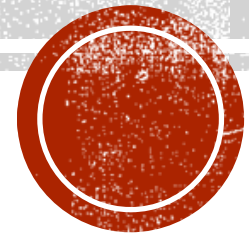


# ASP.NET CORE –BUILDING FORMS WITH TAG HELPERS

Kamal Beydoun

Lebanese University – Faculty of Sciences I

Kamal.Beydoun@ul.edu.lb



# INTRODUCTION

- Tag Helpers are new to ASP.NET Core and for **server-side** HTML rendering.
- They're **Razor components** that you can use to **customize** the **HTML** generated in your **templates**.
- Tag Helpers can be **added to a standard HTML element**, such as an `<input>`.
- They can also be **standalone elements** and can be used to generate completely customized HTML.

# APPLICATION OF THE CHAPTER

The image displays two side-by-side browser windows showing a web application named 'CurrencyConverter'.

**Left Window: Convert currency - CurrencyConv**

- Page Title:** Convert currency - CurrencyConv
- URL:** https://lo...
- Header:** CurrencyConverter
- Section:** Convert values
- Text:** Choose the values
- Form Fields:**
  - Currency From:** A dropdown menu showing 'GBP'.
  - Quantity:** A text input field containing '100'.
  - Currency To:** A dropdown menu showing 'USD'.
- Action:** A blue 'Submit' button.

**Right Window: Checkout - CurrencyConverter**

- Page Title:** Checkout - CurrencyConverter
- URL:** https://lo...
- Header:** CurrencyConverter
- Section:** Checkout
- Form Fields:**
  - Your name:** A text input field containing 'Andrew'.
  - Last name:** An empty text input field with a red border and a red error message below it: 'The Last name field is required.'
  - Email:** A text input field containing 'test@test.com'.
  - Phone number:** A text input field containing 'Not a number'.
- Action:** A blue 'Submit' button.

## Listing 8.1 User registration form using Tag Helpers

```
@page #A
@model ConvertModel #A
<form method="post">
    <div class="form-group">
        <label asp-for="CurrencyFrom"></label> #B
        <input class="form-control" asp-for="CurrencyFrom" /> #C
        <span asp-validation-for="CurrencyFrom"></span> #D
    </div>
    <div class="form-group">
        <label asp-for="Quantity"></label> #B
        <input class="form-control" asp-for="Quantity" /> #C
        <span asp-validation-for="Quantity"></span> #D
    </div>
    <div class="form-group">
        <label asp-for="CurrencyTo"></label> #B
        <input class="form-control" asp-for="CurrencyTo" /> #C
        <span asp-validation-for="CurrencyTo"></span> #D
    </div>
    <button type="submit" class="btn btn-primary">Submit</button>
</form>
```

#A This is the view for the Razor Page Convert.cshtml. The Model type is ConvertModel.

#B asp-for on Labels generates the caption for labels based on the view model.

#C asp-for on Inputs generate the correct type, value, name, and validation attributes for the model.

#D Validation messages are written to a span using Tag Helpers.

# TAG HELPERS VS HTML ELEMENTS

```
<form asp-action="Convert">
  <div class="form-group">
    <label asp-for="CurrencyFrom"></label>
    <input class="form-control" asp-for="CurrencyFrom" />
    <span asp-validation-for="CurrencyFrom"></span>
  </div>
  <div class="form-group">
    <label asp-for="Quantity"></label>
    <input class="form-control" asp-for="Quantity" />
    <span asp-validation-for="Quantity"></span>
  </div>
</form>
```

In Visual Studio, Tag Helpers are distinguishable from normal elements by being **bold** and a **different color**, C# is shaded, and IntelliSense is available.

Tag Helpers are **extra attributes** on standard HTML elements (**or new elements entirely**) that work by modifying the HTML element they're attached to.

# WHY TAG HELPERS ?

- Let us easily **integrate** server-side values, such as those in your **PageModel**, with the **generated HTML**.
- Tag Helpers are used to
  - Automatically **populate** the values from the PageModel property.
  - Choose the correct **id and name**, so that when the form is POSTed back to the Razor Page, the property will be model bound correctly.
  - Choose the **correct input type** to display.
  - Display any **validation** errors.

The screenshot shows a web browser window titled 'Convert currency - CurrencyCon: X'. The address bar shows 'https://lo...'. The page content is titled 'CurrencyConverter' and 'Convert values'. Below the title, it says 'Choose the values'. There are three input fields: 'Currency From' (a dropdown menu showing 'GBP'), 'Quantity' (a text input field with '25'), and 'Currency To' (a text input field with '123'). A blue 'Submit' button is at the bottom. Annotations with arrows point to these fields: 'Label caption calculated from [Display] attribute' points to 'Currency From'; 'Input types determined from DataAnnotations and property type' points to 'Quantity' and 'Currency To'; and 'Validation error message populated from ModelState' points to the text 'Not a valid currency code' below the 'Currency To' field.

# CREATING FORMS USING TAG HELPERS

- You can use Tag Helpers
  - to **generate HTML markup** based on **properties** of your PageModel,
  - creating the correct **id** and **name** attributes of the element
  - **setting the value** of the element to the model property's value

# DATAANNOTATIONS ATTRIBUTES

**Listing 8.2** UserBindingModel for creating a user on a checkout page

```
public class UserBindingModel
{
    [Required]
    [StringLength(100, ErrorMessage = "Maximum length is {1}")]
    [Display(Name = "Your name")]
    public string FirstName { get; set; }

    [Required]
    [StringLength(100, ErrorMessage = "Maximum length is {1}")]

    [Display(Name = "Last name")]
    public string LastName { get; set; }

    [Required]
    [EmailAddress]
    public string Email { get; set; }

    [Phone(ErrorMessage = "Not a valid phone number.")]
    [Display(Name = "Phone number")]
    public string PhoneNumber { get; set; }
}
```

These attributes are also used by the Razor templating language to provide the **metadata required** to **generate the correct HTML** when you use Tag Helpers.

**For simplicity, We are using the same object for both binding model and view model, but in practice you should use two separate objects.**



CurrencyConverter

## Checkout

Your name

Last name

The Last name field is required.

Email

Phone number

Submit

**Listing 8.3 Razor template for binding to `UserBindingModel` on the checkout page**

```
@page
@model CheckoutModel                                     #A
@{
    ViewData["Title"] = "Checkout";

}
<h1>@ViewData["Title"]</h1>
<form asp-page="Checkout"                                #B
    <div class="form-group">
        <label asp-for="Input.FirstName"></label>         #C
        <input class="form-control" asp-for="Input.FirstName" />
        <span asp-validation-for="Input.FirstName"></span>
    </div>
    <div class="form-group">
        <label asp-for="Input.LastName"></label>
        <input class="form-control" asp-for="Input.LastName" />
        <span asp-validation-for="Input.LastName"></span>
    </div>
    <div class="form-group">
        <label asp-for="Input.Email"></label>
        <input class="form-control" asp-for="Input.Email" />   #D
        <span asp-validation-for="Input.Email"></span>
    </div>
    <div class="form-group">
        <label asp-for="Input.PhoneNumber"></label>
        <input class="form-control" asp-for="Input.PhoneNumber" />
        <span asp-validation-for="Input.PhoneNumber"></span>   #E
    </div>
    <button type="submit" class="btn btn-primary">Submit</button>
</form>
```

- #A The `CheckoutModel` is the `PageModel`, which exposes a `UserBindingModel` on the `Input` property
- #B Form Tag Helpers use routing to determine the URL the form will be posted to.
- #C The Label Tag Helper uses `DataAnnotations` on a property to determine the caption to display.
- #D The Input Tag Helper uses `DataAnnotations` to determine the type of input to generate.
- #E The Validation Tag Helper displays error messages associated with the given property.

**Listing 8.4 HTML generated by the Razor template on the checkout page**

```
<form action="/Checkout" method="post">
  <div class="form-group">
    <label for="Input_FirstName">Your name</label>
    <input class="form-control" type="text"
      data-val="true" data-val-length="Maximum length is 100"
      id="Input_FirstName" data-val-length-max="100"
      data-val-required="The Your name field is required."
      maxlength="100" name="Input.FirstName" value="" />
    <span data-valmsg-for="Input.FirstName"
      class="field-validation-valid" data-valmsg-replace="true"></span>
  </div>
  <div class="form-group">
    <label for="Input_LastName">Your name</label>
    <input class="form-control" type="text"
      data-val="true" data-val-length="Maximum length is 100"
      id="Input_LastName" data-val-length-max="100"
      data-val-required="The Your name field is required."
      maxlength="100" name="Input.LastName" value="" />
    <span data-valmsg-for="Input.LastName"
      class="field-validation-valid" data-valmsg-replace="true"></span>
  </div>
  <div class="form-group">
    <label for="Input_Email">Email</label>
    <input class="form-control" type="email" data-val="true"
      data-val-email="The Email field is not a valid e-mail address."
      data-val-required="The Email field is required."
      id="Input_Email" name="Input.Email" value="" />
    <span class="text-danger field-validation-valid"
      data-valmsg-for="Input.Email" data-valmsg-replace="true"></span>
  </div>
  <div class="form-group">
    <label for="Input_PhoneNumber">Phone number</label>
    <input class="form-control" type="tel" data-val="true"
      data-val-phone="Not a valid phone number." id="Input_PhoneNumber"
      name="Input.PhoneNumber" value="" />
    <span data-valmsg-for="Input.PhoneNumber"
      class="text-danger field-validation-valid"
      data-valmsg-replace="true"></span>
  </div>
  <button type="submit" class="btn btn-primary">Submit</button>
  <input name="__RequestVerificationToken" type="hidden"
    value="CfDJ8PkYhAINFx1JmYUVIDWbpPy_TRUNCATED" />
</form>
```

Even better than this, you can also set attributes that are normally generated by a Tag Helper, like the type attribute on an `<input>` element.

# ASP-PAGE

**<form asp-page="Checkout">**

This resulted in the addition of action and method attributes to the final HTML, indicating the **URL that the form should be sent to** when submitted:

**<form action="/Checkout" method="post">**

# ASP-ROUTE-\*

➤ Used to set arbitrary route parameters.

**<form asp-page = "Product" asp-route-id="5">**

Will generate:

**<form action="/Product/5" method="post">**

- You can add **as many** asp-route-\* attributes as necessary to your <form> to generate the correct action URL.
- You can also set the Razor Page **handler** to use using the **asp-page-handler** attribute.

# THE LABEL TAG HELPER

- Used to generate the **caption** (the visible text) and the **for attribute** for a **<label>** element, based on the properties in the view model.
- **[Display]** DataAnnotations attribute determines the appropriate value to display.

```
public class UserBindingModel
{
    [Display(Name = "Your name")]
    public string FirstName { get; set; }
    public string Email { get; set; }
}
```

The following Razor:

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

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

would generate the HTML

```
<label for="FirstName">Your Name</label>
```

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

# THE LABEL TAG HELPER

As well as properties on the PageModel, you can also reference sub-properties on child objects.

```
public class CheckoutModel: PageModel
{
    [BindProperty]
    public UserBindingModel Input { get; set; }
}
```

```
<label asp-for="Input.FirstName"></label>
<label asp-for="Input.Email"></label>
```

# THE LABEL TAG HELPER

- If you **don't want to use the caption generated by the helper**, you could insert your own manually.

**<label asp-for="Email">Please enter your Email</label>**

would generate the HTML

**<label for="Email">Please enter your Email</label>**

# THE INPUT AND TEXTAREA TAG HELPERS

Demo - TagHelpers

TagHelpers

☐ Checkbox entry

Number entry

0

Date entry

dd/mm/yyyy --:--

April 2020

Mo	Tu	We	Th	Fr	Sa	Su
30	31	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	1	2	3
4	5	6	7	8	9	10

Today

15 42

16 43

17 44

18 45

19 46

20 47

21 48

✓ ✕



# THE INPUT AND TEXTAREA TAG HELPERS

- To determine the **type of the input** element to generate, Tag Helpers uses information based on:
  - the **type of the property** (bool, string, int, and so on)
  - **DataAnnotations attributes** applied to the property
- **DataAnnotations** are also used to add data-val-\* **client-side validation** attributes to the generated HTML.

# EXAMPLE

```
<input asp-for="Input.Email" />
```

**Will generate**

```
<input type="email" id="Input_Email" name="Input.Email"  
value="test@example.com" data-val="true"  
data-val-email="The Email Address field is not a valid e-mail address."  
data-val-required="The Email Address field is required."  
>
```

**Perhaps the most striking addition is the swath of data-val-\* attributes. These can be used by client-side JavaScript libraries such as jQuery to provide client-side validation of your DataAnnotations constraints.**

## Client-side validation

In order to enable client-side validation in your application, you need to add some jQuery libraries to your HTML pages. In particular, you need to include the jQuery, jQuery-validation, and jQuery-validation-unobtrusive JavaScript libraries. You can do this in a number of ways, but the simplest is to include the script files at the bottom of your view using

```
<script src="~/lib/jquery-validation/dist/jquery.validate.min.js"></script>
<script src="~/lib/jquery-validation-
unobtrusive/jquery.validate.unobtrusive.min.js"></script>
```

The default templates include these scripts for you, in a handy partial template that you can add to your page in a `Scripts` section. If you're using the default layout and need to add client-side validation to your view, add the following section somewhere on your view:

```
@section Scripts{
    @Html.Partial("_ValidationScriptsPartial")
}
```

This partial view references files in your `wwwroot` folder. The default `_layout` template includes jQuery itself, as that's required by the front-end component library Bootstrap.<sup>35</sup>

# INPUT TAG HELPERS

Data type	How it's specified	Input element type
<code>byte, int, short, long, uint</code>	Property type	number
<code>decimal, double, float</code>	Property type	text
<code>string</code>	Property type, <code>[DataType(DataType.Text)]</code> attribute	text
<code>HiddenInput</code>	<code>[HiddenInput]</code> attribute	hidden
<code>Password</code>	<code>[Password]</code> attribute	password
<code>Phone</code>	<code>[Phone]</code> attribute	tel
<code>EmailAddress</code>	<code>[EmailAddress]</code> attribute	email
<code>Url</code>	<code>[Url]</code> attribute	url
<code>Date</code>	<code>DateTime</code> property type, <code>[DataType(DataType.Date)]</code> attribute	date

As always, you can override the generated type by adding your own type attribute to the Razor.

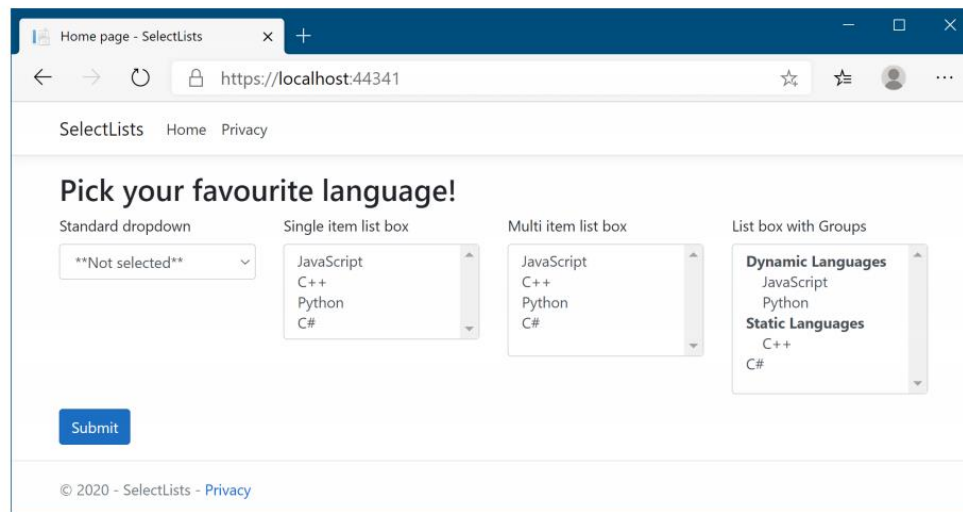
# TEXT AREA TAG HELPER

```
<textarea asp-for="Multiline"></textarea>
```

generates

```
<textarea data-val="true" id="Multiline" name="Multiline"  
data-val-length="Maximum length 200." data-val-length-max="200"  
data-val-required="The Multiline field is required." >This is some text,  
I'm going to display it in a text area</textarea>
```

# THE SELECT TAG HELPER



**Listing 8.5 View model for displaying select element dropdowns and list boxes**

```
public class SelectListModel: PageModel
{
    [BindProperty]
    public class InputModel Input { get; set; } #A
    #A

    public IEnumerable<SelectListItem> Items { get; set; } #B
    #B
    = new List<SelectListItem>
    {
        new SelectListItem{Value= "csharp", Text="C#"}, #B
        new SelectListItem{Value= "python", Text= "Python"}, #B
        new SelectListItem{Value= "cpp", Text="C++"}, #B
        new SelectListItem{Value= "java", Text="Java"}, #B
        new SelectListItem{Value= "js", Text="JavaScript"}, #B
        new SelectListItem{Value= "ruby", Text="Ruby"}, #B
    }; #B

    public class InputModel
    {
        public string SelectedValue1 { get; set; } #C
        public string SelectedValue2 { get; set; } #C
        public IEnumerable<string> MultiValues { get; set; } #D
    }
}
```

#A The InputModel for binding the user's selections to the select boxes

#B The list of items to display in the select boxes

#C These properties will hold the values selected by the single-selection select boxes.

#D To create a multiselect list box, use an IEnumerable<>.

**The Select Tag Helper only works with SelectListItem elements.**

**To use <select> elements in your Razor code, you'll need to include two properties in your PageModel: one property for the list of options to display and one to hold the value (or values) selected.**

# THE SELECT TAG HELPER

**Listing 8.6 Razor template to display a select element in three different ways**

```
@page
@model SelectListModel
```

```
<select asp-for="Input.SelectedValue1"           #A
        asp-items="Model.Items"></select>         #A
<select asp-for="Input.SelectedValue2"           #B
        asp-items="Model.Items" size="4"></select> #B
<select asp-for="Input.MultiValues"               #C
        asp-items="Model.Items"></select>         #C
```

#A Creates a standard drop-down select list by binding to a standard property in asp-for  
#B Creates a single-select list box of height 4 by providing the standard HTML size attribute  
#C Creates a multiselect list box by binding to an IEnumerable property in asp-for

- **asp-for** attribute specifies the **property** in your view model to bind to.
- **asp-items** attribute is provided the **IEnumerable<SelectListItem>** to display the available <option>.

# DISPLAY GROUPS IN YOUR LIST BOXES

The screenshot shows a web browser window with the address bar displaying `https://localhost:44341`. The page title is "Home page - SelectLists". The navigation bar includes links for "SelectLists", "Home", and "Privacy".

The main heading is "Pick your favourite language!". Below this, there are four examples of list boxes:

- Standard dropdown:** A dropdown menu with the text "\*\*Not selected\*\*" and a downward arrow.
- Single item list box:** A list box containing the items: JavaScript, C++, Python, and C#.
- Multi item list box:** A list box containing the items: JavaScript, C++, Python, and C#.
- List box with Groups:** A list box with two groups of items. The first group is "Dynamic Languages" and contains JavaScript and Python. The second group is "Static Languages" and contains C++ and C#.

A blue "Submit" button is located below the list boxes.

At the bottom of the page, there is a footer that reads "© 2020 - SelectLists - [Privacy](#)".



```

public class SelectListModel: PageModel
{
    [BindProperty]
    public IEnumerable<string> SelectedValues { get; set; }
    public IEnumerable<SelectListItem> Items { get; set; }

    public SelectListModel() #A
    {
        var dynamic = new SelectListGroup { Name = "Dynamic" }; #B
        var stat = new SelectListGroup { Name = "Static" }; #B
        Items = new List<SelectListItem>
        {
            new SelectListItem {
                Value= "js",
                Text="Javascript",
                Group = dynamic #C
            },
            new SelectListItem {
                Value= "cpp",
                Text="C++",
                Group = stat #C
            },
            new SelectListItem {
                Value= "python",
                Text="Python",
                Group = dynamic #C
            },
            new SelectListItem { #D
                Value= "csharp",
                Text="C#",
            } #D
        };
    }
}

```

```

@page
@model SelectListModel
<select asp-for="SelectedValues" asp-items="Model.Items"></select>

```

would be rendered to HTML as:

```

<select id="SelectedValues" name="SelectedValues" multiple="multiple">
  <optgroup label="Dynamic">
    <option value="js">JavaScript</option>
    <option value="python">Python</option>
  </optgroup>
  <optgroup label="Static Languages">
    <option value="cpp">C++</option>
    <option value="csharp">C#</option>
  </optgroup>
</select>

```

#A Initializes the list items in the constructor

#B Creates single instance of each group to pass to SelectListItems

#C Sets the appropriate group for each SelectListItem

#D If a SelectListItem doesn't have a Group, it won't be added to an <optgroup>.

# SELECT WITH NO VALUES

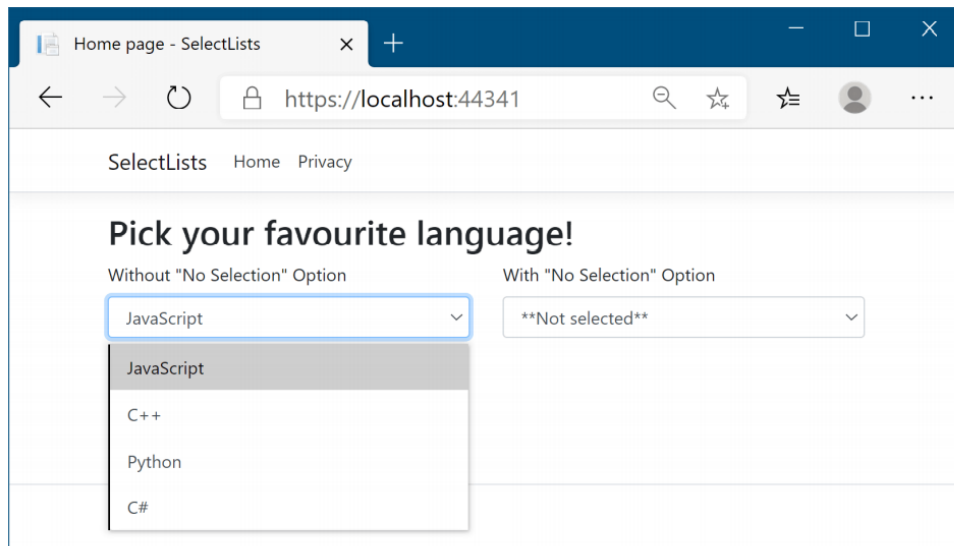


Figure 8.7 Without a “no selection” option, the `<select>` element will always have a value. This may not be the behavior you desire if you don’t want an `<option>` to be selected by default.

You can achieve this in one of two ways: you could either add the “not selected” option to the available `SelectListItem`s, or you could manually add the option to the Razor, for example by using

```
<select asp-for="SelectedValue" asp-items="Model.Items">
  <option Value = "">**Not selected**</option>
</select>
```

This will add an extra `<option>` at the top of your `<select>` element, with a blank `Value` attribute, allowing you to provide a “no selection” option for the user.

# THE VALIDATION MESSAGE TAG HELPER

- This can be achieved for **each property** in your view model using the Validation Message Tag Helper applied to a `<span>` by using the **asp-validation-for** attribute:

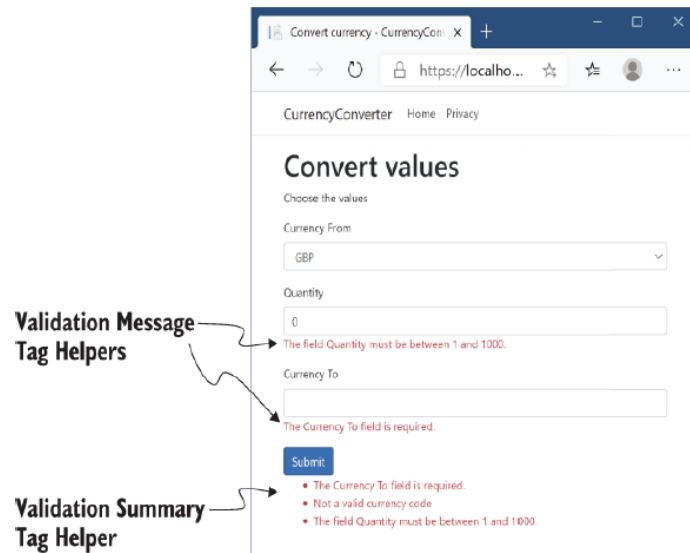
`<span asp-validation-for="Email"></span>`

Email

The Email field is required.

When an error occurs during **client-side validation** or **server-side validation**, the appropriate error message for the referenced property will be displayed in the `<span>`.

# VALIDATION SUMMARY TAG HELPER



Applied to a `<div>` using the **asp-validation-summary** attribute and providing a `ValidationSummary` enum value, such as:

**`<div asp-validation-summary="All"></div>`**

The `ValidationSummary` enum controls which values are displayed, and has three possible values:

- **None**—Don't display a summary.
- **ModelOnly**—Only display errors that are not associated with a property.
- **All**—Display errors either associated with a property or with the model.

```
public class ConvertModel : PageModel
```

```
{
    [BindProperty]
    public InputModel Input { get; set; }

    [HttpPost]
    public IActionResult OnPost()
    {
        if(Input.CurrencyFrom == Input.CurrencyTo)           #A
        {
            ModelState.AddModelError(                        #B
                string.Empty,                                #B
                "Cannot convert currency to itself");         #B
        }
        if (!ModelState.IsValid)                             #C
        {
            return Page();                                   #C
        }                                                     #C

        //store the valid values somewhere etc
        return RedirectToPage("Checkout");
    }
}
```

#A Can't convert currency to itself

#B Adds model-level error, not tied to a specific property, by using empty key

#C If there are any property-level or model-level errors, display them.

Identical currencies  
cause a model-level  
validation error.

Without the Tag Helper,  
the user has no idea  
why the form has been  
redisplayed.

Identical currencies  
cause a model-level  
validation error.

The Tag Helper  
shows model-level  
errors.

# GENERATING LINKS WITH THE ANCHOR TAG HELPER

- The Anchor Tag Helper can be used to generate the URL for a given page handler using **routing**.
  - you provide **asp-page** and **asp-page-handler** attributes, along with **asp-route-\*** attributes as necessary.

```
<ul class="navbar-nav flex-grow-1">  
  <li class="nav-item">  
    <a class="nav-link text-dark"  
      asp-area="" asp-page="/Index">Home</a>  
  </li>  
  <li class="nav-item">  
    <a class="nav-link text-dark"  
      asp-area="" asp-page="/Privacy">Privacy</a>  
  </li>  
</ul>
```

```
<ul class="nav navbar-nav">  
  <li class="nav-item">  
    <a class="nav-link text-dark" href="/">Home</a>  
  </li>  
  <li class="nav-item">  
    <a class="nav-link text-dark" href="/Privacy">Privacy</a>  
  </li>  
</ul>
```

# ANCHOR TAG HELPER PROPERTIES

- **asp-page**—Sets the Razor Page to execute.
- **asp-page-handler**—Sets the Razor Page handler to execute.
- **asp-area**—Sets the area route parameter to use. Areas can be used to provide an additional layer of organization to your application.
- **asp-host**—If set, the link will point to the provided host and will generate an absolute URL instead of a relative URL.
- **asp-protocol**—Sets whether to generate an http or https link. If set, it will generate an absolute URL instead of a relative URL.
- **asp-route**—Uses the named route to generate the URL.
- **asp-route-\***—Sets the route parameters to use during generation. Can be added multiple times for different route parameters.

# CACHE-BUSTING WITH THE APPEND VERSION TAG HELPER

- For performance reasons, browsers often cache files locally and reuse them for subsequent requests.
- A cache-busting query string adds a query parameter to a URL, such as ?v=1. Browsers will cache the response and use it for subsequent requests to the URL.
- When the **resource changes**, the query string is also changed, for example to ?v=2.
- Browsers will see this is a request for a new resource, and will make a fresh request.

When an application goes into production, is ensuring that browsers are all using the latest files.



# APPEND TAG HELPER

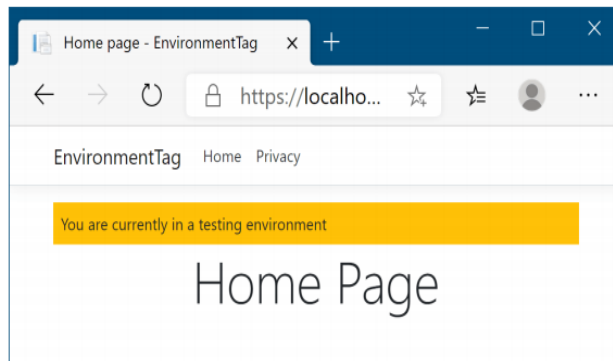
```
<script src="/js/site.js" asp-append-version="true"></script>
```

The **asp-append-version** attribute will load the file being referenced and generate a unique hash based on its contents. This is then appended as a unique query string to the resource URL:

```
<script  
src="/js/site.js?v=EWaMeWsJBYWmL2g_KkgXZQ5nPe"></script>
```

# USING CONDITIONAL MARKUP WITH THE ENVIRONMENT TAG HELPER

- In many cases, you want to render different HTML in your Razor templates depending if your website is running in a **development** or **production** environment.



```
@if(env == "Testing" || env == "Staging")
{
    <div class="warning">You are currently on a testing
environment</div>
}

<environment include="Testing,Staging">
    <div class="warning">You are currently on a testing
environment</div>
</environment>
```