services.AddControllersWithViews**()**;

var app = builder.Build**()**;

// Configure the HTTP request pipeline.

**if** **(**!app.Environment.IsDevelopment**())**

**{**

app.UseExceptionHandler**(**"/Home/Error"**)**;

// The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.

app.UseHsts**()**;

**}**

app.UseHttpsRedirection**()**;

app.UseStaticFiles**()**;

app.UseRouting**()**;

app.UseAuthorization**()**;

app.MapControllerRoute**(**

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}"**)**;

app.Run**()**;

**}**

**}**

**}**

As you can see in the above code, currently, we have the route template pattern as **{controller=Home}/{action=Index}/{id?}**.

**Generating Link Manually:**

We can generate the link manually using the following code by hard-coding the URL paths as **/Home/Details**.

**<a href=”/Home/Details/@student.StudentId”>View</a>**

**Generating Link using Tag Helper:**

In the following code, we generate the link using anchor tag helper.

**<a asp-controller=”Home” asp-action=”Details” asp-route-id=”@student.StudentId”>View</a>**

As you can see with Tag Helper, we have not hardcoded the URL paths. Here, we only specify the controller and action name, route parameters, and their values. When the tag helpers are executed on the server, they automatically look at the route templates and generate the correct URLs.  So, to better understand, modify the Index View as follows:

@model List**<TagHelpersDemo.Models.Student>**

@{

ViewBag.Title = "Student List";

}

**<div** class="table-responsive"**>**

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

**<thead>**

**<tr>**

**<th>**ID**</th>**

**<th>**Name**</th>**

**<th>**Manually**</th>**

**<th>**Tag Helper**</th>**

**</tr>**

**</thead>**

**<tbody>**

@foreach (var student in Model)

{

**<tr>**

**<td>**@student.StudentId**</td>**

**<td>**@student.Name**</td>**

**<td>**

**<a** href="/Home/Details/@student.StudentId" class="btn btn-primary"**>**View**</a>**

**</td>**

**<td>**

**<a** asp-action="Details" asp-controller="Home" asp-route-id="@student.StudentId" class="btn btn-primary"**>**

View

**</a>**

**</td>**

**</tr>**

}

**</tbody>**

**</table>**

**</div>**

Here, both the techniques generate the same URL path, i.e. **(/Home/Details/101),** and it also works with the current route template, i.e. **({controller=Home}/{action=Index}/{id?})**

**Modifying the Routing Configuration:**

Now, let us change the routing template as shown below. Here, we have added the string literal “dotnet” with the Route Template Pattern, i.e., the Pattern is: **dotnet/{controller=Home}/{action=Index}/{id?}**

Run the application and navigate to the URL dotnet/Home/Index. Click on the View buttons. You will observe that manually generating the link will not work with the above changes in place, whereas the link generating with Tag Helpers will work as expected.

**Why Tag Helpers over HTML Helpers in ASP.NET Core MVC?**

Tag Helpers are often preferred over HTML Helpers in ASP.NET Core MVC for several reasons:

**Readable and HTML-Like Syntax**

Tag Helpers use HTML-like syntax, which makes them easier to read and work with, especially for front-end developers familiar with HTML.

**Example Using Tag Helper: <a asp-action=”Details” asp-controller=”Home” asp-route-id=”@student.StudentId” class=”btn btn-primary”>View</a>**  
**Example Using HTML Helper: @Html.ActionLink(“View”, “Details”, “Home”, new { id = student.StudentId }, new { @class = “btn btn-primary” })**

In the Tag Helper example, the code looks like standard HTML and the attributes are more self-explanatory (asp-action, asp-controller, asp-route-id). In contrast, the HTML Helper syntax is more complex and harder to read, especially for designers or developers coming from an HTML background.

**Better IntelliSense Support**

Tag Helpers are fully integrated with IntelliSense in Visual Studio, offering auto-completion, tooltips, and error checking directly in the Razor view. This improves productivity and reduces the chance of mistakes.

**Tag Helper Example: <input asp-for=”UserName” class=”form-control” />**  
As you type asp-for=”UserName”, IntelliSense will suggest model properties and provide contextual information.

**HTML Helper Example: @Html.TextBoxFor(m => m.UserName, new { @class = “form-control” })**  
IntelliSense is less effective in the HTML Helper example, making it harder to remember correct method signatures and syntax.

**Cleaner and More Maintainable Code**

Tag Helpers follow the natural HTML flow, resulting in cleaner and more maintainable Razor views, especially when working with complex forms or UI components.

**Tag Helper Example:**

**<form** asp-action="SubmitForm" method="post"**>**

**<div** class="form-group"**>**

**<label** asp-for="UserName" class="control-label"**></label>**

**<input** asp-for="UserName" class="form-control" **/>**

**</div>**

**<div** class="form-group"**>**

**<label** asp-for="Email" class="control-label"**></label>**

**<input** asp-for="Email" class="form-control" **/>**

**</div>**

**<button** type="submit" class="btn btn-primary"**>**Submit**</button>**

**</form>**

**HTML Helper Example:**

@using (Html.BeginForm("SubmitForm", "Home", FormMethod.Post))

{

**<div** class="form-group"**>**

@Html.LabelFor(m => m.UserName, new { @class = "control-label" })

@Html.TextBoxFor(m => m.UserName, new { @class = "form-control" })

**</div>**

**<div** class="form-group"**>**

@Html.LabelFor(m => m.Email, new { @class = "control-label" })

@Html.TextBoxFor(m => m.Email, new { @class = "form-control" })

**</div>**

**<button** type="submit" class="btn btn-primary"**>**Submit**</button>**

}

The code in the Tag Helper version is cleaner and follows an HTML-first approach. In contrast, the HTML Helper version embeds C# logic in the markup, which can make the view more difficult to maintain, especially as the form’s complexity grows.

In the next article, I will discuss the [**Image Tag Helper in ASP.NET Core MVC**](https://dotnettutorials.net/lesson/image-tag-helper-asp-net-core/) Application. In this article, I explain the basics of **Tag Helpers in an ASP.NET Core MVC** Application with examples. I hope you enjoy this article.