Tag Helpers

in ASP.NET Core

What are Tag Helpers?

- Tag helpers replace HTML helpers
- Tag Helpers attach to HTML elements in Razor views, while HTML Helpers are invoked as methods
- Purpose:
  To simplify the work required to dynamically modify a view's HTML output based on the data fed to it by the controller

Why?

- Tag helpers let you use “HTML helpers” in a view by simply extending the semantics of tags in your markup
  - rather than through ugly Razor method calls that disrupt the flow of HTML markup with embedded .NET code

- Less C# code in the View file!
  - much cleaner HTML code in your views

- **To ensure Forms are generated consistently**
- **Better IntelliSence support**

Use of Tag Helpers

- You do have to opt into using tag helpers by adding:

```
@addTagHelper *, Microsoft.AspNet.Mvc.TagHelpers
```

- In a default ASP.NET Web Application project using the ASP.NET Core Web Application preview template, you'll find this directive in the `_ViewImports.cshtml` file

Syntax

ASP4 MVC5 Html Helper syntax:

```
@Html.ValidationSummary("", new { @class = "text-danger" })
  <div class="form-group">
    @Html.LabelFor(m => m.Email, new { @class = "col-md-2 control-label" })
    <div class="col-md-10">
      @Html.TextBoxFor(m => m.Email, new { @class = "form-control" })
    </div>
  </div>
```

ASP Core MVC Tag Helper syntax:

```
<div asp-validation-summary="ValidationSummary.All" class="text-danger"></div>
  <div class="form-group">
    <asp-label for="Email" class="col-md-2 control-label"></label>
    <div class="col-md-10">
      <input asp-for="Email" class="form-control" />
      <span asp-validation-for="Email" class="text-danger"></span>
    </div>
  </div>
```

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

The FormTagHelper

- The main purpose of the FormTagHelper class is to set the action attribute of the form element using the application's routing configuration, ensuring that the form data is always sent to the correct URL, even when the routing scheme changes

```
<form method="post" asp-controller="Home" asp-action="Create">
    <div . . . >


</form>
```

- The tag helper adds an action attribute to the form element and sets its value using the routing system

```
<form method="post" action="/Home/Create">
```

Using the Anti-forgery Feature

- Cross-site request forgery (CSRF) is a way for hackers to exploit a web application to take advantage of the way that user requests are authenticated
  - For more info see: https://en.wikipedia.org/wiki/Cross-site_request_forgery
- If a form element doesn't contain an action attribute, then the FormTagHelper class automatically enables the anti-CSRF feature, whereby a security token is added to the form in a hidden input element to the HTML sent to the client along with a cookie
- The application will only process the request if it contains both the cookie and the hidden value from the form
  - which the malicious site cannot generate

```
<form method="post" action="/Home/Create">
    < . . .>
    <input name="__RequestVerificationToken" type="hidden" value="CfDJ8Lb7hKPOXURMvA_
</form>hackers
```

Validating Anti-forgery Tokens in the Controller

- Adding the security tokens to HTML responses is only part of the process
- They must also be validated by the controller to be useful!
- The ValidateAntoForgeryToken attribute ensures that a request contains valid anti-CSRF tokens and will throw an exception if they are absent or do not contain the expected values

```
[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Create(Product newProduct)
{
    repository.AddProduct(newProduct);
    return RedirectToAction("Index");
}
```

The asp-antiforgery attribute

- The FormTagHelper class provides the asp-antiforgery attribute to override the default anti-CSRF behaviour
- If the value of the attribute is true
  - then the security tokens will be included in responses, even if the form element has an action attribute
- If the value of the attribute is false
  - then the security tokens will be disabled

The Built-in Tag Helper Attributes for **Input Elements**

- The asp-for attribute is set to the **name of a view model property**, which is then used to set the name, id, type, and value attributes of the input element

```
@model City
<div class="form-group">
        <label asp-for="Name"></label>
        <input class="form-control" asp-for="Name" />
    </div>
```

```
public class City
{
   [Display(Name = "City")]
   public string Name { get; set; }
   public string Country { get; set; }
   [DisplayFormat(DataFormatString = "{(
```

```
<div class="form-group">
    <label for="Name">City</label>
    <input class="form-control" type="text" id="Name" name="Name" value="" />
</div>
```

```
 get; se
et; set;
```

From C# Property Types to html Input Type

- The type attribute of the input element tells the browser how to display the element in a form

C# Type	Input Element Type Attribute
byte, sbyte, int, uint, short, ushort, long, ulong	number
float , double , decimal	text (with additional attributes for model validation)
bool	checkbox
string	text
DateTime	datetime

- You can override the default mappings shown in the table by defining the type attribute on the input element
  - The tag helper won't override the value you define
- If you need to override the default mapping in multiple views, then you can apply the UIHint attribute to the property in the C# model class

Formatting Data Values

- The `asp-format` attribute accepts a value that will be passed to the standard C# string formatting system

```
<div class="form-group">
    <label asp-for="Population"></label>
    <input class="form-control" asp-for="Population" asp-format="{0:#,###}"/>
</div>
```

- This feature should be used with caution because you must ensure that the rest of the application is configured to support the format you use
  - And be aware of which culture you are using

Formatting via the Model Class

- If you always want to use the same formatting for a model property, then you can decorate the C# class with the `DisplayFormat` attribute
- The DisplayFormat attribute requires two arguments to format a data value:
  - The `DataFormatString` argument specifies the formatting string
  - The `ApplyFormatInEditMode` specifies that formatting should be used when values are being edited
- The asp-format attribute takes precedence over the DisplayFormat attribute if both are specified

```
[DisplayFormat(DataFormatString = "{0:#,###}", ApplyFormatInEditMode = false)]
public int? Population { get; set; }
```

Label asp-for

- The tag helper asp-for used on a label element will use the name of the view model property to set the value of the for attribute and the contents of the label element

```
<label asp-for="Name"></label>
```

```
<label asp-for="Country"></label>
```

```
<label for="Name">City</label>
```

```
<label for="Country">Country</label>
```

```
public class City
{
    [Display(Name = "City")]
    public string Name { get; set; }
    public string Country { get; set; }
}
```

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Working with Select and Option Elements

- The select and option elements are used to provide the user with a fixed set of choices
- The Built-in Tag Helper Attributes for select Elements
  - asp-for
    - is used to specify the view model property that the select element represents
  - asp-items
    - is used to specify a source of values for the option elements contained within the select element
- You may manually enter the options in the Razor code

```
<select class="form-control" asp-for="Country" >
            <option disabled selected value="">Select a Country</option>
            <option>UK</option>
            <option>USA</option>
            <option>France</option>
            <option>China</option>
        </select>
```

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Generating Option Elements from an enum

- If you have a fixed set of options to present to the user and don't want to duplicate them in views throughout the application, then you can use an enum and asp-items
- asp-items expect a sequence of SelectListItem objects so you need to generates souch a list in the controller or in the view

```
public enum CountryNames
    {
        UK,
        USA,
        France,
        China
    }
```

```
<select class="form-control" asp-for="Country"
        asp-items="@new SelectList(Enum.GetNames(typeof(CountryNames)))">
    <option disabled selected value="">Select a Country</option>
</select>
```

Generating Option Elements from the Model

- If you need to generate option elements to reflect the data in the model, then the simplest approach is to provide the data required to generate the elements through the ViewBag or ViewData

```
public ViewResult Edit()
{
    ViewBag.Countries = new SelectList(repository.Cities
            .Select(c => c.Country).Distinct());
    return View("Edit", repository.Cities.First());
}
```

```
<select class="form-control" asp-for="Country" asp-items="“@ViewBag.Countries">
    <option disabled selected value="">Select a Country</option>
</select>
```

The Built-In Tag Helper Attributes for Validation

- ValidationMessage
  - generates individual validation errors
- ValidationSummary
  - renders the validation summary message

OTHER BUILT-IN TAG HELPERS

the Hosting Environment Tag Helper

- The `EnvironmentTagHelper` class is applied to the custom environment element and determines whether a region of content is included in the HTML sent to the browser based on the hosting environment
- `names`
  - This attribute is used to specify a comma-separated list of hosting environment names for which the content contained within the environment element will be included in the HTML sent to the client

```
<environment names="development">
    <div class="panel-body bg-info"><h2>This is Development</h2></div>
</environment>
<environment names="production">
    <div class="panel-body bg-danger"><h2>This is Production</h2></div>
</environment>
```

- You may instead use `<environment include="Development">` and `<environment exclude="Development">`

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Tag Helper Attributes for Anchor Elements

Name	Description
asp-action	specifies the action method that the URL will target.
asp-controller	specifies the controller that the URL will target
asp-area	specifies the area that the URL will target
asp-fragment	is used to specify the URL fragment (which appears after the # character)
asp-host	specifies the name of the host that the URL will target
asp-protocol	specifies the protocol that the URL will use
asp-route	specifies the name of the route that will be used to generate the URL
asp-route-*	are used to specify additional values for the URL so that the asp-route-id attribute is used to provide a value for the id segment to the routing system

```
<a asp-action="Create" class="btn btn-primary">Create</a>
```

Writing Tag Helpers

```
[HtmlTargetElement("button", Attributes = "bs-button-color", ParentTag = "form")]
[HtmlTargetElement("a", Attributes = "bs-button-color", ParentTag = "form")]
public class ButtonTagHelper : TagHelper {
    public string BsButtonColor { get; set; }

    public override void Process(TagHelperContext context,
                                 TagHelperOutput output) {
        output.Attributes.SetAttribute("class", $"btn btn-{BsButtonColor}");
    }
}
```

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

References & Links

- Introduction to using tag helpers in forms in ASP.NET Core
  https://docs.microsoft.com/da-dk/aspnet/core/mvc/views/working-with-forms