

Tag Helpers in forms in ASP.NET Core

Article • 07/10/2023

By [Rick Anderson](#), [N. Taylor Mullen](#), [Dave Paquette](#), and [Jerrie Pelser](#)

This document demonstrates working with Forms and the HTML elements commonly used on a Form. The HTML [Form](#) element provides the primary mechanism web apps use to post back data to the server. Most of this document describes [Tag Helpers](#) and how they can help you productively create robust HTML forms. We recommend you read [Introduction to Tag Helpers](#) before you read this document.

In many cases, HTML Helpers provide an alternative approach to a specific Tag Helper, but it's important to recognize that Tag Helpers don't replace HTML Helpers and there's not a Tag Helper for each HTML Helper. When an HTML Helper alternative exists, it's mentioned.

The Form Tag Helper

The [Form Tag Helper](#):

- Generates the HTML `<FORM>` `action` attribute value for a MVC controller action or named route
- Generates a hidden [Request Verification Token](#) to prevent cross-site request forgery (when used with the `[ValidateAntiForgeryToken]` attribute in the HTTP Post action method)
- Provides the `asp-route-<Parameter Name>` attribute, where `<Parameter Name>` is added to the route values. The `routeValues` parameters to `Html.BeginForm` and `Html.BeginRouteForm` provide similar functionality.
- Has an HTML Helper alternative `Html.BeginForm` and `Html.BeginRouteForm`

Sample:

C#HTML

```
<form asp-controller="Demo" asp-action="Register" method="post">
  <!-- Input and Submit elements -->
</form>
```

The Form Tag Helper above generates the following HTML:

HTML

```
<form method="post" action="/Demo/Register">
  <!-- Input and Submit elements -->
  <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>">
</form>
```

The MVC runtime generates the `action` attribute value from the Form Tag Helper attributes `asp-controller` and `asp-action`. The Form Tag Helper also generates a hidden [Request Verification Token](#) to prevent cross-site request forgery (when used with the `[ValidateAntiForgeryToken]` attribute in the HTTP Post action method). Protecting a pure HTML Form from cross-site request forgery is difficult, the Form Tag Helper provides this service for you.

Using a named route

The `asp-route` Tag Helper attribute can also generate markup for the HTML `action` attribute. An app with a [route](#) named `register` could use the following markup for the registration page:

C#HTML

```
<form asp-route="register" method="post">
  <!-- Input and Submit elements -->
</form>
```

Many of the views in the *Views/Account* folder (generated when you create a new web app with *Individual User Accounts*) contain the `asp-route-returnurl` attribute:

C#HTML

```
<form asp-controller="Account" asp-action="Login"
      asp-route-returnurl="@ViewData["ReturnUrl"]">
```

```
method="post" class="form-horizontal" role="form">
```

Note

With the built in templates, `returnUrl` is only populated automatically when you try to access an authorized resource but are not authenticated or authorized. When you attempt an unauthorized access, the security middleware redirects you to the login page with the `returnUrl` set.

The Form Action Tag Helper

The Form Action Tag Helper generates the `formaction` attribute on the generated `<button ...>` or `<input type="image" ...>` tag. The `formaction` attribute controls where a form submits its data. It binds to `<input>` elements of type `image` and `<button>` elements. The Form Action Tag Helper enables the usage of several [AnchorTagHelper](#) `asp-` attributes to control what `formaction` link is generated for the corresponding element.

Supported [AnchorTagHelper](#) attributes to control the value of `formaction`:

[Expand table](#)

| Attribute | Description |
|------------------------------------|---|
| asp-controller | The name of the controller. |
| asp-action | The name of the action method. |
| asp-area | The name of the area. |
| asp-page | The name of the Razor page. |
| asp-page-handler | The name of the Razor page handler. |
| asp-route | The name of the route. |
| asp-route-(value) | A single URL route value. For example, <code>asp-route-id="1234"</code> . |
| asp-all-route-data | All route values. |
| asp-fragment | The URL fragment. |

Submit to controller example

The following markup submits the form to the `Index` action of `HomeController` when the input or button are selected:

CSHTML

```
<form method="post">
  <button asp-controller="Home" asp-action="Index">Click Me</button>
  <input type="image" src="..." alt="Or Click Me" asp-controller="Home"
        asp-action="Index">
</form>
```

The previous markup generates following HTML:

HTML

```
<form method="post">
  <button formaction="/Home">Click Me</button>
  <input type="image" src="..." alt="Or Click Me" formaction="/Home">
</form>
```

Submit to page example

The following markup submits the form to the `About` Razor Page:

CSHTML

```
<form method="post">
  <button asp-page="About">Click Me</button>
```

```
<input type="image" src="..." alt="Or Click Me" asp-page="About">
</form>
```

The previous markup generates following HTML:

HTML

```
<form method="post">
  <button formaction="/About">Click Me</button>
  <input type="image" src="..." alt="Or Click Me" formaction="/About">
</form>
```

Submit to route example

Consider the `/Home/Test` endpoint:

C#

```
public class HomeController : Controller
{
    [Route("/Home/Test", Name = "Custom")]
    public string Test()
    {
        return "This is the test page";
    }
}
```

The following markup submits the form to the `/Home/Test` endpoint.

CSHTML

```
<form method="post">
  <button asp-route="Custom">Click Me</button>
  <input type="image" src="..." alt="Or Click Me" asp-route="Custom">
</form>
```

The previous markup generates following HTML:

HTML

```
<form method="post">
  <button formaction="/Home/Test">Click Me</button>
  <input type="image" src="..." alt="Or Click Me" formaction="/Home/Test">
</form>
```

The Input Tag Helper

The Input Tag Helper binds an HTML `<input>` element to a model expression in your razor view.

Syntax:

CSHTML

```
<input asp-for="<Expression Name>">
```

The Input Tag Helper:

- Generates the `id` and `name` HTML attributes for the expression name specified in the `asp-for` attribute. `asp-for="Property1.Property2"` is equivalent to `m => m.Property1.Property2`. The name of the expression is what is used for the `asp-for` attribute value. See the [Expression names](#) section for additional information.
- Sets the HTML `type` attribute value based on the model type and [data annotation](#) attributes applied to the model property
- Won't overwrite the HTML `type` attribute value when one is specified
- Generates [HTML5](#) validation attributes from [data annotation](#) attributes applied to model properties

- Has an HTML Helper feature overlap with `Html.TextBoxFor` and `Html.EditorFor`. See the [HTML Helper alternatives to Input Tag Helper](#) section for details.
- Provides strong typing. If the name of the property changes and you don't update the Tag Helper you'll get an error similar to the following:

```
An error occurred during the compilation of a resource required to process
this request. Please review the following specific error details and modify
your source code appropriately.

Type expected
'RegisterViewModel' does not contain a definition for 'Email' and no
extension method 'Email' accepting a first argument of type 'RegisterViewModel'
could be found (are you missing a using directive or an assembly reference?)
```

The `Input` Tag Helper sets the HTML `type` attribute based on the .NET type. The following table lists some common .NET types and generated HTML type (not every .NET type is listed).

[Expand table](#)

| .NET type | Input Type |
|----------------|---|
| Bool | type="checkbox" |
| String | type="text" |
| DateTime | type="datetime-local" ↗ |
| Byte | type="number" |
| Int | type="number" |
| Single, Double | type="number" |

The following table shows some common [data annotations](#) attributes that the input tag helper will map to specific input types (not every validation attribute is listed):

[Expand table](#)

| Attribute | Input Type |
|-------------------------------|-----------------|
| [EmailAddress] | type="email" |
| [Url] | type="url" |
| [HiddenInput] | type="hidden" |
| [Phone] | type="tel" |
| [DataType(DataType.Password)] | type="password" |
| [DataType(DataType.Date)] | type="date" |
| [DataType(DataType.Time)] | type="time" |

Sample:

```
C#

using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class RegisterViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email Address")]
        public string Email { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }
    }
}
```

C#HTML

```
@model RegisterViewModel

<form asp-controller="Demo" asp-action="RegisterInput" method="post">
    Email: <input asp-for="Email" /> <br />
    Password: <input asp-for="Password" /><br />
    <button type="submit">Register</button>
</form>
```

The code above generates the following HTML:

HTML

```
<form method="post" action="/Demo/RegisterInput">
    Email:
    <input type="email" data-val="true"
        data-val-email="The Email Address field is not a valid email address."
        data-val-required="The Email Address field is required."
        id="Email" name="Email" value=""><br>
    Password:
    <input type="password" data-val="true"
        data-val-required="The Password field is required."
        id="Password" name="Password"><br>
    <button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>">
</form>
```

The data annotations applied to the `Email` and `Password` properties generate metadata on the model. The Input Tag Helper consumes the model metadata and produces [HTML5](#) `data-val-*` attributes (see [Model Validation](#)). These attributes describe the validators to attach to the input fields. This provides unobtrusive HTML5 and [jQuery](#) validation. The unobtrusive attributes have the format `data-val-rule="Error Message"`, where rule is the name of the validation rule (such as `data-val-required`, `data-val-email`, `data-val-maxlength`, etc.) If an error message is provided in the attribute, it's displayed as the value for the `data-val-rule` attribute. There are also attributes of the form `data-val-ruleName-argumentName="argumentValue"` that provide additional details about the rule, for example, `data-val-maxlength-max="1024"`.

When binding multiple `input` controls to the same property, the generated controls share the same `id`, which makes the generated mark-up invalid. To prevent duplicates, specify the `id` attribute for each control explicitly.

Checkbox hidden input rendering

Checkboxes in HTML5 don't submit a value when they're unchecked. To enable a default value to be sent for an unchecked checkbox, the Input Tag Helper generates an additional hidden input for checkboxes.

For example, consider the following Razor markup that uses the Input Tag Helper for a boolean model property `IsChecked`:

C#HTML

```
<form method="post">
    <input asp-for="@Model.IsChecked" />
    <button type="submit">Submit</button>
</form>
```

The preceding Razor markup generates HTML markup similar to the following:

HTML

```
<form method="post">
    <input name="IsChecked" type="checkbox" value="true" />
    <button type="submit">Submit</button>

    <input name="IsChecked" type="hidden" value="false" />
</form>
```

The preceding HTML markup shows an additional hidden input with a name of `IsChecked` and a value of `false`. By default, this hidden input is rendered at the end of the form. When the form is submitted:

- If the `IsChecked` checkbox input is checked, both `true` and `false` are submitted as values.
- If the `IsChecked` checkbox input is unchecked, only the hidden input value `false` is submitted.

The ASP.NET Core model-binding process reads only the first value when binding to a `bool` value, which results in `true` for checked checkboxes and `false` for unchecked checkboxes.

To configure the behavior of the hidden input rendering, set the `CheckBoxHiddenInputRenderMode` property on `MvcViewOptions.HtmlHelperOptions`. For example:

```
C#

services.Configure<MvcViewOptions>(options =>
    options.HtmlHelperOptions.CheckBoxHiddenInputRenderMode =
        CheckBoxHiddenInputRenderMode.None);
```

The preceding code disables hidden input rendering for checkboxes by setting `CheckBoxHiddenInputRenderMode` to `CheckBoxHiddenInputRenderMode.None`. For all available rendering modes, see the `CheckBoxHiddenInputRenderMode` enum.

HTML Helper alternatives to Input Tag Helper

`Html.TextBox`, `Html.TextBoxFor`, `Html.Editor` and `Html.EditorFor` have overlapping features with the Input Tag Helper. The Input Tag Helper will automatically set the `type` attribute; `Html.TextBox` and `Html.TextBoxFor` won't. `Html.Editor` and `Html.EditorFor` handle collections, complex objects and templates; the Input Tag Helper doesn't. The Input Tag Helper, `Html.EditorFor` and `Html.TextBoxFor` are strongly typed (they use lambda expressions); `Html.TextBox` and `Html.Editor` are not (they use expression names).

HtmlAttributes

`@Html.Editor()` and `@Html.EditorFor()` use a special `ViewDataDictionary` entry named `htmlAttributes` when executing their default templates. This behavior is optionally augmented using `additionalViewData` parameters. The key "htmlAttributes" is case-insensitive. The key "htmlAttributes" is handled similarly to the `htmlAttributes` object passed to input helpers like `@Html.TextBox()`.

```
C#HTML

@Html.EditorFor(model => model.YourProperty,
    new { htmlAttributes = new { @class="myCssClass", style="Width:100px" } })
```

Expression names

The `asp-for` attribute value is a `ModelExpression` and the right hand side of a lambda expression. Therefore, `asp-for="Property1"` becomes `m => m.Property1` in the generated code which is why you don't need to prefix with `Model`. You can use the "@" character to start an inline expression and move before the `m.:`

```
C#HTML

@{
    var joe = "Joe";
}

<input asp-for="@joe">
```

Generates the following:

```
HTML

<input type="text" id="joe" name="joe" value="Joe">
```

With collection properties, `asp-for="CollectionProperty[23].Member"` generates the same name as `asp-for="CollectionProperty[i].Member"` when `i` has the value `23`.

When ASP.NET Core MVC calculates the value of `ModelExpression`, it inspects several sources, including `ModelState`. Consider `<input type="text" asp-for="Name">`. The calculated `value` attribute is the first non-null value from:

- `ModelState` entry with key "Name".
- Result of the expression `Model.Name`.

Navigating child properties

You can also navigate to child properties using the property path of the view model. Consider a more complex model class that contains a child `Address` property.

C#

```
public class AddressViewModel
{
    public string AddressLine1 { get; set; }
}
```

C#

```
public class RegisterAddressViewModel
{
    public string Email { get; set; }

    [DataType(DataType.Password)]
    public string Password { get; set; }

    public AddressViewModel Address { get; set; }
}
```

In the view, we bind to `Address.AddressLine1`:

CSHTML

```
@model RegisterAddressViewModel

<form asp-controller="Demo" asp-action="RegisterAddress" method="post">
    Email: <input asp-for="Email" /> <br />
    Password: <input asp-for="Password" /> <br />
    Address: <input asp-for="Address.AddressLine1" /> <br />
    <button type="submit">Register</button>
</form>
```

The following HTML is generated for `Address.AddressLine1`:

HTML

```
<input type="text" id="Address_AddressLine1" name="Address.AddressLine1" value="">
```

Expression names and Collections

Sample, a model containing an array of `Colors`:

C#

```
public class Person
{
    public List<string> Colors { get; set; }

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

The action method:

C#

```
public IActionResult Edit(int id, int colorIndex)
{
    ViewData["Index"] = colorIndex;
    return View(GetPerson(id));
}
```

The following Razor shows how you access a specific `Color` element:

CSHTML

```
@model Person
@{
```

```

    var index = (int)ViewData["index"];
}

<form asp-controller="ToDo" asp-action="Edit" method="post">
    @Html.EditorFor(m => m.Colors[index])
    <label asp-for="Age"></label>
    <input asp-for="Age" /><br />
    <button type="submit">Post</button>
</form>

```

The `Views/Shared/EditorTemplates/String.cshtml` template:

```

CSHTML

@model string

<label asp-for="@Model"></label>
<input asp-for="@Model" /> <br />

```

Sample using `List<T>`:

```

C#

public class ToDoItem
{
    public string Name { get; set; }

    public bool IsDone { get; set; }
}

```

The following Razor shows how to iterate over a collection:

```

CSHTML

@model List<ToDoItem>

<form asp-controller="ToDo" asp-action="Edit" method="post">
    <table>
        <tr> <th>Name</th> <th>Is Done</th> </tr>

        @for (int i = 0; i < Model.Count; i++)
        {
            <tr>
                @Html.EditorFor(model => model[i])
            </tr>
        }

    </table>
    <button type="submit">Save</button>
</form>

```

The `Views/Shared/EditorTemplates/ToDoItem.cshtml` template:

```

CSHTML

@model ToDoItem

<td>
    <label asp-for="@Model.Name"></label>
    @Html.DisplayFor(model => model.Name)
</td>
<td>
    <input asp-for="@Model.IsDone" />
</td>

@*
    This template replaces the following Razor which evaluates the indexer three times.
    <td>
        <label asp-for="@Model[i].Name"></label>
        @Html.DisplayFor(model => model[i].Name)
    </td>
    <td>
        <input asp-for="@Model[i].IsDone" />
    </td>
*@

```


`foreach` should be used if possible when the value is going to be used in an `asp-for` or `Html.DisplayFor` equivalent context. In general, `for` is better than `foreach` (if the scenario allows it) because it doesn't need to allocate an enumerator; however, evaluating an indexer in a LINQ expression can be expensive and should be minimized.

📌 Note

The commented sample code above shows how you would replace the lambda expression with the `@` operator to access each `ToDoItem` in the list.

The Textarea Tag Helper

The `Textarea Tag Helper` tag helper is similar to the `Input Tag Helper`.

- Generates the `id` and `name` attributes, and the data validation attributes from the model for a `<textarea>` [↗](#) element.
- Provides strong typing.
- HTML Helper alternative: `Html.TextAreaFor`

Sample:

```
C#  
  
using System.ComponentModel.DataAnnotations;  
  
namespace FormsTagHelper.ViewModels  
{  
    public class DescriptionViewModel  
    {  
        [MinLength(5)]  
        [MaxLength(1024)]  
        public string Description { get; set; }  
    }  
}
```

```
CSHTML  
  
@model DescriptionViewModel  
  
<form asp-controller="Demo" asp-action="RegisterTextArea" method="post">  
    <textarea asp-for="Description"></textarea>  
    <button type="submit">Test</button>  
</form>
```

The following HTML is generated:

```
HTML  
  
<form method="post" action="/Demo/RegisterTextArea">  
    <textarea data-val="true"  
        data-val-maxlength="The field Description must be a string or array type with a maximum length of &#x27;1024&#x27;."  
        data-val-maxlength-max="1024"  
        data-val-minlength="The field Description must be a string or array type with a minimum length of &#x27;5&#x27;."  
        data-val-minlength-min="5"  
        id="Description" name="Description">  
    </textarea>  
    <button type="submit">Test</button>  
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>">  
</form>
```

The Label Tag Helper

- Generates the label caption and `for` attribute on a `<label>` [↗](#) element for an expression name
- HTML Helper alternative: `Html.LabelFor`.

The `Label Tag Helper` provides the following benefits over a pure HTML label element:

- You automatically get the descriptive label value from the `Display` attribute. The intended display name might change over time, and the combination of `Display` attribute and Label Tag Helper will apply the `Display` everywhere it's used.
- Less markup in source code
- Strong typing with the model property.

Sample:

C#

```
using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class SimpleViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email Address")]
        public string Email { get; set; }
    }
}
```

CSHTML

```
@model SimpleViewModel

<form asp-controller="Demo" asp-action="RegisterLabel" method="post">
    <label asp-for="Email"></label>
    <input asp-for="Email" /> <br />
</form>
```

The following HTML is generated for the `<label>` element:

HTML

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

The Label Tag Helper generated the `for` attribute value of "Email", which is the ID associated with the `<input>` element. The Tag Helpers generate consistent `id` and `for` elements so they can be correctly associated. The caption in this sample comes from the `Display` attribute. If the model didn't contain a `Display` attribute, the caption would be the property name of the expression. To override the default caption, add a caption inside the label tag.

The Validation Tag Helpers

There are two Validation Tag Helpers. The `Validation Message Tag Helper` (which displays a validation message for a single property on your model), and the `Validation Summary Tag Helper` (which displays a summary of validation errors). The `Input Tag Helper` adds HTML5 client side validation attributes to input elements based on data annotation attributes on your model classes. Validation is also performed on the server. The Validation Tag Helper displays these error messages when a validation error occurs.

The Validation Message Tag Helper

- Adds the [HTML5](#) `data-valmsg-for="property"` attribute to the [span](#) element, which attaches the validation error messages on the input field of the specified model property. When a client side validation error occurs, [jQuery](#) displays the error message in the `` element.
- Validation also takes place on the server. Clients may have JavaScript disabled and some validation can only be done on the server side.
- HTML Helper alternative: `Html.ValidationMessageFor`

The `Validation Message Tag Helper` is used with the `asp-validation-for` attribute on an HTML [span](#) element.

CSHTML

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

The Validation Message Tag Helper will generate the following HTML:

HTML

```
<span class="field-validation-valid" data-valmsg-for="Email" data-valmsg-replace="true"></span>
```

You generally use the Validation Message Tag Helper after an Input Tag Helper for the same property. Doing so displays any validation error messages near the input that caused the error.

Note

You must have a view with the correct JavaScript and jQuery script references in place for client side validation. See Model Validation for more information.

When a server side validation error occurs (for example when you have custom server side validation or client-side validation is disabled), MVC places that error message as the body of the element.

HTML

```
<span class="field-validation-error" data-valmsg-for="Email" data-valmsg-replace="true">
    The Email Address field is required.
</span>
```

The Validation Summary Tag Helper

- Targets <div> elements with the asp-validation-summary attribute
- HTML Helper alternative: @Html.ValidationSummary

The Validation Summary Tag Helper is used to display a summary of validation messages. The asp-validation-summary attribute value can be any of the following:

Expand table

| asp-validation-summary | Validation messages displayed |
|------------------------|-------------------------------|
| All | Property and model level |
| ModelOnly | Model |
| None | None |

Sample

In the following example, the data model has DataAnnotation attributes, which generates validation error messages on the <input> element. When a validation error occurs, the Validation Tag Helper displays the error message:

C#

```
using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class RegisterViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email Address")]
        public string Email { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }
    }
}
```

C#HTML

```
@model RegisterViewModel

<form asp-controller="Demo" asp-action="RegisterValidation" method="post">
    <div asp-validation-summary="ModelOnly"></div>
    Email:  <input asp-for="Email" /> <br />
    <span asp-validation-for="Email"></span><br />
    Password: <input asp-for="Password" /><br />
    <span asp-validation-for="Password"></span><br />
    <button type="submit">Register</button>
</form>
```

The generated HTML (when the model is valid):

HTML

```
<form action="/DemoReg/Register" method="post">
  Email:  <input name="Email" id="Email" type="email" value=""
    data-val-required="The Email field is required."
    data-val-email="The Email field is not a valid email address."
    data-val="true"><br>
  <span class="field-validation-valid" data-valmsg-replace="true"
    data-valmsg-for="Email"></span><br>
  Password: <input name="Password" id="Password" type="password"
    data-val-required="The Password field is required." data-val="true"><br>
  <span class="field-validation-valid" data-valmsg-replace="true"
    data-valmsg-for="Password"></span><br>
  <button type="submit">Register</button>
  <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>">
</form>
```

The Select Tag Helper

- Generates [select](#) and associated [option](#) elements for properties of your model.
- Has an HTML Helper alternative `Html.DropDownListFor` and `Html.ListBoxFor`

The `Select Tag Helper` `asp-for` specifies the model property name for the [select](#) element and `asp-items` specifies the [option](#) elements. For example:

C#HTML

```
<select asp-for="Country" asp-items="Model.Countries"></select>
```

Sample:

C#

```
using Microsoft.AspNetCore.Mvc.Rendering;
using System.Collections.Generic;

namespace FormsTagHelper.ViewModels
{
    public class CountryViewModel
    {
        public string Country { get; set; }

        public List<SelectListItem> Countries { get; } = new List<SelectListItem>
        {
            new SelectListItem { Value = "MX", Text = "Mexico" },
            new SelectListItem { Value = "CA", Text = "Canada" },
            new SelectListItem { Value = "US", Text = "USA" },
        };
    }
}
```

The `Index` method initializes the `CountryViewModel`, sets the selected country and passes it to the `Index` view.

C#

```
public IActionResult Index()
{
    var model = new CountryViewModel();
```

```
model.Country = "CA";  
return View(model);  
}
```

The HTTP POST `Index` method displays the selection:

C#

```
[HttpPost]  
[ValidateAntiForgeryToken]  
public IActionResult Index(CountryViewModel model)  
{  
    if (ModelState.IsValid)  
    {  
        var msg = model.Country + " selected";  
        return RedirectToAction("IndexSuccess", new { message = msg });  
    }  
  
    // If we got this far, something failed; redisplay form.  
    return View(model);  
}
```

The `Index` view:

CSHTML

```
@model CountryViewModel  
  
<form asp-controller="Home" asp-action="Index" method="post">  
    <select asp-for="Country" asp-items="Model.Countries"></select>  
    <br /><button type="submit">Register</button>  
</form>
```

Which generates the following HTML (with "CA" selected):

HTML

```
<form method="post" action="/">  
    <select id="Country" name="Country">  
        <option value="MX">Mexico</option>  
        <option selected="selected" value="CA">Canada</option>  
        <option value="US">USA</option>  
    </select>  
    <br /><button type="submit">Register</button>  
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>">  
</form>
```

Note

We don't recommend using `ViewBag` or `ViewData` with the Select Tag Helper. A view model is more robust at providing MVC metadata and generally less problematic.

The `asp-for` attribute value is a special case and doesn't require a `Model` prefix, the other Tag Helper attributes do (such as `asp-items`)

CSHTML

```
<select asp-for="Country" asp-items="Model.Countries"></select>
```

Enum binding

It's often convenient to use `<select>` with an `enum` property and generate the `SelectListItem` elements from the `enum` values.

Sample:

C#

```
public class CountryEnumViewModel  
{
```

```
public CountryEnum EnumCountry { get; set; }  
}
```

C#

```
using System.ComponentModel.DataAnnotations;  
  
namespace FormsTagHelper.ViewModels  
{  
    public enum CountryEnum  
    {  
        [Display(Name = "United Mexican States")]  
        Mexico,  
        [Display(Name = "United States of America")]  
        USA,  
        Canada,  
        France,  
        Germany,  
        Spain  
    }  
}
```

The `GetEnumSelectList` method generates a `SelectList` object for an enum.

CSHTML

```
@model CountryEnumViewModel  
  
<form asp-controller="Home" asp-action="IndexEnum" method="post">  
    <select asp-for="EnumCountry"  
        asp-items="Html.GetEnumSelectList<CountryEnum>()">  
    </select>  
    <br /><button type="submit">Register</button>  
</form>
```

You can mark your enumerator list with the `Display` attribute to get a richer UI:

C#

```
using System.ComponentModel.DataAnnotations;  
  
namespace FormsTagHelper.ViewModels  
{  
    public enum CountryEnum  
    {  
        [Display(Name = "United Mexican States")]  
        Mexico,  
        [Display(Name = "United States of America")]  
        USA,  
        Canada,  
        France,  
        Germany,  
        Spain  
    }  
}
```

The following HTML is generated:

HTML

```
<form method="post" action="/Home/IndexEnum">  
    <select data-val="true" data-val-required="The EnumCountry field is required."  
        id="EnumCountry" name="EnumCountry">  
        <option value="0">United Mexican States</option>  
        <option value="1">United States of America</option>  
        <option value="2">Canada</option>  
        <option value="3">France</option>  
        <option value="4">Germany</option>  
        <option selected="selected" value="5">Spain</option>  
    </select>  
    <br /><button type="submit">Register</button>  
    <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>">  
</form>
```

Option Group

The HTML `<optgroup>` [↗](#) element is generated when the view model contains one or more `SelectListGroup` objects.

The `CountryViewModelGroup` groups the `SelectListItem` elements into the "North America" and "Europe" groups:

```
C#

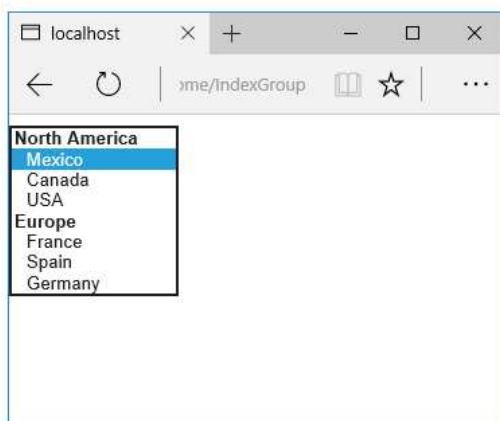
public class CountryViewModelGroup
{
    public CountryViewModelGroup()
    {
        var NorthAmericaGroup = new SelectListGroup { Name = "North America" };
        var EuropeGroup = new SelectListGroup { Name = "Europe" };

        Countries = new List<SelectListItem>
        {
            new SelectListItem
            {
                Value = "MEX",
                Text = "Mexico",
                Group = NorthAmericaGroup
            },
            new SelectListItem
            {
                Value = "CAN",
                Text = "Canada",
                Group = NorthAmericaGroup
            },
            new SelectListItem
            {
                Value = "US",
                Text = "USA",
                Group = NorthAmericaGroup
            },
            new SelectListItem
            {
                Value = "FR",
                Text = "France",
                Group = EuropeGroup
            },
            new SelectListItem
            {
                Value = "ES",
                Text = "Spain",
                Group = EuropeGroup
            },
            new SelectListItem
            {
                Value = "DE",
                Text = "Germany",
                Group = EuropeGroup
            }
        };
    }

    public string Country { get; set; }

    public List<SelectListItem> Countries { get; }
}
```

The two groups are shown below:



The generated HTML:

HTML

```
<form method="post" action="/Home/IndexGroup">
  <select id="Country" name="Country">
    <optgroup label="North America">
      <option value="MEX">Mexico</option>
      <option value="CAN">Canada</option>
      <option value="US">USA</option>
    </optgroup>
    <optgroup label="Europe">
      <option value="FR">France</option>
      <option value="ES">Spain</option>
      <option value="DE">Germany</option>
    </optgroup>
  </select>
  <br /><button type="submit">Register</button>
  <input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>">
</form>
```

Multiple select

The Select Tag Helper will automatically generate the `multiple = "multiple"` attribute if the property specified in the `asp-for` attribute is an `IEnumerable`. For example, given the following model:

C#

```
using Microsoft.AspNetCore.Mvc.Rendering;
using System.Collections.Generic;

namespace FormsTagHelper.ViewModels
{
    public class CountryViewModelIEnumerable
    {
        public IEnumerable<string> CountryCodes { get; set; }

        public List<SelectListItem> Countries { get; } = new List<SelectListItem>
        {
            new SelectListItem { Value = "MX", Text = "Mexico" },
            new SelectListItem { Value = "CA", Text = "Canada" },
            new SelectListItem { Value = "US", Text = "USA" },
            new SelectListItem { Value = "FR", Text = "France" },
            new SelectListItem { Value = "ES", Text = "Spain" },
            new SelectListItem { Value = "DE", Text = "Germany" }
        };
    }
}
```

With the following view:

CSHTML

```
@model CountryViewModelIEnumerable

<form asp-controller="Home" asp-action="IndexMultiSelect" method="post">
  <select asp-for="CountryCodes" asp-items="Model.Countries"></select>
  <br /><button type="submit">Register</button>
</form>
```

Generates the following HTML:

HTML

```
<form method="post" action="/Home/IndexMultiSelect">
  <select id="CountryCodes"
    multiple="multiple"
    name="CountryCodes"><option value="MX">Mexico</option>
  <option value="CA">Canada</option>
  <option value="US">USA</option>
  <option value="FR">France</option>
  <option value="ES">Spain</option>
  <option value="DE">Germany</option>
</select>
  <br /><button type="submit">Register</button>
```



```
<input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>">
</form>
```

No selection

If you find yourself using the "not specified" option in multiple pages, you can create a template to eliminate repeating the HTML:

C#HTML

```
@model CountryViewModel

<form asp-controller="Home" asp-action="IndexEmpty" method="post">
    @Html.EditorForModel()
    <br /><button type="submit">Register</button>
</form>
```

The `Views/Shared/EditorTemplates/CountryViewModel.cshtml` template:

C#HTML

```
@model CountryViewModel

<select asp-for="Country" asp-items="Model.Countries">
    <option value="">--none--</option>
</select>
```

Adding HTML `<option>` elements isn't limited to the *No selection* case. For example, the following view and action method will generate HTML similar to the code above:

C#

```
public IActionResult IndexNone()
{
    var model = new CountryViewModel();
    model.Countries.Insert(0, new SelectListItem("<none>", ""));
    return View(model);
}
```

C#HTML

```
@model CountryViewModel

<form asp-controller="Home" asp-action="IndexEmpty" method="post">
    <select asp-for="Country">
        <option value="">&lt;none&gt;</option>
        <option value="MX">Mexico</option>
        <option value="CA">Canada</option>
        <option value="US">USA</option>
    </select>
    <br /><button type="submit">Register</button>
</form>
```

The correct `<option>` element will be selected (contain the `selected="selected"` attribute) depending on the current `Country` value.

C#



```
public IActionResult IndexOption(int id)
{
    var model = new CountryViewModel();
    model.Country = "CA";
    return View(model);
}
```

HTML

```
<form method="post" action="/Home/IndexEmpty">
    <select id="Country" name="Country">
        <option value="">&lt;none&gt;</option>
        <option value="MX">Mexico</option>
        <option value="CA" selected="selected">Canada</option>
        <option value="US">USA</option>
```

```
</select>
<br /><button type="submit">Register</button>
<input name="__RequestVerificationToken" type="hidden" value="<removed for brevity>">
</form>
```

Additional resources

- [Tag Helpers in ASP.NET Core](#)
- [HTML Form element](#) 
- [Request Verification Token](#)
- [Model Binding in ASP.NET Core](#)
- [Model validation in ASP.NET Core MVC](#)
- [IAttributeAdapter Interface](#)
- [Code snippets for this document](#) 

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see [our contributor guide](#).



ASP.NET Core feedback

ASP.NET Core is an open source project. Select a link to provide feedback:

 [Open a documentation issue](#)

 [Provide product feedback](#)