

Introduction

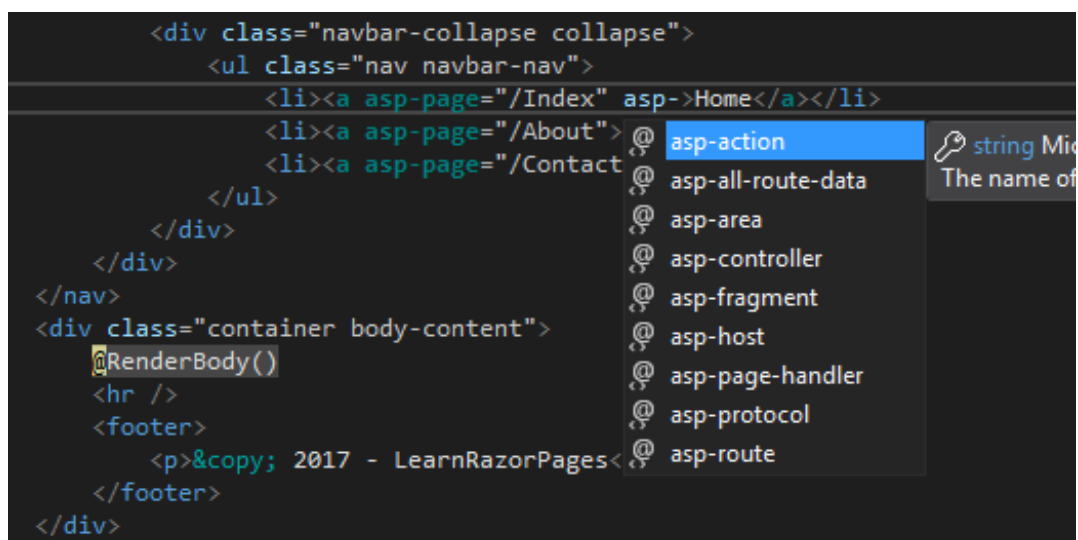
 learnrazorpages.com/razor-pages/tag-helpers

Your guide to using ASP.NET Core Razor Pages

Tag helpers are reusable components for automating the generation of HTML in Razor Pages. Tag helpers target specific HTML tags. The ASP.NET Core framework includes a number of predefined tag helpers targeting many commonly used HTML elements as well as some custom tags:

The Tag helpers used in Razor Pages were introduced as part of ASP.NET MVC Core and are found in the `Microsoft.AspNetCore.Mvc.TagHelpers` package which is included as part of the `Microsoft.AspNetCore.All` meta-package. It is also possible to create your own custom tag helpers to automate the generation of HTML in a limitless range of scenarios.

The following image illustrates an Anchor tag helper, which targets the HTML anchor `<a>` tag :



Each tag helper augments the target element with additional attributes, prefixed with `asp-`. In the image above, you can see that the `asp-page` attribute in the tag has a value applied, and additional attributes are shown by IntelliSense (in IDEs that provide this feature). Some of the attributes are specific to Razor Pages and some are specific to MVC. Others are relevant to both development platforms.

Enabling Tag Helpers [link](#)

Tag helpers are an opt-in feature. They are not available to the page by default. They are enabled by adding an `@addTagHelper` directive to the page, or more usually to a `ViewImports.cshtml` file:

1. `@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers`

The `@addTagHelper` directive is followed by a wildcard character (`*`) to specify that all tag helpers found in the specified assembly should be used, and then the name of the assembly containing the tag helpers is provided. The name of the assembly is the name of your Razor Pages project in most cases, unless you are defining your tag helpers in a separate project. If you want to enable tag helpers defined in this site, which has a `.csproj` file named `LearnRazorPages.csproj`, you would do so like this:

1. `@addTagHelper *, LearnRazorPages`

Note: The value provided to the `@addTagHelper` directive is not enclosed in quotes. This requirement was removed when ASP.NET Core was at Release Candidate 2. However, if you prefer, you can still surround values in quotes:

1. `@addTagHelper "*", Microsoft.AspNetCore.Mvc.TagHelpers"`

Selective tag processing [link](#)

Once a tag helper has been enabled, it will process every instance of the tag that it targets. That may not be desirable, especially so where tags don't feature special attributes that need to be processed. It is possible to opt in or out of tag processing selectively. You can use the `@addTagHelper` and `@removeTagHelper` directives to opt in or opt out of processing all tags of a certain type. Rather than pass the wildcard character to the `@addtagHelper` directive, you can pass the name(s) of the tag helpers that you want to enable:

1. `@addTagHelper "Microsoft.AspNetCore.Mvc.TagHelpers.AnchorTagHelper, Microsoft.AspNetCore.Mvc.TagHelpers"`

The only tag helper that is enabled in the previous snippet is the `AnchorTagHelper`. This approach is suitable if you only want to enable a small selection of tag helpers. If you want to enable most of the tag helpers in a library, you can use the `@removeTagHelper` directive to filter out tag helpers having enabled all of them. Here's how you would disable the `AnchorTagHelper` using this method:

1. `@addTagHelper "*", Microsoft.AspNetCore.Mvc.TagHelpers"`
2. `@removeTagHelper "Microsoft.AspNetCore.Mvc.TagHelpers.AnchorTagHelper, Microsoft.AspNetCore.Mvc.TagHelpers"`

You can opt individual tags out of processing by placing the `!` prefix just prior to the tag name. The following example illustrates how that is applied to an anchor tag to prevent it being processed unnecessarily:

1. `<!a href="https://www.learnrazorpages.com">Learn Razor Pages</!a>`

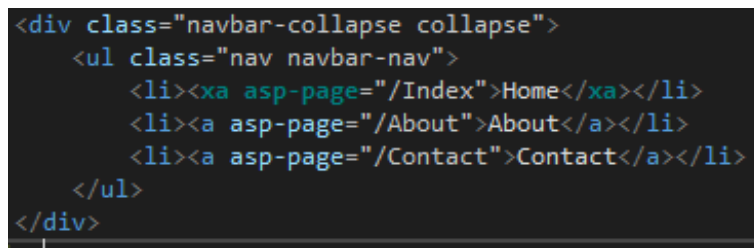
The prefix is placed in both the start and end tag. Any tag without the `!` prefix will be processed by an associated tag helper. The alternative option is to opt specific tags in to processing at parse time. You achieve this by registering a custom prefix via the `@tagHelperPrefix` directive and then applying your chosen prefix to tags you want to take part in processing. You can register your prefix in the `_ViewImports.cshtml` file, where you enabled tag helper processing:

1. `@tagHelperPrefix x`

You can use pretty much any string you like as a prefix. Then you apply it to both the start and end tag, just like the `!` prefix:

1. `<xa asp-page="/Index">Home</xa>`

Only those tags that feature the prefix will be processed. The image below illustrates how Visual Studio shows enabled tag helpers with a different font:



```
<div class="navbar-collapse collapse">
  <ul class="nav navbar-nav">
    <li><xa asp-page="/Index">Home</xa></li>
    <li><a asp-page="/About">About</a></li>
    <li><a asp-page="/Contact">Contact</a></li>
  </ul>
</div>
```

For the sake of clarity, most developers are likely to use punctuation to separate the prefix from the tag name, for example:

1. `@tagHelperPrefix x:`

1. `<x:a asp-page="/Index">Home</x:a>`

This should reduce any visual confusion especially for designers when they look at the HTML content.

The Anchor tag helper

 [learnrazorpages.com/razor-pages/tag-helpers/anchor-tag-helper](https://www.learnrazorpages.com/razor-pages/tag-helpers/anchor-tag-helper)

Your guide to using ASP.NET Core Razor Pages

The anchor tag helper targets the HTML anchor (`<a>`) tag which is used to generate relative links from the values passed to various custom attributes. It can also be used to generate absolute URLs to external resources.

The anchor tag helper's role is to generate an `href` attribute from the parameter values passed to its custom attributes. Therefore, if you try to add an `href` attribute to the anchor tag helper as well as values to the custom attributes, an exception will be raised:

InvalidOperationException: Cannot override the 'href' attribute for `<a>` . An `<a>` with a specified 'href' must not have attributes starting with 'asp-route-' or an 'asp-action', 'asp-controller', 'asp-area', 'asp-route', 'asp-protocol', 'asp-host', 'asp-fragment', 'asp-page' or 'asp-page-handler' attribute.

If a route cannot be matched from the values passed to the taghelper attributes, the output for the `href` attribute will silently fall back to an empty string.

Attribute	Description
<code>action</code>	The name of the action method on an MVC controller
<code>all-route-data</code> ¹	Multiple route parameter values
<code>area</code>	The name of the Area
<code>controller</code>	The name of the MVC controller
<code>fragment</code> ²	The fragment in the URL
<code>host</code>	The domain
<code>page</code> ³	The Razor page to link to
<code>page-handler</code> ⁴	The Razor page handler method to invoke

Attribute	Description
<code>protocol</code>	The protocol (http, https, ftp etc)
<code>route</code> ⁵	The name of the route
<code>route-</code> ⁶	A single route parameter value

Notes [link](#)

1. If the target URL includes multiple route parameters, their values can be packaged as a `Dictionary<string, string>` and passed to the `all-route-data` parameter:

```
1. @{
2.     var d = new Dictionary<string, string>
3.     {
4.         { "key1", "value1" },
5.         { "key2", "value2" }
6.     };
7. }
8. <a asp-all-route-data="d">Click</a>
```

If the route has parameters defined, the anchor tag helper will output the values as URL segments: `Click` . If it doesn't, the route parameters will be appended to the URL as query string values: `Click` .

2. The fragment is the value after a hash or pound sign (`#`) in a URL used to identify a named portion of a document. The "Notes" heading above has an identity value of "notes" and can be referenced in a URL using the anchor tag helper like this:

```
1. <a asp-fragment="notes">Click</a>
```

producing this HTML: `Click`

3. The name of the Razor page to link to must be provided without the file extension:

1. `<a asp-page="page">Click`

If no valid page name is specified, the tag helper will generate a link to the current page. If you want to generate a link to the default page in a folder, you must include the default page's file name:

1. `<a asp-page="/folder/index">Folder`

This renders as `Folder`

The taghelper will generate an `href` attribute whose value will be taken from the route template of the specified page.

4. The page handler method name will appear as a query string value unless it has been included as a route parameter for the target page.

5. Razor pages doesn't support named routes. This parameter will only be used for MVC routes.

6. The `route-` attribute enables you to specify the value for a single route data parameter. The parameter name is added after the hyphen. Here, the route parameter name is "key1":

```
<a asp-route-key1="value1">Click</a>
```

This renders as

```
<a href="/Page/value1">Click</a>
```

if the `key1` parameter is defined as part of the page's route template, or

```
<a href="/Page?key1=value1">Click</a>
```

if not.

Ambient Route Values ^{link}

Route Data values for the current request are considered *ambient* values in ASP.NET Core 2.1 and earlier*. This means that they do not necessarily need to be specified when generating links from the current page to another that requires the same Route Data value. For example, you may provide a page that shows the details of an order that has an `id` as a route data parameter: `"{id:int}"`.

Within the *Details* page, you want to provide a link to an *Edit* page, which also accepts a route data parameter named `id`. There is no need to specify the `asp-route-id` attribute to the anchor tag helper because the ambient value from the current request context will be used by default. Therefore the following is sufficient to generate a link to `/edit/3`:

```
<a asp-page="edit">Edit</a>
```

In this case, you only need to specify a value for `asp-route-*` if you want to override the ambient value.

If you override an ambient value by explicitly providing a value, all subsequent ambient values will be ignored. The template `"{pageno=1}/{sortby=Id}"` specifies two parameters, each with a default value. If you override the `pageno` value e.g. `<a asp-page="list" asp-route-pageno="2">2`, the `sortby` value is omitted from the generated URL (along with any subsequent parameters) so that the generated URL is `list/2`. However, if you override the `sortby` value, the `pageno` value is retained (along with any other previous parameters).

***Note:** This behaviour changed in ASP.NET Core 2.2. Ambient route values are no longer reused when the destination page is different to the source page.

Routing Options *link*

The anchor tag helper will generate URLs with the page name capitalised when you pass a value to the `page` attribute e.g.

```
1. <a asp-page="page">Click</a>
```

becomes

```
1. <a href="/Page">Click</a>
```

You can configure `RouteOptions` if you prefer the generated URL to be all lower case:

```
1. public void ConfigureServices(IServiceCollection services)
2. {
3.     services.AddMvc();
4.     services.Configure<RouteOptions>(options =>
5.     {
6.         options.LowercaseUrls = true;
7.     });
8. }
```

Another option, `AppendTrailingSlash` will result in a trailing slash being appended to the page name in every case:

```
1. public void ConfigureServices(IServiceCollection services)
2. {
3.     services.AddMvc();
4.     services.Configure<RouteOptions>(options =>
5.     {
6.         options.AppendTrailingSlash = true;
7.     });
8. }
```

With both options enabled, the resulting HTML looks like this:

```
1. <a href="/page/">Click</a>
```


The Cache Tag Helper

 learnrazorpages.com/razor-pages/tag-helpers/cache-tag-helper

Your guide to using ASP.NET Core Razor Pages

The cache tag helper enables you to cache regions of a Razor page in memory on the server. Typical uses for this helper are for View Components or partial views that require relatively expensive database or web service calls, but where the content doesn't change very often. This type of caching is primarily used to improve a website's performance.

Unlike most other tag helpers, the Cache tag helper doesn't target a standard HTML element. It targets the `<cache>` tag, which is not rendered to the browser and doesn't appear in the HTML source either.

The cache tag helper uses an in-memory-based store for content. This is volatile and can be cleared unexpectedly for any number of reasons, including app pool recycling, server restarts, low memory pressure and so on. The cache tag helper is not intended to be used for reliable long term storage.

Attribute	Description
<code>enabled</code> ¹	Used to specify if this tag is enabled or not
<code>expires-after</code> ²	Specifies the period of time after which the cached item should expire
<code>expires-on</code> ³	Specifies the time at which the cached entry should expire
<code>expires-sliding</code> ⁴	The period of time after the last access that the item should expire
<code>priority</code> ⁵	Specifies the <code>CacheItemPriority</code> value of the cached item
<code>vary-by</code> ⁶	Used to specify the parameters that determine whether different versions of the same content should be cached

Notes [link](#)

1. The cache tag helper is enabled by default. You might want to conditionally disable it under certain circumstances. The `enabled` attribute facilitates this. The cached item in this example is disabled on Sundays:

```
1. <cache enabled="DateTime.Now.DayOfWeek != DayOfWeek.Sunday">@DateTime.Now</cache>
```

2. If you don't provide values for any of the `expires-*` attributes, your item will be cached without an expiration date, meaning that it will only expire when the memory store is cleared. The `expires-after` attribute takes a `TimeSpan` value that represents the period of time that the item should be stored in the cache. This example shows the item being stored for 1 hour:

```
1. <cache expires-after="TimeSpan.FromHours(1)">@DateTimeNow</cache>
```

3. The `expires-on` attribute takes a `DateTimeOffset` value that specifies the absolute expiry time of an item. This item is set to expire at 8:15 am on June 14th 2017 UTC:

```
1. <cache expires-on="new DateTimeOffset(new DateTime(2017, 6, 14, 8, 15, 0))">@DateTime.Now</cache>
```

4. An item can be set to expire after a specified period of inactivity. This is known as "sliding expiration". The `expires-sliding` attribute takes a `TimeSpan` value. In this example, the item is set to expire 20 minutes after the last time the page was requested:

```
1. <cache expires-sliding="TimeSpan.FromMinutes(20)">@DateTime.Now</cache>
```

As each request is made for the cached item, the expiration time is reset to 20 minutes after the request.

5. Items in the cache can have a priority applied to them, determining how they are dealt with when memory pressure results in items being cleared from the cache before their pre-ordained expiry time. The `priority` attribute takes a `CacheItemPriority` enum value that specifies the relative priority of items in the cache. The available values are

- High
- Low
- Normal
- NeverRemove

Items with `High` priority will be the last to be removed. Items set as `NeverRemove` will remain in the cache. You should exercise caution when applying `NeverRemove` as there is a danger of overflowing the cache with more data than it can handle. This is how to set an item with `High` priority:

1. `<cache priority="CacheItemPriority.High">@DateTime.Now</cache>`

6. You can cache multiple versions of the same content based on different criteria. The `vary-by` attribute provides a mechanism for you to specify the criteria to be used to determine whether to store another version of existing cached content.

The attribute has a number of preset options:

- `vary-by-cookie`
- `vary-by-header`
- `vary-by-query`
- `vary-by-route`
- `vary-by-user`

Cookie

The values of cookies can be taken into consideration when caching different versions of content. The `vary-by-cookie` option takes a comma-separated string representing the names of the cookies:

1. `<cache vary-by-cookie="AppCookie1, AppCookie2">@DateTime.Now</cache>`

You can store multiple versions of content based on differing request header values. Multiple headers can be applied as a comma-separated string. This example stores different versions of the cached content based on the user's preferred language:

1. `<cache vary-by-header="Accept-Language">@DateTime.Now</cache>`

Query

The `vary-by-query` option enables the use of query string parameters as the criteria for caching different versions. Query string parameter names are supplied in a comma-separated list:

1. `<cache vary-by-query="id">@DateTime.Now</cache>`

Route

You can have route data parameters taken into account by using the `vary-by-route` option. This attribute also accepts a comma-separated string of route parameter names as a value:

1. `<cache vary-by-route="key1, key2">@DateTime.Now</cache>`

User

The final built-in option enables you to specify that each user gets their own version of cached content. The `vary-by-user` attribute requires a `bool`, which should be set to `true`:

1. `<cache vary-by-user="true">@DateTime.Now</cache>`

String

Finally, you can provide an arbitrary string value as a parameter if you want to use a value that isn't exposed by one of the preset options:

1. `<cache vary-by="@Model.Id">@DateTime.Now</cache>`

You can use any combination of the `vary-by` attributes that you need.

The Environment Tag Helper



learnrazorpages.com/razor-pages/tag-helpers/environment-tag-helper

Your guide to using ASP.NET Core Razor Pages

The environment tag helper supports rendering different content dependent on the current value of the `ASPNETCORE_ENVIRONMENT` environment variable for the application.

Attribute	Description
<code>names</code>	The name(s) of the environment(s) for which the content should be rendered
<code>include</code>	The name(s) of the environment(s) for which the content should be rendered
<code>exclude</code>	The name(s) of the environment(s) for which the content should not be rendered

Notes [link](#)

The most common use case for the environment tag helper is to include the full version of CSS or JavaScript files when the application is in a development stage, and the bundled and minified versions when the application is in other environments.

```
1. <environment names="Development">
2.     <link rel="stylesheet" href="~/css/style1.css" />
3.     <link rel="stylesheet" href="~/css/style2.css" />
4. </environment>
5. <environment names="Staging, Test, Production">
6.     <link rel="stylesheet" href="~/css/style.min.css" />
7. </environment>
```

If an environment name is part of an `exclude` list, the content will not be rendered under any circumstances for that environment. Inclusion in the `exclude` list overrides inclusion in the `include` list or the `names` list.

The Formaction Tag Helper

 learnrazorpages.com/razor-pages/tag-helpers/form-action-tag-helper

Your guide to using ASP.NET Core Razor Pages

The formaction tag helper adds a "formaction" attribute to the target element with a value generated from the parameters passed to the various custom attributes.

The formaction tag helper targets two elements:

- Button
- Input with a `type` attribute set to `image` or `submit`

The formaction tag helper is an example where the name of the tag helper doesn't follow the convention that matches tag helper class names with the name of the target element.

The `formaction` attribute specifies where a form is to be posted. It overrides the form's `action` attribute. It is new to HTML5 and is not supported in IE9 or earlier.

Attribute	Description
<code>action</code>	The name of the action method on an MVC controller
<code>all-route-data</code> ¹	Multiple route parameter values
<code>area</code>	The name of the MVC area
<code>controller</code>	The name of the MVC controller
<code>fragment</code> ²	The fragment in the URL
<code>host</code>	The domain
<code>page</code> ³	The Razor page to link to
<code>page-handler</code> ⁴	The Razor page handler method to invoke
<code>protocol</code>	The protocol (http, https, ftp etc)
<code>route</code> ⁵	The name of the route

Attribute	Description
<code>route-</code> ⁶	A single route parameter value

Notes [link](#)

1. If the target URL includes multiple route parameters, their values can be packaged as a `Dictionary<string, string>` and passed to the `all-route-data` parameter:

```

1. @{
2.     var d = new Dictionary<string, string>
3.     {
4.         { "key1", "value1" },
5.         { "key2", "value2" }
6.     };
7. }
8. <button asp-all-route-data="d">Submit</button>

```

If the route has parameters defined, the formaction tag helper will output the values as URL segments: `<button formaction="/Page/value1/value2">Submit</button>` . If it doesn't, the route parameters will be appended to the URL as query string values: `<button formaction="/Page?key1=value1&key2=value2">Submit</button>` .

2. The fragment is the value after a hash or pound sign (#) in a URL used to identify a named portion of a document. The "Notes" heading above has an identity value of "notes" and can be referenced in a URL using the formaction tag helper like this:

```
1. <button asp-fragment="notes">Submit</button>
```

producing this HTML: `<button formaction="/Page#notes">Submit</button>` . It should be noted that fragments have no influence on the submission of a form.

3. The name of the Razor page to link to must be provided without the file extension:

```
1. <button asp-page="page">Submit</button>
```

If no page name is specified, the tag helper will generate a link to the current page.

4. The page handler method name will appear as a query string value unless it has been included as a route parameter for the target page.
5. Razor pages doesn't support named routes. This parameter will only be used for MVC routes.
6. The `route-` parameter enables you to specify the value for a single route value. The route parameter name is added after the hyphen. Here, the route parameter name is "key1":

```
<button asp-route-key1="value1">Submit</button>
```

This renders as

```
<button formaction="/Page/value1">Submit</button>
```

if the `key1` parameter is defined as part of the page's route template, or

```
<button formaction="/Page?key1=value1">Submit</button>
```

if not.

The Form Tag Helper

 [learnrazorpages.com/razor-pages/tag-helpers/form-tag-helper](https://www.learnrazorpages.com/razor-pages/tag-helpers/form-tag-helper)

Your guide to using ASP.NET Core Razor Pages

The form tag helper renders an `action` attribute within a form element. It also includes an anti-forgery token for request verification. If a `method` attribute is not specified in the form element, the form tag helper will render one with a value of `post`.

The form tag helper's primary role is to generate an `action` attribute from the parameters passed to its custom attributes. Therefore, if you try to provide an `action` attribute to the form tag helper in addition to the custom attributes, an exception will be raised.

Attribute	Description
<code>action</code>	The name of the action method on an MVC controller
<code>all-route-data</code> ¹	Multiple route parameter values
<code>antiforgery</code> ⁷	Specifies whether an anti-forgery token is rendered.
<code>area</code>	The name of the MVC area
<code>controller</code>	The name of the MVC controller
<code>fragment</code> ²	The fragment in the URL
<code>host</code>	The domain
<code>page</code> ³	The Razor page to link to
<code>page-handler</code> ⁴	The Razor page handler method to invoke
<code>protocol</code>	The protocol (http, https, ftp etc)
<code>route</code> ⁵	The name of the route
<code>route-</code> ⁶	A single route parameter value

Notes [link](#)

1. If the target URL for the `action` includes multiple route parameters, their values can be packaged as a `Dictionary<string, string>` and passed to the `all-route-data` parameter:

```

1. @{
2.     var d = new Dictionary<string, string>
3.     {
4.         { "key1", "value1" },
5.         { "key2", "value2" }
6.     };
7. }
8. <form asp-all-route-data="d">...</form>

```

If the route has parameters defined, the form tag helper will output the values as URL segments: `<form action="/Page/value1/value2">...</form>` . If it doesn't, the route parameters will be appended to the URL as query string values: `<form action="/Page?key1=value1&key2=value2" method="post">...</form>` .

2. The fragment is the value after a hash or pound sign (`#`) in a URL used to identify a named portion of a document. The "Notes" heading above has an identity value of "notes" and can be referenced in a URL using the anchor tag helper like this:

```
1. <form asp-fragment="notes">...</form>
```

producing this HTML: `<form action="/Page#notes" method="post">...</form>` . It should be noted that fragments have no influence on the submission of a form.

3. The name of the Razor page to link to must be provided without the file extension:

```
1. <form asp-page="page">...</form>
```

If no page name is specified, the tag helper will generate a link to the current page.

4. The page handler method name will appear as a query string value unless it has been included as a route parameter for the target page.
5. Razor pages doesn't support named routes. This parameter will only be used for MVC routes.

6. The `route-` parameter enables you to specify the value for a single route value. The route parameter name is added after the hyphen. Here, the route parameter name is "key1":

```
<form asp-route-key1="value1">...</form>
```

This renders as

```
<form action="/Page/value1" method="post">...</form>
```

if the `key1` parameter is defined as part of the page's route template, or

```
<form action="/Page?key1=value1" method="post">...</form>
```

if not.

7. The anti-forgery token is rendered as a hidden input named `__RequestVerificationToken` . It will be rendered by default, unless the form has an `action` attribute specified, the form's method is set to `GET` or the `antiforgerytoken` value is set to `false` .

The Image Tag Helper

 learnrazorpages.com/razor-pages/tag-helpers/image-tag-helper

Your guide to using ASP.NET Core Razor Pages

The image tag helper targets the `` element and enables versioning of image files.

Attribute	Description
<code>append-version</code>	A boolean value indicating whether to append the image URL with a file version

Notes [link](#)

The query string named `v` is added to the image's URL if the `append-version` attribute is set to `true`. The value is calculated from the file contents, so if the file is amended, the value will differ. Browsers take query string values into account when determining whether a request can be satisfied from its cache. Therefore, if the query string value changes, the browser will retrieve the new version of the file from the server.

This usage example utilises one of the images provided with the Razor Pages (and MVC) template:

```
1. 
```


The rendered HTML is as follows:

```
1. 
```

The thing about this image is that it is an `.svg` file, which makes it easy to edit using any text editor. The following shows the new version tag after changing just the first path's `fill` color value from `#56B4D9` to `#66B4D9`:

```
1. 
```

The Input Tag Helper

 learnrazorpages.com/razor-pages/tag-helpers/input-tag-helper

Your guide to using ASP.NET Core Razor Pages

The Input tag helper generates appropriate `name` and `id` attribute values based on the `PageModel` property that is assigned to it. It will also generate an appropriate value for the `type` attribute, based on the property's meta data. The tag helper will also emit attributes that provide support for unobtrusive client-side validation.

The input tag helper has one attribute:

Attribute	Description
<code>for</code>	An expression to be evaluated against the current PageModel, usually a PageModel property name

Notes [link](#)

Although it only has one attribute, the input tag helper is quite powerful. It examines the meta data of the type that is passed to the `for` attribute, including any Data Annotations that have been applied to the property and emits HTML accordingly.

Here is a class with various property types and data annotation attributes applied to it:

```
1. public class Member
2. {
3.     public int PersonId { get; set; }
4.     public string Name { get; set; }
5.     [EmailAddress]
6.     public string Email { get; set; }
7.     [DataType(DataType.Password)]
8.     public string Password { get; set; }
9.     [DataType(DataType.PhoneNumber)]
10.    public string Telephone { get; set; }
11.    [Display(Name="Date of Birth")]
12.    public DateTime DateOfBirth { get; set; }
13.    public decimal Salary { get; set; }
14.    [Url]
15.    public string Website { get; set; }
16.    [Display(Name="Send spam to me")]
17.    public bool SendSpam { get; set; }
18.    public int? NumberOfCats { get; set; }
19.    public IFormFile Selfie { get; set; }
20. }
```

This is then applied as a property to a PageModel for a page called `Register.cshtml` :

```
1. public class RegisterModel : PageModel
2. {
3.     [BindProperty]
4.     public Member Member { get; set; }
5.     public void OnGet()
6.     {
7.     }
8. }
```

The properties of the model are applied to various input tag helpers in the Razor file:


```

1. <form method="post">
2.     <input asp-for="Member.PersonId" /><br />
3.     <input asp-for="Member.Name" /><br />
4.     <input asp-for="Member.Email" /><br />
5.     <input asp-for="Member.Password" /><br />
6.     <input asp-for="Member.Telephone" /><br />
7.     <input asp-for="Member.Website" /><br />
8.     <input asp-for="Member.DateOfBirth" /><br />
9.     <input asp-for="Member.Salary" /><br />
10.    <input asp-for="Member.SendSpam" /><br />
11.    <input asp-for="Member.NumberOfCats" /><br />
12.    <input asp-for="Member.Selfies" /><br />
13.    <button>Submit</button>
14. </form>

```

And this generates the following HTML:

```

1. <form method="post">
2.     <input type="number" data-val="true" data-val-required="The PersonId field is required."
   id="Member_PersonId" name="Member.PersonId" value="" /><br />
3.     <input type="text" id="Member_Name" name="Member.Name" value="" /><br />
4.     <input type="email" data-val="true" data-val-email="The Email field is not a valid e-mail
   address." id="Member_Email" name="Member.Email" value="" /><br />
5.     <input type="password" id="Member_Password" name="Member.Password" /><br />
6.     <input type="tel" id="Member_Telephone" name="Member.Telephone" value="" /><br />
7.     <input type="url" data-val="true" data-val-url="The Website field is not a valid fully-
   qualified http, https, or ftp URL." id="Member_Website" name="Member.Website" value="" /><br />
8.     <input type="datetime-local" data-val="true" data-val-required="The Date of Birth field is
   required." id="Member_DateOfBirth" name="Member.DateOfBirth" value="" /><br />
9.     <input type="text" data-val="true" data-val-number="The field Salary must be a number."
   data-val-required="The Salary field is required." id="Member_Salary" name="Member.Salary"
   value="" /><br />
10.    <input data-val="true" data-val-required="The Send spam to me field is required."
   id="Member_SendSpam" name="Member.SendSpam" type="checkbox" value="true" /><br />
11.    <input type="number" id="Member_NumberOfCats" name="Member.NumberOfCats" value="" /><br />
12.    <input type="file" id="Member_Selfie" name="Member.Selfie" /><br />
13.    <button>Submit</button>
14.    <input name="__RequestVerificationToken" type="hidden" value="CfDJ8I..." />
15.    <input name="Member.SendSpam" type="hidden" value="false" />
16. </form>

```

Type attribute based on data type [link](#)

The data type of the property is taken into account when the input tag helper determines the `type` attribute's value to apply. HTML5 types are used wherever possible to take advantage of features provided by supporting browsers. They behave as `type="text"` in browsers that don't support the rendered HTML5 type:

.NET Type	Input type
bool	checkbox
byte, short, int, long	number
decimal, double, float	text ¹
string	text
DateTime	datetime-local
IFormFile	file ²

1. Despite the fact that the input type is set to `text` for these data types, they will still be validated for numeric values.
2. The `IFormFile` type is located in the `Microsoft.AspNetCore.Http` namespace. It is the .NET Core successor to the `HttpPostedFile` type.

Type attribute based on data annotations [link](#)

Data annotation attributes applied to properties are also a determining factor for the selection of the input tag helper's `type` attribute. Most types are specified by a `DataType` enumeration value provided to the `DataType` attribute. Some types have their own attribute. The following table provides the `DataType` enumeration values, with equivalent attributes where they exist:

DataType Enumeration	Attribute	Input type
EmailAddress	EmailAddress	email

DataType Enumeration	Attribute	Input type
PhoneNumber	Phone	tel
Password		password
Url	Url	url
Date		date
Time		time
DateTime, Duration		datetime-local
HiddenInput ¹		hidden

All other `DataType` enumeration values (`CreditCard` , `Currency` , `Html` , `ImageUrl` , `MultilineText` , `PostCode` and `Upload`) result in `type="text"` being applied.

1. The `HiddenInput` attribute requires a reference to `Microsoft.AspNetCore.Mvc` . All other attributes reside in the `System.ComponentModel.DataAnnotations` namespace.

Validation support [link](#)

The input tag helper also emits `data` attributes that work with ASP.NET's Unobtrusive Client Validation framework (an extension to jQuery Validation). The validation rules are specified in `data-val-*` attributes and are calculated from the data types and any data annotation attributes that have been applied to model properties.

The following attributes are designed for validation purposes and will result in appropriate `data-val` error messages and other attributes being generated:

- Compare
- MaxLength
- MinLength
- Range
- Required ¹

- StringLength

1. Non-nullable properties are treated as `Required` .

In addition, the following annotation attributes generate `data-val` attributes:

- EmailAddress / `DataType.EmailAddress`
- Phone / `DataType.PhoneNumber`
- Url / `DataType.Url`

For further information on the validation attributes, see [Validation](#).

The Label Tag Helper



learnrazorpages.com/razor-pages/tag-helpers/label-tag-helper

Your guide to using ASP.NET Core Razor Pages

The label tag helper generates appropriate **for** attribute values and content based on the PageModel property that is assigned to it. It has just one attribute:

Attribute	Description
for	An expression to be evaluated against the current page model

Notes [link](#)

The label tag helper is intended to work alongside the Input tag helper. It takes a property of the PageModel as a parameter to the **asp-for** attribute and renders the name of the property as a value for the label's **for** attribute and as the content of the label tag. Assuming that the PageModel has a property named "Email":

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

This renders as

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

Note that the closing tag is required. If you use a self-closing tag e.g. `<label asp-for="Email" />`, the content will not be rendered.

You can override the value that is rendered in the label in two ways. The first option is to provide an alternative value:

```
1. <label asp-for="Email">Email Address</label>
```

Alternatively, you can use the Data Annotations **Display** attribute to change the content that is rendered:

1. `[Display(Name="Email Address")]`
2. `public string EmailAddress { get; set; }`

Both of these approaches will render as

1. `<label for="EmailAddress">Email Address</label>`

The Link Tag Helper

 learnrazorpages.com/razor-pages/tag-helpers/link-tag-helper

Your guide to using ASP.NET Core Razor Pages

The role of the link tag helper is to generate links dynamically to CSS files and fallbacks in the event that the primary file is not available, such as if the primary file is located on a remote Content Delivery Network (CDN) which is unavailable for any reason.

Attribute	Description
<code>href-include</code>	A comma separated list of globbed file patterns of CSS stylesheets to load. The glob patterns are assessed relative to the application's 'webroot' setting.
<code>href-exclude</code>	A comma separated list of globbed file patterns of CSS stylesheets to exclude from loading. The glob patterns are assessed relative to the application's 'webroot' setting. Must be used in conjunction with <code>href-include</code>
<code>fallback-href</code>	The URL of a CSS stylesheet to fallback to in the case the primary one fails.
<code>fallback-href-include</code>	A comma separated list of globbed file patterns of CSS stylesheets to fallback to in the case the primary one fails. The glob patterns are assessed relative to the application's 'webroot' setting.
<code>fallback-href-exclude</code>	A comma separated list of globbed file patterns of CSS stylesheets to exclude from the fallback list, in the case the primary one fails. The glob patterns are assessed relative to the application's 'webroot' setting. Must be used in conjunction with <code>fallback-href-include</code> .
<code>fallback-test-class</code>	The class name defined in the stylesheet to use for the fallback test. Must be used in conjunction with <code>fallback-test-property</code> and <code>fallback-test-value</code> , and either <code>fallback-href</code> or <code>fallback-href-include</code> .
<code>fallback-test-property</code>	The CSS property name to use for the fallback test. Must be used in conjunction with <code>fallback-test-class</code> and <code>fallback-test-value</code> , and either <code>fallback-href</code> or <code>fallback-href-include</code> .

Attribute	Description
<code>fallback-test-value</code>	The CSS property value to use for the fallback test. Must be used in conjunction with <code>fallback-test-class</code> and <code>fallback-test-property</code> , and either <code>fallback-href</code> or <code>fallback-href-include</code> .
<code>append-version</code>	Boolean value indicating if file version should be appended to the href urls.

Notes [link](#)

You are only likely to use this helper if you use a CDN version of a CSS file and your site will be unusable in the event that it is not available. The helper checks to see if CDN version of the style sheet is available by testing that the value for the specified class property is correct. If not, the helper injects a new link to the specified style sheet.

The default Razor Pages template illustrates its usage:

1. `<link rel="stylesheet"`
 `href="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.7/css/bootstrap.min.css"`
2. `asp-fallback-href="~/lib/bootstrap/dist/css/bootstrap.min.css"`
3. `asp-fallback-test-class="sr-only" asp-fallback-test-property="position" asp-fallback-`
 `test-value="absolute" />`
4. `<link rel="stylesheet" href="~/css/site.min.css" asp-append-version="true" />`

The preferred version of Bootstrap is hosted on the Microsoft CDN, and the fallback is stored locally. The style sheet contains a class named `sr-only` which has the following definition:

1. `.sr-only{`
2. `position:absolute;`
3. `width:1px;height:1px;`
4. `padding:0;margin:-1px;`
5. `overflow:hidden;`
6. `clip:rect(0,0,0,0);`
7. `border:0`
8. `}`

The `position` property is the subject of the test, which determines (in this particular case) whether the computed value of the property is `absolute`. If the test fails, the local version of the style sheet is loaded. The tag helper is responsible for generating a meta tag and the JavaScript for the test:

```
1. <meta name="x-stylesheet-fallback-test" content="" class="sr-only" />
2. <script>
3. !function(a,b,c,d){
4.     var e,
5.     f=document,
6.     g=f.getElementsByTagName("SCRIPT"),
7.     h=g[g.length-1].previousElementSibling,
8.     i=f.defaultView&&f.defaultView.getComputedStyle?
       f.defaultView.getComputedStyle(h):h.currentStyle;
9.     if(i&&i[a]!==b)
10.         for(e=0;e<c.length;e++)
11.             f.write('<link href="'+c[e]+' '+d+'"/>')("position","absolute",
12.             ["\\lib\\bootstrap\\dist\\css\\bootstrap.min.css"], "rel=\\u0022stylesheet\\u0022 ");
12. </script>
```

The Options Tag Helper

 learnrazorpages.com/razor-pages/tag-helpers/option-tag-helper

Your guide to using ASP.NET Core Razor Pages

The option tag helper is designed to work with the select tag helper. It has no custom attributes. It has two main uses:

1. It enables you to manually add items to a list of options to be rendered such as a default option.
2. If any of the option values that you provide manually match the value of the select tag helper's `for` attribute, they will be set as `selected`.

The first example shows a default option manually added to the select tag helper. First, here is a simple PageModel with a property called `Items` which is a collection of `SelectListItem` to be bound to a select tag helper. The options are just the numbers 1 to 3. The PageModel also has a property named `Number` which represents the selected item:

```
1. public class TaghelpersModel : PageModel
2. {
3.     public List<SelectListItem> Items =>
4.         Enumerable.Range(1, 3).Select(x => new SelectListItem {
5.             Value = x.ToString(),
6.             Text = x.ToString()
7.         }).ToList();
8.     public int Number { get; set; }
9.     public void OnGet()
10.    {
11.    }
12. }
```

Next is the select tag helper with a single option tag helper that has no value applied to it, and instructional text:

```
1. <select asp-for="Number" asp-items="Model.Items">
2.     <option value="">Pick one</option>
3. </select>
```

This results in the following rendered HTML:

```
1. <select data-val="true" data-val-required="The Number field is required." id="Number"
   name="Number">
2.     <option value="">Pick one</option>
3.     <option value="1">1</option>
4.     <option value="2">2</option>
5.     <option value="3">3</option>
6. </select>
```

The second example shows a number of options manually added to the select tag helper, one of which matches the value of the `for` attribute property. This is similar to the PageModel in the previous example, except that in this instance, the options are not present and the `Number` property has a default value of 2 applied to it:

```
1. public class TaghelpersModel : PageModel
2. {
3.     public int Number { get; set; } = 2;
4.     public void OnGet()
5.     {
6.     }
7. }
```


Here all of the options for the select tag helper have been added manually:

```
1. <select asp-for="Number">
2.     <option value="">Pick one</option>
3.     <option>1</option>
4.     <option>2</option>
5.     <option>3</option>
6. </select>
```

The rendered HTML shows that the option with a value of `2` has been set as `selected`, despite the fact that the `value` attributes on the option tag helper have not been explicitly set:

```
1. <select data-val="true" data-val-required="The Number field is required." id="Number"
   name="Number">
2.     <option value="">Pick one</option>
3.     <option>1</option>
4.     <option selected="selected">2</option>
5.     <option>3</option>
6. </select>
```

The Partial Tag Helper

 learnrazorpages.com/razor-pages/tag-helpers/partial-tag-helper

Your guide to using ASP.NET Core Razor Pages

The Partial tag helper is designed to replace the `Html.Partial` and `Html.RenderPartial` methods as the recommended mechanism for including partial pages in a Razor content page.

Attribute	Description
<code>name</code>	The name of the partial file
<code>model</code>	The model to be passed in to the partial view
<code>for</code>	An expression to be evaluated against the current page model. Cannot be used in conjunction with the <code>model</code> attribute
<code>view-data</code>	A <code>ViewData</code> dictionary to be passed to the partial view

Notes [link](#)

The `name` attribute will search all registered partial locations for a file with the supplied name (without the extension). The following example will search for `_MyPartial.cshtml` in *Pages*, *Pages/Shared* and *Views/Shared* by default:

```
1. <partial name="_MyPartial" ... />
```

You can provide a partial path, which will be appended to the search locations:

```
1. <partial name="Folder1/_MyPartial" ... />
```

Now the framework will search for the following:

- `Pages/Folder1/_MyPartial.cshtml`
- `Pages/Shared/Folder1/_MyPartial.cshtml`
- `Views/Shared/Folder1/_MyPartial.cshtml`

The `model` attribute and the `for` attribute both provide a means to pass data to the Partial. You can use one or the other but not both. The examples that follow illustrating the difference both assume that the current pagemodel has a `Contacts` property:

1. `<partial name="_MyPartial" model="Model.Contacts" />`

1. `<partial name="_MyPartial" for="@Model.Contacts" />`

2. or

3. `<partial name="_MyPartial" for="Contacts" />`

The last two examples are equivalent. The `@Model` part of the expression is inferred in the second approach.

The Script Tag Helper

 [learnrazorpages.com/razor-pages/tag-helpers/script-tag-helper](https://www.learnrazorpages.com/razor-pages/tag-helpers/script-tag-helper)

Your guide to using ASP.NET Core Razor Pages

The role of the script tag helper is to generate links dynamically to script files and fallbacks in the event that the primary file is not available, such as if the primary file is located on a remote Content Delivery Network (CDN) which is unavailable for any reason.

Attribute	Description
<code>src-include</code>	A comma separated list of globbed file patterns of script files to load. The glob patterns are assessed relative to the application's 'webroot' setting.
<code>src-exclude</code>	A comma separated list of globbed file patterns of script files to exclude from loading. The glob patterns are assessed relative to the application's 'webroot' setting. Must be used in conjunction with <code>src-include</code>
<code>fallback-src</code>	The URL of a script file to fallback to in the case the primary one fails.
<code>fallback-src-include</code>	A comma separated list of globbed file patterns of script files to fallback to in the case the primary one fails. The glob patterns are assessed relative to the application's 'webroot' setting.
<code>fallback-src-exclude</code>	A comma separated list of globbed file patterns of script files to exclude from the fallback list, in the case the primary one fails. The glob patterns are assessed relative to the application's 'webroot' setting. Must be used in conjunction with <code>fallback-src-include</code> .
<code>fallback-test</code>	A Javascript expression to use for the fallback test. Should resolve to <code>true</code> if the primary script loads successfully.
<code>append-version</code>	Boolean value indicating if a file version token should be appended to the <code>src</code> urls.

Notes [link](#)

You are only likely to use this helper if you use a CDN version of a script file and your site will be unusable in the event that it is not available. The test expression that you specify should resolve to **true** if the CDN version of the script is available. The helper will render the expression with

The default Razor Pages template illustrates its usage:

```
1. <script src="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.7/bootstrap.min.js"
2.     asp-fallback-src="~/lib/bootstrap/dist/js/bootstrap.min.js"
3.     asp-fallback-test="window.jQuery && window.jQuery.fn && window.jQuery.fn.modal"
4.     crossorigin="anonymous"
5.     integrity="sha384-Tc5IQib027qvyjSMfHjOMaLkfuWvXZxUPnCJA7l2mCWNIpG9mGCD8wGNlcpdTxa">
6. </script>
```

The preferred version of *bootstrap.min.js* is hosted on the Microsoft CDN, and the fallback is stored locally. The tag helper includes the test expression in its output:

```
1. <script src="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.7/bootstrap.min.js"
2.   crossorigin="anonymous"
3.   integrity="sha384-Tc5IQib027qvyjSMfHjOMaLkfuWvXZxUPnCJA7l2mCWNIpG9mGCD8wGNlcpdTxa">
4. </script>
5. <script>
6.   (window.jQuery && window.jQuery.fn && window.jQuery.fn.modal || document.write("\u003Cscript
   src=\u0022~/lib/bootstrap/dist/js/bootstrap.min.js\u0022 crossorigin=\u0022anonymous\u0022
   integrity=\u0022sha384-
   Tc5IQib027qvyjSMfHjOMaLkfuWvXZxUPnCJA7l2mCWNIpG9mGCD8wGNlcpdTxa\u0022\u003E\u003C\/script\u003E
7. </script>
```

The test expression is incorporated into an OR statement that results in a **script** tag for the local version being rendered if the test expression (which in this case is looking for the presence of jQuery and specifically the Bootstrap modal function) resolves to **false**.

The Select Tag Helper

 learnrazorpages.com/razor-pages/tag-helpers/select-tag-helper

Your guide to using ASP.NET Core Razor Pages

The role of the select tag helper is to render an HTML `select` element populated with options generated from a collection of `SelectListItem` objects, enumeration and/or options set via the option tag helper.

Attribute	Description
<code>for</code>	The property on the PageModel that represents the selected element(s)
<code>items</code>	A collection of <code>SelectListItem</code> objects, a <code>SelectList</code> object or an enumeration that provide the options for the select list.

Notes [link](#)

General [link](#)

The `for` attribute value is a property on the PageModel. The select tag helper uses the name of the property to generate values for the `name` and `id` attributes on the rendered `select` element. The selected value(s) will be bound to the specified model's property if the property is model bound. If the property is a collection, support for multiple selections will be enabled by the inclusion of the `multiple` attribute in the rendered element.

Options [link](#)

The `items` attribute value is a collection of `SelectListItem` or a `SelectList` that represent the options that appear in the select list, or it can be an expression that returns a collection of `SelectListItem` or a `SelectList`. The following example shows a list of numbers being created and added to `ViewData` in the `OnGet()` handler and then being bound to the `items` attribute:

```
1. public void OnGet()
2. {
3.     ViewData["Numbers"] = Enumerable.Range(1,5)
4.         .Select(n => new SelectListItem {
5.             Value = n.ToString(),
6.             Text = n.ToString()
7.         }).ToList();
8. }
```

```
1. <select asp-for="Number" asp-items="@((List<SelectListItem>)ViewData["Numbers"])">
2.     <option value="">Pick one</option>
3. </select>
```

The **ViewData** approach requires casting to the correct type, so the advice is to add the collection as a property of the PageModel:

```
1. public List<SelectListItem> Numbers => Enumerable.Range(1, 5)
2.     .Select(n => new SelectListItem {
3.         Value = n.ToString(),
4.         Text = n.ToString()
5.     }).ToList();
```

```
1. <select asp-for="Number" asp-items="Model.Numbers">
2.     <option value="">Pick one</option>
3. </select>
```

Here's another approach that shows how to use a generic **List** as a source for a **SelectList** :

```
1. [BindProperty]
2. public int PersonId { get; set; }
3. public List<Person> People { get; set; }
4. public void OnGet()
5. {
6.     People = new List<Person> {
7.         new Person { Id = 1, Name="Mike" },
8.         new Person { Id = 2, Name="Pete" },
9.         new Person { Id = 3, Name="Katy" },
10.        new Person { Id = 4, Name="Carl" }
11.    };
12. }
```

```
1. <select asp-for="PersonId" asp-items="@new SelectList(Model.People, "Id", "Name")">
2.     <option value="">Pick one</option>
3. </select>
```

Setting Selected Item [link](#)

The `SelectListItem` type has a `boolean` property named `Selected`. This can be used to set the selected item unless you specify a property for the `asp-for` attribute. If you do that, the value of the property will be set as selected, if it has one that matches the value of a `SelectListItem`:

```
1. [BindProperty]
2. public int PersonId { get; set; } = 3;
3. public List<SelectListItem> People { get; set; }
4. public void OnGet()
5. {
6.     People = new List<SelectListItem> {
7.         new SelectListItem { Value = "1", Text = "Mike" },
8.         new SelectListItem { Value = "2", Text = "Pete" },
9.         new SelectListItem { Value = "3", Text = "Katy" },
10.        new SelectListItem { Value = "4", Text = "Carl" }
11.    };
12. }
```

```
1. <select asp-for="PersonId" asp-items="Model.People">
2.     <option value="">Pick one</option>
3. </select>
```

The resulting HTML:

```
1. <select data-val="true" data-val-required="The Person field is required." id="Person"
   name="Person">
2.     <option value="">Pick one</option>
3.     <option value="1">Mike</option>
4.     <option value="2">Pete</option>
5.     <option selected="selected" value="3">Katy</option>
6.     <option value="4">Carl</option>
7. </select>
```

Enumerations [link](#)

The `Html.GetEnumSelectList` method makes it easy to use an enumeration as the data source for a select list. This next example shows how to use the `System.DayOfWeek` enumeration to present the days of the week as option values, and assumes that the `PageModel` has a property of the correct type called `DayOfWeek`:

```
1. public DayOfWeek DayOfWeek { get; set; }

1. <select asp-for="DayOfWeek" asp-items="Html.GetEnumSelectList<DayOfWeek>()">
2.     <option value="">Pick one</option>
3. </select>
```

The resulting HTML looks like this:

```
1. <select data-val="true" data-val-required="The DayOfWeek field is required." id="DayOfWeek"
   name="DayOfWeek">
2.     <option value="">Pick one</option>
3.     <option selected="selected" value="0">Sunday</option>
4.     <option value="1">Monday</option>
5.     <option value="2">Tuesday</option>
6.     <option value="3">Wednesday</option>
7.     <option value="4">Thursday</option>
8.     <option value="5">Friday</option>
9.     <option value="6">Saturday</option>
10. </select>
```

In this example, the first option is selected. This is because it matches the default value of `DayOfWeek`. If you do not want the default value to be pre-selected, you can make your model property nullable:

```
1. public DayOfWeek? DayOfWeek { get; set; }
```

SelectList [link](#)

You can create a `SelectList` from any collection but you need to specify the `DataTextField` and `DataValueField` values for the select tag helper to bind the options correctly:

```
1. public SelectList Options { get; set; }
2. public void OnGet()
3. {
4.     Options = new SelectList(context.Authors, "AuthorId", "Name");
5. }
```

Here's a version that takes a Dictionary:

```
1. public SelectList Options { get; set; }
2. public void OnGet()
3. {
4.     var dictionary = context.Authors.ToDictionary<int, string>(k => k.AuthorId, v => v.Name);
5.     Options = new SelectList(dictionary, "Key", "Value");
6. }
```

OptGroups [link](#)

The `SelectListGroup` class represents an HTML `optgroup` element. If you want to use optgroups, you can create `SelectListGroup` instances as required, and then apply them to individual `SelectListItem` s:

```
1. public int Employee { get; set; }
2. public List<SelectListItem> Staff { get; set; }
3. public void OnGet()
4. {
5.     var Sales = new SelectListGroup { Name = "Sales" };
6.     var Admin = new SelectListGroup { Name = "Admin" };
7.     var IT = new SelectListGroup { Name = "IT" };
8.     Staff = new List<SelectListItem>
9.     {
10.         new SelectListItem{ Value = "1", Text = "Mike", Group = IT},
11.         new SelectListItem{ Value = "2", Text = "Pete", Group = Sales},
12.         new SelectListItem{ Value = "3", Text = "Katy", Group = Admin},
13.         new SelectListItem{ Value = "4", Text = "Carl", Group = Sales}
14.     };
15. }
```

The following shows the rendered HTML (indented for clarity):

```
1. <select data-val="true" data-val-required="The Employee field is required." id="Employee"
   name="Employee">
2.     <option value="">Pick one</option>
3.     <optgroup label="IT">
4.         <option value="1">Mike</option>
5.     </optgroup>
6.     <optgroup label="Sales">
7.         <option value="2">Pete</option>
8.         <option value="4">Carl</option>
9.     </optgroup>
10.    <optgroup label="Admin">
11.        <option value="3">Katy</option>
12.    </optgroup>
13. </select>
```

If you are using a `SelectList`, you can specify the property to be used for grouping in the constructor:

```
1. public SelectList Staff { get; set; }
2. [BindProperty]
3. public int SelectedStaffId { get; set; }
4. public void OnGet()
5. {
6.     var staff = new List<Person>{
7.         new Person{ Id = 1, Name = "Mike", Department = "IT"},
8.         new Person{ Id = 2, Name = "Pete", Department = "Sales"},
9.         new Person{ Id = 3, Name = "Katy", Department = "Admin"},
10.        new Person{ Id = 4, Name = "Carl", Department = "Sales"}
11.    };
12.    Staff = new SelectList(staff, nameof(Person.Id), nameof(Person.Name), null,
13.        nameof(Person.Department));
14. }
```



```
1. <select asp-for="SelectedStaffId" asp-items="Model.Staff"></select>
```

In the example above, the `nameof` operator is used to minimise the chances of typos creeping into the code.

The Validation Tag Helper

 learnrazorpages.com/razor-pages/tag-helpers/validation-message-tag-helper

Your guide to using ASP.NET Core Razor Pages

The validation tag helper targets the HTML `span` element, and is used to render property-specific validation error messages.

Attribute	Description
<code>validation-for</code>	An expression to be evaluated against the current PageModel, usually a PageModel property name

Notes [link](#)

Validation tag helpers display both client-side and server-side validation error messages. They apply a CSS class named `field-validation-valid` to the span, which is changed to `field-validation-error` in the event that the form value is invalid. These styles are added to any others that you specify via the `class` attribute.

```
1. <span asp-validation-for="FirstName" class="myclass"></span>
```

renders as

```
1. <span class="myclass field-validation-valid" data-valmsg-for="FirstName" data-valmsg-replace="true"></span>
```

It is customary to position the tag helper as close to the control that it refers to so that it is easier for users to relate the error message to the relevant form value.

The Validation Summary Tag Helper

 learnrazorpages.com/razor-pages/tag-helpers/validation-summary-tag-helper

Your guide to using ASP.NET Core Razor Pages

The validation summary tag helper targets the HTML `div` element, and is used to render a summary of form validation error messages.

Attribute	Description
<code>validation-summary</code>	A <code>ValidationSummary</code> enumeration that specifies what validation errors to display.

Possible `ValidationSummary` enumeration values are:

Enum	Description
<code>None</code>	providing no validation summary (the default)
<code>ModelOnly</code>	summarising only model validation errors
<code>All</code>	summarising model <i>and</i> property validation errors

For more information on the various types of error (model or property), see the Validation topic.

Notes [link](#)

The validation summary tag helper is normally placed at the top of the form. Individual items that form the summary are displayed in an unordered list:

```
1. <div class="validation-summary-errors" data-valmsg-summary="true">
2.     <ul>
3.         <li>The FirstName field is required.</li>
4.         <li>The LastName field is required.</li>
5.         <li>The DateOfBirth field is required.</li>
6.     </ul>
7. </div>
```

You can include additional content to appear before the summary list by adding it to the content of the validation summary tag helper:

1. `<div asp-validation-summary="All">`
2. `Please correct the following errors`
3. `</div>`

The additional content will be visible by default. If you don't want users to see the content unless the page is invalid, change the `validation-summary-valid` CSS class (which is injected into the `div` by the tag helper when the page is valid) so that it hides the `div` or its content:


1. `.validation-summary-valid { display: none; }`

or, suitable for the example above where the additional content is in a `span` :

1. `.validation-summary-valid span { display: none; }`

If you specify `None` as the value for the `validation-summary` attribute, an empty `div` is rendered.

The Textarea Tag Helper

 learnrazorpages.com/razor-pages/tag-helpers/textarea-tag-helper

Your guide to using ASP.NET Core Razor Pages

The role of the textarea tag helper is to render and HTML `textarea` element for capturing multiline text.

The textarea tag helper has one attribute:

Attribute	Description
<code>for</code>	An expression to be evaluated against the current page model

Notes [link](#)

The textarea tag helper renders `id` and `name` attributes based on the name of the model property passed to the `asp-for` attribute. It also renders any associated `data` attributes required for property validation.

The `MainText` property below has a maximum length of 300 applied to it:

1. `[BindProperty, MaxLength(300)]`
2. `public string MainText { get; set; }`

This is passed to the value of the `asp-for` attribute of the tag helper:

1. `<textarea asp-for="MainText"></textarea>`

The resulting HTML includes the validation attributes for unobtrusive validation as well as the appropriate `name` attribute value for model binding:

```
1. <textarea
2.     data-val="true"
3.     data-val-maxlength="The field MainText must be a string or array type with a maximum length
   of &#x27;300&#x27;."
4.     data-val-maxlength-max="300"
5.     id="MainText"
6.     name="MainText">
```