**Creating Custom Tag Helpers With ASP.NET Core MVC**

**Introduction**

The tag helper enables us to run the server-side code to generate HTML, CSS, Javascript code in Razor View. In my previous article, I have talked about [tag helper](https://www.c-sharpcorner.com/article/introduction-to-tag-helpers-in-asp-net-core/), [form tag helper, and other common tag helpers](https://www.c-sharpcorner.com/article/form-tag-helper-and-other-common-tag-helper-in-asp-net-core-mvc/). In this article, I will explain how to create custom tag helper in ASP.NET Core MVC.

Using the tag helper, we can extend our existing HTML elements or create our own custom HTML elements. The custom tag helper is nothing but the class that implements the ITagHelper interface. However, .NET Core MVC provides us the implementation of this interface.

To create custom tag helper, the first step is to create a class that inherits from "TagHelper" class. This class has a virtual method to generate HTML tags. It contains both synchronous (Process) and asynchronous (ProcessAsync) implementation of the virtual method.

1. **public** **virtual** Task ProcessAsync(TagHelperContext context, TagHelperOutput output);
2. **public** **virtual** **void** Process(TagHelperContext context, TagHelperOutput output);

We can implement either one of these two or both, based on our requirement. The Process method (or ProcessAsync) is responsible to generate the HTML that is rendered by the browser. It receives context of tag helper instance and TegHelperOuter which we can be used to read and change the content of our tag helper that is within scope.

**Example**

In the following example, I have created a simple tag helper that will generate < my-first-tag-helper > </ my-first-tag-helper > tag helper inside our supplied text.

Using HtmlTargetElement attribute, we can provide a tag helper target. It passes an attribute parameter that specifies the HTML element which contains an HTML attribute named, for example, "my-first-tag-helper". It will match and process the override method in the class it will run.

**TagHelper**

**namespace** CustomTagHelper.TagHelpers

{

**using** Microsoft.AspNetCore.Razor.TagHelpers;

**using** System.Text;

    [HtmlTargetElement("my-first-tag-helper")]

**public** **class** MyCustomTagHelper : TagHelper

    {

**public** string Name { get; set; }

**public** override **void** Process(TagHelperContext context, TagHelperOutput output)

        {

            output.TagName = "CustumTagHelper";

            output.TagMode = TagMode.StartTagAndEndTag;

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

            sb.AppendFormat("<span>Hi! {0}</span>", **this**.Name);

            output.PreContent.SetHtmlContent(sb.ToString());

        }

    }

}

The next step is to register the tag helper into \_ViewImports.cshtml file. If we add this file at root level of the View folder and register our tag helper, it will be available to all the Views under “View” folder. In this example, I have registered “\*”, so it will add all the tag helpers within the assembly.

**\_ViewImports.cshtml**

1. @addTagHelper \*, CustomTagHelper

Next, we need to use our tag helper into View. Visual Studio has rich intelliSense support for tag helpers.

**Index.cshtml**

1. <div **class**="row">
2. <my-first-tag-helper name="Jignesh Trivedi">
4. </my-first-tag-helper>
5. </div>

**Output**

**Passing a model Data to a Tag Helper**

We can also pass the model data to the tag helper via model binding by creating properties of type "ModelExpression". Using HtmlAttributeName attribute, we can create a friendly attribute name.

The ModelExpression describes a model expression passed to the tag helper. Using Model property of this class, we can get model object for the expression of interest.

In the following example, I have created a tag helper for showing employee details, such as name and designation. Here, I have created two properties for taking model expression: EmployeeName and Designation. Within Process method, I have used these properties and created HTML assigned to output. From the View, I have passed the property name of EmployeeViewModel and tag helper will evaluate the values for model property to use the value whenever required.

**Tag Helper**

1. **namespace** CustomTagHelper.TagHelpers
2. {
3. **using** Microsoft.AspNetCore.Mvc.ViewFeatures;
4. **using** Microsoft.AspNetCore.Razor.TagHelpers;
5. **using** System.Text;
7. [HtmlTargetElement("employee-details")]
8. **public** **class** EmployeeDetailTagHelper : TagHelper
9. {
10. [HtmlAttributeName("for-name")]
11. **public** ModelExpression EmployeeName { get; set; }
12. [HtmlAttributeName("for-designation")]
13. **public** ModelExpression Designation { get; set; }
14. **public** override **void** Process(TagHelperContext context, TagHelperOutput output)
15. {
16. output.TagName = "EmployeeDetails";
17. output.TagMode = TagMode.StartTagAndEndTag;
19. var sb = **new** StringBuilder();
20. sb.AppendFormat("<span>Name: {0}</span> <br/>", **this**.EmployeeName.Model);
21. sb.AppendFormat("<span>Designation: {0}</span>", **this**.Designation.Model);
23. output.PreContent.SetHtmlContent(sb.ToString());
24. }
25. }
26. }

**Model**

1. **namespace** CustomTagHelper.Models
2. {
3. **using** System.ComponentModel.DataAnnotations;
4. **public** **class** EmployeeViewModel
5. {
6. **public** string Name { get; set; }
7. **public** string Designation { get; set; }
8. }
9. }

**Controller**

1. **public** IActionResult Index1()
2. {
3. EmployeeViewModel e = **new** EmployeeViewModel();
4. e.Name = "Jignesh Trivedi";
5. e.Designation = "Senior Consultant";
6. **return** View(e);
7. }

**View**

1. @model CustomTagHelper.Models.EmployeeViewModel
2. <div **class**="row">
3. <employee-details **for**-name ="Name" **for**-designation="Designation"></employee-details>
4. </div>

**Output**

We can use ViewContext type within tag helper to access the View’s contextual information such as ModelState, HttpContext, etc. This is done by using ViewContext attribute and type by declaring property type to ViewContext and annotating with ViewContext attribute. The property should not be bound to the HTML attribute of tag helper if it is annotating with HttpAttributeNotBound attribute.

**Example**

1. [HtmlAttributeNotBound]
2. [ViewContext]
3. **public** ViewContext ViewContext { get; set; }

We can use ProcessAsync method to execute the tag helper with the given context asynchronously. The implementation is the same as the Process method, but we can call other async methods from this method and wait until it is finished.

1. **public** override Task ProcessAsync(TagHelperContext context, TagHelperOutput output)
2. {
4. }

**Summary**

With the tag helper, we can extend the existing element or create new elements. The tag helper allows us to create reusable attributes or elements. Tag helper does also follow MVC naming conventions (example - Controller). It ships with MVC provided methods and properties for tag helper class.