**Introduction to Razor Pages in ASP.NET Core**

Razor Pages can make coding page-focused scenarios easier and more productive than using controllers and views.

If you're looking for a tutorial that uses the Model-View-Controller approach, see [Get started with ASP.NET Core MVC](https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/start-mvc?view=aspnetcore-8.0).

This document provides an introduction to Razor Pages. It's not a step by step tutorial. If you find some of the sections too advanced, see [Get started with Razor Pages](https://learn.microsoft.com/en-us/aspnet/core/tutorials/razor-pages/razor-pages-start?view=aspnetcore-8.0). For an overview of ASP.NET Core, see the [Introduction to ASP.NET Core](https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-8.0).

**Prerequisites**

* [Visual Studio](https://learn.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-8.0&tabs=visual-studio#tabpanel_1_visual-studio)
* [Visual Studio Code](https://learn.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-8.0&tabs=visual-studio#tabpanel_1_visual-studio-code)
* [Visual Studio for Mac](https://learn.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-8.0&tabs=visual-studio#tabpanel_1_visual-studio-mac)
* [Visual Studio 2022](https://visualstudio.microsoft.com/vs/#download) with the **ASP.NET and web development** workload.
* [.NET 6.0 SDK](https://dotnet.microsoft.com/download/dotnet/6.0)

**Create a Razor Pages project**

* [Visual Studio](https://learn.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-8.0&tabs=visual-studio#tabpanel_2_visual-studio)
* [Visual Studio Code](https://learn.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-8.0&tabs=visual-studio#tabpanel_2_visual-studio-code)
* [Visual Studio for Mac](https://learn.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-8.0&tabs=visual-studio#tabpanel_2_visual-studio-mac)

See [Get started with Razor Pages](https://learn.microsoft.com/en-us/aspnet/core/tutorials/razor-pages/razor-pages-start?view=aspnetcore-8.0) for detailed instructions on how to create a Razor Pages project.

**Razor Pages**

Razor Pages is enabled in Program.cs:

C#Copy

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddRazorPages();

var app = builder.Build();

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/Error");

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapRazorPages();

app.Run();

In the preceding code:

* [AddRazorPages](https://learn.microsoft.com/en-us/dotnet/api/microsoft.extensions.dependencyinjection.mvcservicecollectionextensions.addrazorpages) adds services for Razor Pages to the app.
* [MapRazorPages](https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.builder.razorpagesendpointroutebuilderextensions.maprazorpages) adds endpoints for Razor Pages to the [IEndpointRouteBuilder](https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.routing.iendpointroutebuilder).

Consider a basic page:

CSHTMLCopy

@page

<h1>Hello, world!</h1>

<h2>The time on the server is @DateTime.Now</h2>

The preceding code looks a lot like a [Razor view file](https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/adding-view?view=aspnetcore-8.0) used in an ASP.NET Core app with controllers and views. What makes it different is the [@page](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-8.0#page) directive. @page makes the file into an MVC action, which means that it handles requests directly, without going through a controller. @page must be the first Razor directive on a page. @page affects the behavior of other [Razor](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-8.0) constructs. Razor Pages file names have a .cshtml suffix.

A similar page, using a PageModel class, is shown in the following two files. The Pages/Index2.cshtml file:

CSHTMLCopy

@page

@using RazorPagesIntro.Pages

@model Index2Model

<h2>Separate page model</h2>

<p>

@Model.Message

</p>

The Pages/Index2.cshtml.cs page model:

C#Copy

using Microsoft.AspNetCore.Mvc.RazorPages;

using Microsoft.Extensions.Logging;

using System;

namespace RazorPagesIntro.Pages

{

public class Index2Model : PageModel

{

public string Message { get; private set; } = "PageModel in C#";

public void OnGet()

{

Message += $" Server time is { DateTime.Now }";

}

}

}

By convention, the PageModel class file has the same name as the Razor Page file with .cs appended. For example, the previous Razor Page is Pages/Index2.cshtml. The file containing the PageModel class is named Pages/Index2.cshtml.cs.

The associations of URL paths to pages are determined by the page's location in the file system. The following table shows a Razor Page path and the matching URL:

Expand table

| **File name and path** | **matching URL** |
| --- | --- |
| /Pages/Index.cshtml | / or /Index |
| /Pages/Contact.cshtml | /Contact |
| /Pages/Store/Contact.cshtml | /Store/Contact |
| /Pages/Store/Index.cshtml | /Store or /Store/Index |

Notes:

* The runtime looks for Razor Pages files in the *Pages* folder by default.
* Index is the default page when a URL doesn't include a page.

**Write a basic form**

Razor Pages is designed to make common patterns used with web browsers easy to implement when building an app. [Model binding](https://learn.microsoft.com/en-us/aspnet/core/mvc/models/model-binding?view=aspnetcore-8.0), [Tag Helpers](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/intro?view=aspnetcore-8.0), and HTML helpers work with the properties defined in a Razor Page class. Consider a page that implements a basic "contact us" form for the Contact model:

For the samples in this document, the DbContext is initialized in the [Program.cs](https://github.com/dotnet/AspNetCore.Docs/blob/main/aspnetcore/razor-pages/index/6.0sample/RazorPagesContacts/Program.cs#L11-L12) file.

The in memory database requires the Microsoft.EntityFrameworkCore.InMemory NuGet package.

C#Copy

using Microsoft.EntityFrameworkCore;

using RazorPagesContacts.Data;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddRazorPages();

builder.Services.AddDbContext<CustomerDbContext>(options =>

options.UseInMemoryDatabase("name"));

var app = builder.Build();

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/Error");

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapRazorPages();

app.Run();

The data model:

C#Copy

using System.ComponentModel.DataAnnotations;

namespace RazorPagesContacts.Models

{

public class Customer

{

public int Id { get; set; }

[Required, StringLength(10)]

public string? Name { get; set; }

}

}

The db context:

C#Copy

using Microsoft.EntityFrameworkCore;

namespace RazorPagesContacts.Data

{

public class CustomerDbContext : DbContext

{

public CustomerDbContext (DbContextOptions<CustomerDbContext> options)

: base(options)

{

}

public DbSet<RazorPagesContacts.Models.Customer> Customer => Set<RazorPagesContacts.Models.Customer>();

}

}

The Pages/Customers/Create.cshtml view file:

CSHTMLCopy

@page

@model RazorPagesContacts.Pages.Customers.CreateModel

@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers

<p>Enter a customer name:</p>

<form method="post">

Name:

<input asp-for="Customer!.Name" />

<input type="submit" />

</form>

The Pages/Customers/Create.cshtml.cs page model:

C#Copy

public class CreateModel : PageModel

{

private readonly Data.CustomerDbContext \_context;

public CreateModel(Data.CustomerDbContext context)

{

\_context = context;

}

public IActionResult OnGet()

{

return Page();

}

[BindProperty]

public Customer? Customer { get; set; }

public async Task<IActionResult> OnPostAsync()

{

if (!ModelState.IsValid)

{

return Page();

}

if (Customer != null) \_context.Customer.Add(Customer);

await \_context.SaveChangesAsync();

return RedirectToPage("./Index");

}

}

By convention, the PageModel class is called <PageName>Model and is in the same namespace as the page.

The PageModel class allows separation of the logic of a page from its presentation. It defines page handlers for requests sent to the page and the data used to render the page. This separation allows:

* Managing of page dependencies through [dependency injection](https://learn.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-8.0).
* [Unit testing](https://learn.microsoft.com/en-us/aspnet/core/test/razor-pages-tests?view=aspnetcore-8.0)

The page has an OnPostAsync *handler method*, which runs on POST requests (when a user posts the form). Handler methods for any HTTP verb can be added. The most common handlers are:

* OnGet to initialize state needed for the page. In the preceding code, the OnGet method displays the CreateModel.cshtml Razor Page.
* OnPost to handle form submissions.

The Async naming suffix is optional but is often used by convention for asynchronous functions. The preceding code is typical for Razor Pages.

If you're familiar with ASP.NET apps using controllers and views:

* The OnPostAsync code in the preceding example looks similar to typical controller code.
* Most of the MVC primitives like [model binding](https://learn.microsoft.com/en-us/aspnet/core/mvc/models/model-binding?view=aspnetcore-8.0), [validation](https://learn.microsoft.com/en-us/aspnet/core/mvc/models/validation?view=aspnetcore-8.0), and action results work the same with Controllers and Razor Pages.

The previous OnPostAsync method:

[BindProperty]

public Customer? Customer { get; set; }

public async Task<IActionResult> OnPostAsync()

{

if (!ModelState.IsValid)

{

return Page();

}

if (Customer != null) \_context.Customer.Add(Customer);

await \_context.SaveChangesAsync();

return RedirectToPage("./Index");

}

The basic flow of OnPostAsync:

Check for validation errors.

* If there are no errors, save the data and redirect.
* If there are errors, show the page again with validation messages. In many cases, validation errors would be detected on the client, and never submitted to the server.

The Pages/Customers/Create.cshtml view file:

CSHTMLCopy

@page

@model RazorPagesContacts.Pages.Customers.CreateModel

@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers

<p>Enter a customer name:</p>

<form method="post">

Name:

<input asp-for="Customer!.Name" />

<input type="submit" />

</form>

The rendered HTML from Pages/Customers/Create.cshtml:

HTMLCopy

<p>Enter a customer name:</p>

<form method="post">

Name:

<input type="text" data-val="true"

data-val-length="The field Name must be a string with a maximum length of 10."

data-val-length-max="10" data-val-required="The Name field is required."

id="Customer\_Name" maxlength="10" name="Customer.Name" value="" />

<input type="submit" />

<input name="\_\_RequestVerificationToken" type="hidden"

value="<Antiforgery token here>" />

</form>

In the previous code, posting the form:

* With valid data:
  + The OnPostAsync handler method calls the [RedirectToPage](https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.razorpages.pagemodel.redirecttopage) helper method. RedirectToPage returns an instance of [RedirectToPageResult](https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.redirecttopageresult). RedirectToPage:
    - Is an action result.
    - Is similar to RedirectToAction or RedirectToRoute (used in controllers and views).
    - Is customized for pages. In the preceding sample, it redirects to the root Index page (/Index). RedirectToPage is detailed in the [URL generation for Pages](https://learn.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-8.0&tabs=visual-studio#url_gen) section.
* With validation errors that are passed to the server:
  + The OnPostAsync handler method calls the [Page](https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.razorpages.pagebase.page) helper method. Page returns an instance of [PageResult](https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.razorpages.pageresult). Returning Page is similar to how actions in controllers return View. PageResult is the default return type for a handler method. A handler method that returns void renders the page.
  + In the preceding example, posting the form with no value results in [ModelState.IsValid](https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.modelbinding.modelstatedictionary.isvalid#microsoft-aspnetcore-mvc-modelbinding-modelstatedictionary-isvalid) returning false. In this sample, no validation errors are displayed on the client. Validation error handling is covered later in this document.

[BindProperty]

public Customer? Customer { get; set; }

public async Task<IActionResult> OnPostAsync()

{

if (!ModelState.IsValid)

{

return Page();

}

if (Customer != null) \_context.Customer.Add(Customer);

await \_context.SaveChangesAsync();

return RedirectToPage("./Index");

}

* With validation errors detected by client side validation:
  + Data is **not** posted to the server.
  + Client-side validation is explained later in this document.

The Customer property uses [[BindProperty]](https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.bindpropertyattribute) attribute to opt in to model binding:

[BindProperty]

public Customer? Customer { get; set; }

public async Task<IActionResult> OnPostAsync()

{

if (!ModelState.IsValid)

{

return Page();

}

if (Customer != null) \_context.Customer.Add(Customer);

await \_context.SaveChangesAsync();

return RedirectToPage("./Index");

}

[BindProperty] should **not** be used on models containing properties that should not be changed by the client. For more information, see [Overposting](https://learn.microsoft.com/en-us/aspnet/core/data/ef-rp/crud?view=aspnetcore-8.0#overposting).

Razor Pages, by default, bind properties only with non-GET verbs. Binding to properties removes the need to writing code to convert HTTP data to the model type. Binding reduces code by using the same property to render form fields (<input asp-for="Customer.Name">) and accept the input.

**Warning**

For security reasons, you must opt in to binding GET request data to page model properties. Verify user input before mapping it to properties. Opting into GET binding is useful when addressing scenarios that rely on query string or route values.

To bind a property on GET requests, set the [**[BindProperty]**](https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.bindpropertyattribute) attribute's SupportsGet property to true:

C#Copy

[BindProperty(SupportsGet = true)]

For more information, see [**ASP.NET Core Community Standup: Bind on GET discussion (YouTube)**](https://www.youtube.com/watch?v=p7iHB9V-KVU&feature=youtu.be&t=54m27s).

Reviewing the Pages/Customers/Create.cshtml view file:

CSHTMLCopy

@page

@model RazorPagesContacts.Pages.Customers.CreateModel

@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers

<p>Enter a customer name:</p>

<form method="post">

Name:

<input asp-for="Customer!.Name" />

<input type="submit" />

</form>

* In the preceding code, the [input tag helper](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/working-with-forms?view=aspnetcore-8.0#the-input-tag-helper) <input asp-for="Customer.Name" /> binds the HTML <input> element to the Customer.Name model expression.
* [@addTagHelper](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/intro?view=aspnetcore-8.0#addtaghelper-makes-tag-helpers-available) makes Tag Helpers available.

**The home page**

Index.cshtml is the home page:

CSHTMLCopy

@page

@model RazorPagesContacts.Pages.Customers.IndexModel

@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers

<h1>Contacts home page</h1>

<form method="post">

<table class="table">

<thead>

<tr>

<th>ID</th>

<th>Name</th>

<th></th>

</tr>

</thead>

<tbody>

@if (Model.Customers != null)

{

foreach (var contact in Model.Customers)

{

<tr>

<td> @contact.Id </td>

<td>@contact.Name</td>

<td>

<!-- <snippet\_Edit> -->

<a asp-page="./Edit" asp-route-id="@contact.Id">Edit</a> |

<!-- </snippet\_Edit> -->

<!-- <snippet\_Delete> -->

<button type="submit" asp-page-handler="delete" asp-route-id="@contact.Id">delete</button>

<!-- </snippet\_Delete> -->

</td>

</tr>

}

}

</tbody>

</table>

<a asp-page="Create">Create New</a>

</form>

The associated PageModel class (Index.cshtml.cs):

public class IndexModel : PageModel

{

private readonly Data.CustomerDbContext \_context;

public IndexModel(Data.CustomerDbContext context)

{

\_context = context;

}

public IList<Customer>? Customers { get; set; }

public async Task OnGetAsync()

{

Customers = await \_context.Customer.ToListAsync();

}

public async Task<IActionResult> OnPostDeleteAsync(int id)

{

var contact = await \_context.Customer.FindAsync(id);

if (contact != null)

{

\_context.Customer.Remove(contact);

await \_context.SaveChangesAsync();

}

return RedirectToPage();

}

}

The Index.cshtml file contains the following markup:

CSHTMLCopy

<a asp-page="./Edit" asp-route-id="@contact.Id">Edit</a> |

The <a /a> [Anchor Tag Helper](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/built-in/anchor-tag-helper?view=aspnetcore-8.0) used the asp-route-{value} attribute to generate a link to the Edit page. The link contains route data with the contact ID. For example, https://localhost:5001/Edit/1. [Tag Helpers](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/intro?view=aspnetcore-8.0) enable server-side code to participate in creating and rendering HTML elements in Razor files.

The Index.cshtml file contains markup to create a delete button for each customer contact:

CSHTMLCopy

<button type="submit" asp-page-handler="delete" asp-route-id="@contact.Id">delete</button>

The rendered HTML:

HTMLCopy

<button type="submit" formaction="/Customers?id=1&amp;handler=delete">delete</button>

When the delete button is rendered in HTML, its [formaction](https://developer.mozilla.org/docs/Web/HTML/Element/button#attr-formaction) includes parameters for:

* The customer contact ID, specified by the asp-route-id attribute.
* The handler, specified by the asp-page-handler attribute.

When the button is selected, a form POST request is sent to the server. By convention, the name of the handler method is selected based on the value of the handler parameter according to the scheme OnPost[handler]Async.

Because the handler is delete in this example, the OnPostDeleteAsync handler method is used to process the POST request. If the asp-page-handler is set to a different value, such as remove, a handler method with the name OnPostRemoveAsync is selected.

C#Copy

public async Task<IActionResult> OnPostDeleteAsync(int id)

{

var contact = await \_context.Customer.FindAsync(id);

if (contact != null)

{

\_context.Customer.Remove(contact);

await \_context.SaveChangesAsync();

}

return RedirectToPage();

}

The OnPostDeleteAsync method:

* Gets the id from the query string.
* Queries the database for the customer contact with FindAsync.
* If the customer contact is found, it's removed and the database is updated.
* Calls [RedirectToPage](https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.razorpages.pagemodel.redirecttopage) to redirect to the root Index page (/Index).

**The Edit.cshtml file**

CSHTMLCopy

@page "{id:int}"

@model RazorPagesContacts.Pages.Customers.EditModel

@{

ViewData["Title"] = "Edit";

}

<h1>Edit</h1>

<h4>Customer</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form method="post">

<div asp-validation-summary="ModelOnly" class="text-danger"></div>

<input type="hidden" asp-for="Customer!.Id" />

<div class="form-group">

<label asp-for="Customer!.Name" class="control-label"></label>

<input asp-for="Customer!.Name" class="form-control" />

<span asp-validation-for="Customer!.Name" class="text-danger"></span>

</div>

<div class="form-group">

<input type="submit" value="Save" class="btn btn-primary" />

</div>

</form>

</div>

</div>

<div>

<a asp-page="./Index">Back to List</a>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

The first line contains the @page "{id:int}" directive. The routing constraint "{id:int}" tells the page to accept requests to the page that contain int route data. If a request to the page doesn't contain route data that can be converted to an int, the runtime returns an HTTP 404 (not found) error. To make the ID optional, append ? to the route constraint:

CSHTMLCopy

@page "{id:int?}"

The Edit.cshtml.cs file:

C#Copy

public class EditModel : PageModel

{

private readonly RazorPagesContacts.Data.CustomerDbContext \_context;

public EditModel(RazorPagesContacts.Data.CustomerDbContext context)

{

\_context = context;

}

[BindProperty]

public Customer? Customer { get; set; }

public async Task<IActionResult> OnGetAsync(int? id)

{

if (id == null)

{

return NotFound();

}

Customer = await \_context.Customer.FirstOrDefaultAsync(m => m.Id == id);

if (Customer == null)

{

return NotFound();

}

return Page();

}

// To protect from overposting attacks, enable the specific properties you want to bind to.

// For more details, see https://aka.ms/RazorPagesCRUD.

public async Task<IActionResult> OnPostAsync()

{

if (!ModelState.IsValid)

{

return Page();

}

if (Customer != null)

{

\_context.Attach(Customer).State = EntityState.Modified;

try

{

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!CustomerExists(Customer.Id))

{

return NotFound();

}

else

{

throw;

}

}

}

return RedirectToPage("./Index");

}

private bool CustomerExists(int id)

{

return \_context.Customer.Any(e => e.Id == id);

}

}

**Validation**

Validation rules:

* Are declaratively specified in the model class.
* Are enforced everywhere in the app.

The [System.ComponentModel.DataAnnotations](https://learn.microsoft.com/en-us/dotnet/api/system.componentmodel.dataannotations) namespace provides a set of built-in validation attributes that are applied declaratively to a class or property. DataAnnotations also contains formatting attributes like [[DataType]](https://learn.microsoft.com/en-us/dotnet/api/system.componentmodel.dataannotations.datatypeattribute) that help with formatting and don't provide any validation.

Consider the Customer model:

C#Copy

using System.ComponentModel.DataAnnotations;

namespace RazorPagesContacts.Models

{

public class Customer

{

public int Id { get; set; }

[Required, StringLength(10)]

public string? Name { get; set; }

}

}

Using the following Create.cshtml view file:

CSHTMLCopy

@page

@model RazorPagesContacts.Pages.Customers.CreateModel

@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers

<p>Validation: customer name:</p>

<form method="post">

<div asp-validation-summary="ModelOnly"></div>

<span asp-validation-for="Customer!.Name"></span>

Name:

<input asp-for="Customer!.Name" />

<input type="submit" />

</form>

<script src="~/lib/jquery/dist/jquery.js"></script>

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

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

The preceding code:

* Includes jQuery and jQuery validation scripts.
* Uses the <div /> and <span /> [Tag Helpers](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/intro?view=aspnetcore-8.0) to enable:
  + Client-side validation.
  + Validation error rendering.
* Generates the following HTML:

HTMLCopy

<p>Enter a customer name:</p>

<form method="post">

Name:

<input type="text" data-val="true"

data-val-length="The field Name must be a string with a maximum length of 10."

data-val-length-max="10" data-val-required="The Name field is required."

id="Customer\_Name" maxlength="10" name="Customer.Name" value="" />

<input type="submit" />

<input name="\_\_RequestVerificationToken" type="hidden"

value="<Antiforgery token here>" />

</form>

<script src="/lib/jquery/dist/jquery.js"></script>

<script src="/lib/jquery-validation/dist/jquery.validate.js"></script>

<script src="/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.js"></script>

Posting the Create form without a name value displays the error message "The Name field is required." on the form. If JavaScript is enabled on the client, the browser displays the error without posting to the server.

The [StringLength(10)] attribute generates data-val-length-max="10" on the rendered HTML. data-val-length-max prevents browsers from entering more than the maximum length specified. If a tool such as [Fiddler](https://www.telerik.com/fiddler) is used to edit and replay the post:

* With the name longer than 10.
* The error message "The field Name must be a string with a maximum length of 10." is returned.

Consider the following Movie model:

C#Copy

using System;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace RazorPagesMovie.Models

{

public class Movie

{

public int ID { get; set; }

[StringLength(60, MinimumLength = 3)]

[Required]

public string Title { get; set; }

[Display(Name = "Release Date")]

[DataType(DataType.Date)]

public DateTime ReleaseDate { get; set; }

[Range(1, 100)]

[DataType(DataType.Currency)]

[Column(TypeName = "decimal(18, 2)")]

public decimal Price { get; set; }

[RegularExpression(@"^[A-Z]+[a-zA-Z\s]\*$")]

[Required]

[StringLength(30)]

public string Genre { get; set; }

[RegularExpression(@"^[A-Z]+[a-zA-Z0-9""'\s-]\*$")]

[StringLength(5)]

[Required]

public string Rating { get; set; }

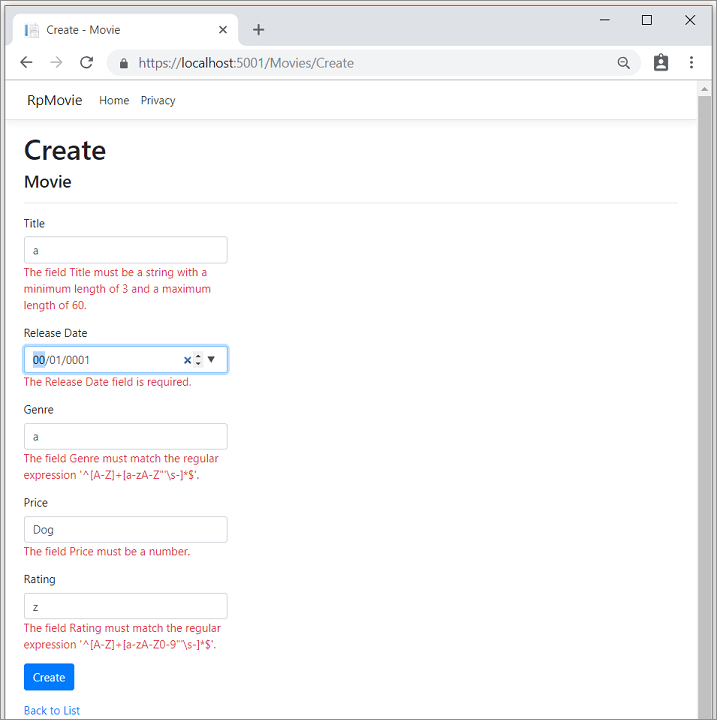
}

}

The validation attributes specify behavior to enforce on the model properties they're applied to:

* The Required and MinimumLength attributes indicate that a property must have a value, but nothing prevents a user from entering white space to satisfy this validation.
* The RegularExpression attribute is used to limit what characters can be input. In the preceding code, "Genre":
  + Must only use letters.
  + The first letter is required to be uppercase. White space, numbers, and special characters are not allowed.
* The RegularExpression "Rating":
  + Requires that the first character be an uppercase letter.
  + Allows special characters and numbers in subsequent spaces. "PG-13" is valid for a rating, but fails for a "Genre".
* The Range attribute constrains a value to within a specified range.
* The StringLength attribute sets the maximum length of a string property, and optionally its minimum length.
* Value types (such as decimal, int, float, DateTime) are inherently required and don't need the [Required] attribute.

The Create page for the Movie model shows displays errors with invalid values:



For more information, see:

* [Add validation to the Movie app](https://learn.microsoft.com/en-us/aspnet/core/tutorials/razor-pages/validation?view=aspnetcore-8.0)
* [Model validation in ASP.NET Core](https://learn.microsoft.com/en-us/aspnet/core/mvc/models/validation?view=aspnetcore-8.0).

**CSS isolation**

Isolate CSS styles to individual pages, views, and components to reduce or avoid:

* Dependencies on global styles that can be challenging to maintain.
* Style conflicts in nested content.

To add a *scoped CSS file* for a page or view, place the CSS styles in a companion .cshtml.css file matching the name of the .cshtml file. In the following example, an Index.cshtml.css file supplies CSS styles that are only applied to the Index.cshtml page or view.

Pages/Index.cshtml.css (Razor Pages) or Views/Index.cshtml.css (MVC):

cssCopy

h1 {

color: red;

}

CSS isolation occurs at build time. The framework rewrites CSS selectors to match markup rendered by the app's pages or views. The rewritten CSS styles are bundled and produced as a static asset, {APP ASSEMBLY}.styles.css. The placeholder {APP ASSEMBLY} is the assembly name of the project. A link to the bundled CSS styles is placed in the app's layout.

In the <head> content of the app's Pages/Shared/\_Layout.cshtml (Razor Pages) or Views/Shared/\_Layout.cshtml (MVC), add or confirm the presence of the link to the bundled CSS styles:

HTMLCopy

<link rel="stylesheet" href="~/{APP ASSEMBLY}.styles.css" />

In the following example, the app's assembly name is WebApp:

HTMLCopy

<link rel="stylesheet" href="WebApp.styles.css" />

The styles defined in a scoped CSS file are only applied to the rendered output of the matching file. In the preceding example, any h1 CSS declarations defined elsewhere in the app don't conflict with the Index's heading style. CSS style cascading and inheritance rules remain in effect for scoped CSS files. For example, styles applied directly to an <h1> element in the Index.cshtml file override the scoped CSS file's styles in Index.cshtml.css.

**Note**

In order to guarantee CSS style isolation when bundling occurs, importing CSS in Razor code blocks isn't supported.

CSS isolation only applies to HTML elements. CSS isolation isn't supported for [**Tag Helpers**](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/intro?view=aspnetcore-8.0).

Within the bundled CSS file, each page, view, or Razor component is associated with a scope identifier in the format b-{STRING}, where the {STRING} placeholder is a ten-character string generated by the framework. The following example provides the style for the preceding <h1> element in the Index page of a Razor Pages app:

cssCopy

/\* /Pages/Index.cshtml.rz.scp.css \*/

h1[b-3xxtam6d07] {

color: red;

}

In the Index page where the CSS style is applied from the bundled file, the scope identifier is appended as an HTML attribute:

HTMLCopy

<h1 b-3xxtam6d07>

The identifier is unique to an app. At build time, a project bundle is created with the convention {STATIC WEB ASSETS BASE PATH}/Project.lib.scp.css, where the placeholder {STATIC WEB ASSETS BASE PATH} is the static web assets base path.

If other projects are utilized, such as NuGet packages or [Razor class libraries](https://learn.microsoft.com/en-us/aspnet/core/razor-pages/ui-class?view=aspnetcore-8.0), the bundled file:

* References the styles using CSS imports.
* Isn't published as a static web asset of the app that consumes the styles.

**CSS preprocessor support**

CSS preprocessors are useful for improving CSS development by utilizing features such as variables, nesting, modules, mixins, and inheritance. While CSS isolation doesn't natively support CSS preprocessors such as Sass or Less, integrating CSS preprocessors is seamless as long as preprocessor compilation occurs before the framework rewrites the CSS selectors during the build process. Using Visual Studio for example, configure existing preprocessor compilation as a **Before Build** task in the Visual Studio Task Runner Explorer.

Many third-party NuGet packages, such as [AspNetCore.SassCompiler](https://www.nuget.org/packages/AspNetCore.SassCompiler#readme-body-tab), can compile SASS/SCSS files at the beginning of the build process before CSS isolation occurs, and no additional configuration is required.

**CSS isolation configuration**

CSS isolation permits configuration for some advanced scenarios, such as when there are dependencies on existing tools or workflows.

**Customize scope identifier format**

*In this section, the {Pages|Views} placeholder is either Pages for Razor Pages apps or Views for MVC apps.*

By default, scope identifiers use the format b-{STRING}, where the {STRING} placeholder is a ten-character string generated by the framework. To customize the scope identifier format, update the project file to a desired pattern:

XMLCopy

<ItemGroup>

<None Update="{Pages|Views}/Index.cshtml.css" CssScope="custom-scope-identifier" />

</ItemGroup>

In the preceding example, the CSS generated for Index.cshtml.css changes its scope identifier from b-{STRING} to custom-scope-identifier.

Use scope identifiers to achieve inheritance with scoped CSS files. In the following project file example, a BaseView.cshtml.css file contains common styles across views. A DerivedView.cshtml.css file inherits these styles.

XMLCopy

<ItemGroup>

<None Update="{Pages|Views}/BaseView.cshtml.css" CssScope="custom-scope-identifier" />

<None Update="{Pages|Views}/DerivedView.cshtml.css" CssScope="custom-scope-identifier" />

</ItemGroup>

Use the wildcard (\*) operator to share scope identifiers across multiple files:

XMLCopy

<ItemGroup>

<None Update="{Pages|Views}/\*.cshtml.css" CssScope="custom-scope-identifier" />

</ItemGroup>

**Change base path for static web assets**

The scoped CSS file is generated at the root of the app. In the project file, use the StaticWebAssetBasePath property to change the default path. The following example places the scoped CSS file, and the rest of the app's assets, at the \_content path:

XMLCopy

<PropertyGroup>

<StaticWebAssetBasePath>\_content/$(PackageId)</StaticWebAssetBasePath>

</PropertyGroup>

**Disable automatic bundling**

To opt out of how framework publishes and loads scoped files at runtime, use the DisableScopedCssBundling property. When using this property, other tools or processes are responsible for taking the isolated CSS files from the obj directory and publishing and loading them at runtime:

XMLCopy

<PropertyGroup>

<DisableScopedCssBundling>true</DisableScopedCssBundling>

</PropertyGroup>

**Razor class library (RCL) support**

When a [Razor class library (RCL)](https://learn.microsoft.com/en-us/aspnet/core/razor-pages/ui-class?view=aspnetcore-8.0) provides isolated styles, the <link> tag's href attribute points to {STATIC WEB ASSET BASE PATH}/{PACKAGE ID}.bundle.scp.css, where the placeholders are:

* {STATIC WEB ASSET BASE PATH}: The static web asset base path.
* {PACKAGE ID}: The library's [package identifier](https://learn.microsoft.com/en-us/nuget/create-packages/creating-a-package-msbuild#set-properties). The package identifier defaults to the project's assembly name if the package identifier isn't specified in the project file.

In the following example:

* The static web asset base path is \_content/ClassLib.
* The class library's assembly name is ClassLib.

Pages/Shared/\_Layout.cshtml (Razor Pages) or Views/Shared/\_Layout.cshtml (MVC):

HTMLCopy

<link href="\_content/ClassLib/ClassLib.bundle.scp.css" rel="stylesheet">

For more information on RCLs, see the following articles:

* [Reusable Razor UI in class libraries with ASP.NET Core](https://learn.microsoft.com/en-us/aspnet/core/razor-pages/ui-class?view=aspnetcore-8.0)
* [Consume ASP.NET Core Razor components from a Razor class library (RCL)](https://learn.microsoft.com/en-us/aspnet/core/blazor/components/class-libraries?view=aspnetcore-8.0)

For information on Blazor CSS isolation, see [ASP.NET Core Blazor CSS isolation](https://learn.microsoft.com/en-us/aspnet/core/blazor/components/css-isolation?view=aspnetcore-8.0).

**Handle HEAD requests with an OnGet handler fallback**

HEAD requests allow retrieving the headers for a specific resource. Unlike GET requests, HEAD requests don't return a response body.

Ordinarily, an OnHead handler is created and called for HEAD requests:

C#Copy

public void OnHead()

{

HttpContext.Response.Headers.Add("Head Test", "Handled by OnHead!");

}

Razor Pages falls back to calling the OnGet handler if no OnHead handler is defined.

**XSRF/CSRF and Razor Pages**

Razor Pages are protected by [Antiforgery validation](https://learn.microsoft.com/en-us/aspnet/core/security/anti-request-forgery?view=aspnetcore-8.0). The [FormTagHelper](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/working-with-forms?view=aspnetcore-8.0#the-form-tag-helper) injects antiforgery tokens into HTML form elements.

**Using Layouts, partials, templates, and Tag Helpers with Razor Pages**

Pages work with all the capabilities of the Razor view engine. Layouts, partials, templates, Tag Helpers, \_ViewStart.cshtml, and \_ViewImports.cshtml work in the same way they do for conventional Razor views.

Let's declutter this page by taking advantage of some of those capabilities.

Add a [layout page](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/layout?view=aspnetcore-8.0) to Pages/Shared/\_Layout.cshtml:

CSHTMLCopy

<!DOCTYPE html>

<html>

<head>

<title>RP Sample</title>

<link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.css" />

</head>

<body>

<a asp-page="/Index">Home</a>

<a asp-page="/Customers/Create">Create</a>

<a asp-page="/Customers/Index">Customers</a> <br />

@RenderBody()

<script src="~/lib/jquery/dist/jquery.js"></script>

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

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

</body>

</html>

The [Layout](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/layout?view=aspnetcore-8.0):

* Controls the layout of each page (unless the page opts out of layout).
* Imports HTML structures such as JavaScript and stylesheets.
* The contents of the Razor page are rendered where @RenderBody() is called.

For more information, see [layout page](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/layout?view=aspnetcore-8.0).

The [Layout](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/layout?view=aspnetcore-8.0#specifying-a-layout) property is set in Pages/\_ViewStart.cshtml:

CSHTMLCopy

@{

Layout = "\_Layout";

}

The layout is in the *Pages/Shared* folder. Pages look for other views (layouts, templates, partials) hierarchically, starting in the same folder as the current page. A layout in the *Pages/Shared* folder can be used from any Razor page under the *Pages* folder.

The layout file should go in the *Pages/Shared* folder.

We recommend you **not** put the layout file in the *Views/Shared* folder. *Views/Shared* is an MVC views pattern. Razor Pages are meant to rely on folder hierarchy, not path conventions.

View search from a Razor Page includes the *Pages* folder. The layouts, templates, and partials used with MVC controllers and conventional Razor views *just work*.

Add a Pages/\_ViewImports.cshtml file:

CSHTMLCopy

@namespace RazorPagesContacts.Pages

@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers

@namespace is explained later in the tutorial. The @addTagHelper directive brings in the [built-in Tag Helpers](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/built-in/?view=aspnetcore-8.0) to all the pages in the *Pages* folder.

The @namespace directive set on a page:

CSHTMLCopy

@page

@namespace RazorPagesIntro.Pages.Customers

@model NameSpaceModel

<h2>Name space</h2>

<p>

@Model.Message

</p>

The @namespace directive sets the namespace for the page. The @model directive doesn't need to include the namespace.

When the @namespace directive is contained in \_ViewImports.cshtml, the specified namespace supplies the prefix for the generated namespace in the Page that imports the @namespace directive. The rest of the generated namespace (the suffix portion) is the dot-separated relative path between the folder containing \_ViewImports.cshtml and the folder containing the page.

For example, the PageModel class Pages/Customers/Edit.cshtml.cs explicitly sets the namespace:

C#Copy

namespace RazorPagesContacts.Pages

{

public class EditModel : PageModel

{

private readonly AppDbContext \_db;

public EditModel(AppDbContext db)

{

\_db = db;

}

// Code removed for brevity.

The Pages/\_ViewImports.cshtml file sets the following namespace:

CSHTMLCopy

@namespace RazorPagesContacts.Pages

@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers

The generated namespace for the Pages/Customers/Edit.cshtml Razor Page is the same as the PageModel class.

@namespace *also works with conventional Razor views.*

Consider the Pages/Customers/Create.cshtml view file:

CSHTMLCopy

@page

@model RazorPagesContacts.Pages.Customers.CreateModel

@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers

<p>Validation: customer name:</p>

<form method="post">

<div asp-validation-summary="ModelOnly"></div>

<span asp-validation-for="Customer!.Name"></span>

Name:

<input asp-for="Customer!.Name" />

<input type="submit" />

</form>

<script src="~/lib/jquery/dist/jquery.js"></script>

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

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

The updated Pages/Customers/Create.cshtml view file with \_ViewImports.cshtml and the preceding layout file:

CSHTMLCopy

@page

@model CreateModel

<p>Enter a customer name:</p>

<form method="post">

Name:

<input asp-for="Customer!.Name" />

<input type="submit" />

</form>

In the preceding code, the \_ViewImports.cshtml imported the namespace and Tag Helpers. The layout file imported the JavaScript files.

The [Razor Pages starter project](https://learn.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-8.0&tabs=visual-studio#rpvs17) contains the Pages/\_ValidationScriptsPartial.cshtml, which hooks up client-side validation.

For more information on partial views, see [Partial views in ASP.NET Core](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/partial?view=aspnetcore-8.0).

**URL generation for Pages**

The Create page, shown previously, uses RedirectToPage:

C#Copy

public class CreateModel : PageModel

{

private readonly Data.CustomerDbContext \_context;

public CreateModel(Data.CustomerDbContext context)

{

\_context = context;

}

public IActionResult OnGet()

{

return Page();

}

[BindProperty]

public Customer? Customer { get; set; }

public async Task<IActionResult> OnPostAsync()

{

if (!ModelState.IsValid)

{

return Page();

}

if (Customer != null) \_context.Customer.Add(Customer);

await \_context.SaveChangesAsync();

return RedirectToPage("./Index");

}

}

The app has the following file/folder structure:

* */Pages*
  + Index.cshtml
  + Privacy.cshtml
  + */Customers*
    - Create.cshtml
    - Edit.cshtml
    - Index.cshtml

The Pages/Customers/Create.cshtml and Pages/Customers/Edit.cshtml pages redirect to Pages/Customers/Index.cshtml after success. The string ./Index is a relative page name used to access the preceding page. It is used to generate URLs to the Pages/Customers/Index.cshtml page. For example:

* Url.Page("./Index", ...)
* <a asp-page="./Index">Customers Index Page</a>
* RedirectToPage("./Index")

The absolute page name /Index is used to generate URLs to the Pages/Index.cshtml page. For example:

* Url.Page("/Index", ...)
* <a asp-page="/Index">Home Index Page</a>
* RedirectToPage("/Index")

The page name is the path to the page from the root */Pages* folder including a leading / (for example, /Index). The preceding URL generation samples offer enhanced options and functional capabilities over hard-coding a URL. URL generation uses [routing](https://learn.microsoft.com/en-us/aspnet/core/mvc/controllers/routing?view=aspnetcore-8.0) and can generate and encode parameters according to how the route is defined in the destination path.

URL generation for pages supports relative names. The following table shows which Index page is selected using different RedirectToPage parameters in Pages/Customers/Create.cshtml.

Expand table

| **RedirectToPage(x)** | **Page** |
| --- | --- |
| RedirectToPage("/Index") | *Pages/Index* |
| RedirectToPage("./Index"); | *Pages/Customers/Index* |
| RedirectToPage("../Index") | *Pages/Index* |
| RedirectToPage("Index") | *Pages/Customers/Index* |

RedirectToPage("Index"), RedirectToPage("./Index"), and RedirectToPage("../Index") are *relative names*. The RedirectToPage parameter is *combined* with the path of the current page to compute the name of the destination page.

Relative name linking is useful when building sites with a complex structure. When relative names are used to link between pages in a folder:

* Renaming a folder doesn't break the relative links.
* Links are not broken because they don't include the folder name.

To redirect to a page in a different [Area](https://learn.microsoft.com/en-us/aspnet/core/mvc/controllers/areas?view=aspnetcore-8.0), specify the area:

RedirectToPage("/Index", new { area = "Services" });

For more information, see [Areas in ASP.NET Core](https://learn.microsoft.com/en-us/aspnet/core/mvc/controllers/areas?view=aspnetcore-8.0) and [Razor Pages route and app conventions in ASP.NET Core](https://learn.microsoft.com/en-us/aspnet/core/razor-pages/razor-pages-conventions?view=aspnetcore-8.0).

**ViewData attribute**

Data can be passed to a page with [ViewDataAttribute](https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.viewdataattribute). Properties with the [ViewData] attribute have their values stored and loaded from the [ViewDataDictionary](https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.viewfeatures.viewdatadictionary).

In the following example, the AboutModel applies the [ViewData] attribute to the Title property:

public class AboutModel : PageModel

{

[ViewData]

public string Title { get; } = "About";

public void OnGet()

{ }

}

In the About page, access the Title property as a model property:

<h1>@Model.Title</h1>

In the layout, the title is read from the ViewData dictionary:

<!DOCTYPE html>

<html lang="en">

<head>

<title>@ViewData["Title"] - WebApplication</title>

...

**TempData**

ASP.NET Core exposes the [TempData](https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.controller.tempdata#microsoft-aspnetcore-mvc-controller-tempdata). This property stores data until it's read. The [Keep](https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.viewfeatures.tempdatadictionary.keep) and [Peek](https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.viewfeatures.tempdatadictionary.peek) methods can be used to examine the data without deletion. TempData is useful for redirection, when data is needed for more than a single request.

The following code sets the value of Message using TempData:

public class CreateDotModel : PageModel

{

private readonly AppDbContext \_db;

public CreateDotModel(AppDbContext db)

{

\_db = db;

}

[TempData]

public string Message { get; set; }

[BindProperty]

public Customer Customer { get; set; }

public async Task<IActionResult> OnPostAsync()

{

if (!ModelState.IsValid)

{

return Page();

}

\_db.Customers.Add(Customer);

await \_db.SaveChangesAsync();

Message = $"Customer {Customer.Name} added";

return RedirectToPage("./Index");

}

}

The following markup in the Pages/Customers/Index.cshtml file displays the value of Message using TempData.

CSHTMLCopy

<h3>Msg: @Model.Message</h3>

The Pages/Customers/Index.cshtml.cs page model applies the [TempData] attribute to the Message property.

C#Copy

[TempData]

public string Message { get; set; }

For more information, see [TempData](https://learn.microsoft.com/en-us/aspnet/core/fundamentals/app-state?view=aspnetcore-8.0#tempdata).

**Multiple handlers per page**

The following page generates markup for two handlers using the asp-page-handler Tag Helper:

CSHTMLCopy

@page

@model CreateFATHModel

<html>

<body>

<p>

Enter your name.

</p>

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

<form method="POST">

<div>Name: <input asp-for="Customer.Name" /></div>

<!-- <snippet\_Handlers> -->

<input type="submit" asp-page-handler="JoinList" value="Join" />

<input type="submit" asp-page-handler="JoinListUC" value="JOIN UC" />

<!-- </snippet\_Handlers> -->

</form>

</body>

</html>

The form in the preceding example has two submit buttons, each using the FormActionTagHelper to submit to a different URL. The asp-page-handler attribute is a companion to asp-page. asp-page-handler generates URLs that submit to each of the handler methods defined by a page. asp-page isn't specified because the sample is linking to the current page.

The page model:

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.RazorPages;

using RazorPagesContacts.Data;

namespace RazorPagesContacts.Pages.Customers

{

public class CreateFATHModel : PageModel

{

private readonly AppDbContext \_db;

public CreateFATHModel(AppDbContext db)

{

\_db = db;

}

[BindProperty]

public Customer Customer { get; set; }

public async Task<IActionResult> OnPostJoinListAsync()

{

if (!ModelState.IsValid)

{

return Page();

}

\_db.Customers.Add(Customer);

await \_db.SaveChangesAsync();

return RedirectToPage("/Index");

}

public async Task<IActionResult> OnPostJoinListUCAsync()

{

if (!ModelState.IsValid)

{

return Page();

}

Customer.Name = Customer.Name?.ToUpperInvariant();

return await OnPostJoinListAsync();

}

}

}

The preceding code uses *named handler methods*. Named handler methods are created by taking the text in the name after On<HTTP Verb> and before Async (if present). In the preceding example, the page methods are OnPost**JoinList**Async and OnPost**JoinListUC**Async. With *OnPost* and *Async* removed, the handler names are JoinList and JoinListUC.

CSHTMLCopy

<input type="submit" asp-page-handler="JoinList" value="Join" />

<input type="submit" asp-page-handler="JoinListUC" value="JOIN UC" />

Using the preceding code, the URL path that submits to OnPostJoinListAsync is https://localhost:5001/Customers/CreateFATH?handler=JoinList. The URL path that submits to OnPostJoinListUCAsync is https://localhost:5001/Customers/CreateFATH?handler=JoinListUC.

**Custom routes**

Use the @page directive to:

* Specify a custom route to a page. For example, the route to the About page can be set to /Some/Other/Path with @page "/Some/Other/Path".
* Append segments to a page's default route. For example, an "item" segment can be added to a page's default route with @page "item".
* Append parameters to a page's default route. For example, an ID parameter, id, can be required for a page with @page "{id}".

A root-relative path designated by a tilde (~) at the beginning of the path is supported. For example, @page "~/Some/Other/Path" is the same as @page "/Some/Other/Path".

If you don't like the query string ?handler=JoinList in the URL, change the route to put the handler name in the path portion of the URL. The route can be customized by adding a route template enclosed in double quotes after the @page directive.

CSHTMLCopy

@page "{handler?}"

@model CreateRouteModel

<html>

<body>

<p>

Enter your name.

</p>

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

<form method="POST">

<div>Name: <input asp-for="Customer.Name" /></div>

<input type="submit" asp-page-handler="JoinList" value="Join" />

<input type="submit" asp-page-handler="JoinListUC" value="JOIN UC" />

</form>

</body>

</html>

Using the preceding code, the URL path that submits to OnPostJoinListAsync is https://localhost:5001/Customers/CreateFATH/JoinList. The URL path that submits to OnPostJoinListUCAsync is https://localhost:5001/Customers/CreateFATH/JoinListUC.

The ? following handler means the route parameter is optional.

**Collocation of JavaScript (JS) files**

Collocation of JavaScript (JS) files for pages and views is a convenient way to organize scripts in an app.

Collocate JS files using the following filename extension conventions:

* Pages of Razor Pages apps and views of MVC apps: .cshtml.js. Examples:
  + Pages/Index.cshtml.js for the Index page of a Razor Pages app at Pages/Index.cshtml.
  + Views/Home/Index.cshtml.js for the Index view of an MVC app at Views/Home/Index.cshtml.

Collocated JS files are publicly addressable using the ***path to the file in the project***:

* Pages and views from a collocated scripts file in the app:

{PATH}/{PAGE, VIEW, OR COMPONENT}.{EXTENSION}.js

* + The {PATH} placeholder is the path to the page, view, or component.
  + The {PAGE, VIEW, OR COMPONENT} placeholder is the page, view, or component.
  + The {EXTENSION} placeholder matches the extension of the page, view, or component, either razor or cshtml.

Razor Pages example:

A JS file for the Index page is placed in the Pages folder (Pages/Index.cshtml.js) next to the Index page (Pages/Index.cshtml). In the Index page, the script is referenced at the path in the Pages folder:

razorCopy

@section Scripts {

<script src="~/Pages/Index.cshtml.js"></script>

}

When the app is published, the framework automatically moves the script to the web root. In the preceding example, the script is moved to bin\Release\{TARGET FRAMEWORK MONIKER}\publish\wwwroot\Pages\Index.cshtml.js, where the {TARGET FRAMEWORK MONIKER} placeholder is the [Target Framework Moniker (TFM)](https://learn.microsoft.com/en-us/dotnet/standard/frameworks). No change is required to the script's relative URL in the Index page.

When the app is published, the framework automatically moves the script to the web root. In the preceding example, the script is moved to bin\Release\{TARGET FRAMEWORK MONIKER}\publish\wwwroot\Components\Pages\Index.razor.js, where the {TARGET FRAMEWORK MONIKER} placeholder is the [Target Framework Moniker (TFM)](https://learn.microsoft.com/en-us/dotnet/standard/frameworks). No change is required to the script's relative URL in the Index component.

* For scripts provided by a Razor class library (RCL):

\_content/{PACKAGE ID}/{PATH}/{PAGE, VIEW, OR COMPONENT}.{EXTENSION}.js

* + The {PACKAGE ID} placeholder is the RCL's package identifier (or library name for a class library referenced by the app).
  + The {PATH} placeholder is the path to the page, view, or component. If a Razor component is located at the root of the RCL, the path segment isn't included.
  + The {PAGE, VIEW, OR COMPONENT} placeholder is the page, view, or component.
  + The {EXTENSION} placeholder matches the extension of page, view, or component, either razor or cshtml.

**Advanced configuration and settings**

The configuration and settings in following sections is not required by most apps.

To configure advanced options, use the [AddRazorPages](https://learn.microsoft.com/en-us/dotnet/api/microsoft.extensions.dependencyinjection.mvcservicecollectionextensions.addrazorpages) overload that configures [RazorPagesOptions](https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.razorpages.razorpagesoptions):

using Microsoft.EntityFrameworkCore;

using RazorPagesContacts.Data;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddRazorPages(options =>

{

options.RootDirectory = "/MyPages";

options.Conventions.AuthorizeFolder("/MyPages/Admin");

});

builder.Services.AddDbContext<CustomerDbContext>(options =>

options.UseInMemoryDatabase("name"));

var app = builder.Build();

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/Error");

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapRazorPages();

app.Run();

Use the [RazorPagesOptions](https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.razorpages.razorpagesoptions) to set the root directory for pages, or add application model conventions for pages. For more information on conventions, see [Razor Pages authorization conventions](https://learn.microsoft.com/en-us/aspnet/core/security/authorization/razor-pages-authorization?view=aspnetcore-8.0).

To precompile views, see [Razor view compilation](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/view-compilation?view=aspnetcore-8.0).

**Specify that Razor Pages are at the content root**

By default, Razor Pages are rooted in the */Pages* directory. Add [WithRazorPagesAtContentRoot](https://learn.microsoft.com/en-us/dotnet/api/microsoft.extensions.dependencyinjection.mvcrazorpagesmvcbuilderextensions.withrazorpagesatcontentroot) to specify that your Razor Pages are at the [content root](https://learn.microsoft.com/en-us/aspnet/core/fundamentals/?view=aspnetcore-8.0#content-root) ([ContentRootPath](https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.hosting.ihostingenvironment.contentrootpath#microsoft-aspnetcore-hosting-ihostingenvironment-contentrootpath)) of the app:

using Microsoft.EntityFrameworkCore;

using RazorPagesContacts.Data;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddRazorPages(options =>

{

options.Conventions.AuthorizeFolder("/MyPages/Admin");

})

.WithRazorPagesAtContentRoot();

builder.Services.AddDbContext<CustomerDbContext>(options =>

options.UseInMemoryDatabase("name"));

var app = builder.Build();

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/Error");

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapRazorPages();

app.Run();

**Specify that Razor Pages are at a custom root directory**

Add [WithRazorPagesRoot](https://learn.microsoft.com/en-us/dotnet/api/microsoft.extensions.dependencyinjection.mvcrazorpagesmvccorebuilderextensions.withrazorpagesroot) to specify that Razor Pages are at a custom root directory in the app (provide a relative path):

using Microsoft.EntityFrameworkCore;

using RazorPagesContacts.Data;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddRazorPages(options =>

{

options.Conventions.AuthorizeFolder("/MyPages/Admin");

})

.WithRazorPagesRoot("/path/to/razor/pages");

builder.Services.AddDbContext<CustomerDbContext>(options =>

options.UseInMemoryDatabase("name"));

var app = builder.Build();

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/Error");

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapRazorPages();

app.Run();

**Razor syntax reference for ASP.NET Core**

* Article
* 11/14/2023
* 27 contributors

Feedback

**In this article**

1. [Rendering HTML](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-8.0#rendering-html)
2. [Razor syntax](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-8.0#razor-syntax)
3. [Implicit Razor expressions](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-8.0#implicit-razor-expressions)
4. [Explicit Razor expressions](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-8.0#explicit-razor-expressions)

Show 12 more

By [Rick Anderson](https://twitter.com/RickAndMSFT), [Taylor Mullen](https://twitter.com/ntaylormullen), and [Dan Vicarel](https://github.com/Rabadash8820)

Razor is a markup syntax for embedding .NET based code into webpages. The Razor syntax consists of Razor markup, C#, and HTML. Files containing Razor generally have a .cshtml file extension. Razor is also found in [Razor component](https://learn.microsoft.com/en-us/aspnet/core/blazor/components/?view=aspnetcore-8.0) files (.razor). Razor syntax is similar to the templating engines of various JavaScript single-page application (SPA) frameworks, such as Angular, React, VueJs, and Svelte. For more information see, [The features described in this article are obsolete as of ASP.NET Core 3.0](https://learn.microsoft.com/en-us/aspnet/core/client-side/spa-services?view=aspnetcore-8.0).

[Introduction to ASP.NET Web Programming Using the Razor Syntax](https://learn.microsoft.com/en-us/aspnet/web-pages/overview/getting-started/introducing-razor-syntax-c) provides many samples of programming with Razor syntax. Although the topic was written for ASP.NET rather than ASP.NET Core, most of the samples apply to ASP.NET Core.

**Rendering HTML**

The default Razor language is HTML. Rendering HTML from Razor markup is no different than rendering HTML from an HTML file. HTML markup in .cshtml Razor files is rendered by the server unchanged.

**Razor syntax**

Razor supports C# and uses the @ symbol to transition from HTML to C#. Razor evaluates C# expressions and renders them in the HTML output.

When an @ symbol is followed by a [Razor reserved keyword](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-8.0#razor-reserved-keywords), it transitions into Razor-specific markup. Otherwise, it transitions into plain HTML.

To escape an @ symbol in Razor markup, use a second @ symbol:

CSHTMLCopy

<p>@@Username</p>

The code is rendered in HTML with a single @ symbol:

HTMLCopy

<p>@Username</p>

HTML attributes and content containing email addresses don't treat the @ symbol as a transition character. The email addresses in the following example are untouched by Razor parsing:

CSHTMLCopy

<a href="mailto:Support@contoso.com">Support@contoso.com</a>

**Scalable Vector Graphics (SVG)**

[SVG](https://developer.mozilla.org/docs/Web/SVG) [foreignObject](https://developer.mozilla.org/docs/Web/SVG/Element/foreignObject) elements are supported:

HTMLCopy

@{

string message = "foreignObject example with Scalable Vector Graphics (SVG)";

}

<svg width="200" height="200" xmlns="http://www.w3.org/2000/svg">

<rect x="0" y="0" rx="10" ry="10" width="200" height="200" stroke="black"

fill="none" />

<foreignObject x="20" y="20" width="160" height="160">

<p>@message</p>

</foreignObject>

</svg>

**Implicit Razor expressions**

Implicit Razor expressions start with @ followed by C# code:

CSHTMLCopy

<p>@DateTime.Now</p>

<p>@DateTime.IsLeapYear(2016)</p>

With the exception of the C# await keyword, implicit expressions must not contain spaces. If the C# statement has a clear ending, spaces can be intermingled:

CSHTMLCopy

<p>@await DoSomething("hello", "world")</p>

Implicit expressions **cannot** contain C# generics, as the characters inside the brackets (<>) are interpreted as an HTML tag. The following code is **not** valid:

CSHTMLCopy

<p>@GenericMethod<int>()</p>

The preceding code generates a compiler error similar to one of the following:

* The "int" element wasn't closed. All elements must be either self-closing or have a matching end tag.
* Cannot convert method group 'GenericMethod' to non-delegate type 'object'. Did you intend to invoke the method?`

Generic method calls must be wrapped in an [explicit Razor expression](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-8.0#explicit-razor-expressions) or a [Razor code block](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-8.0#razor-code-blocks).

**Explicit Razor expressions**

Explicit Razor expressions consist of an @ symbol with balanced parenthesis. To render last week's time, the following Razor markup is used:

CSHTMLCopy

<p>Last week this time: @(DateTime.Now - TimeSpan.FromDays(7))</p>

Any content within the @() parenthesis is evaluated and rendered to the output.

Implicit expressions, described in the previous section, generally can't contain spaces. In the following code, one week isn't subtracted from the current time:

CSHTMLCopy

<p>Last week: @DateTime.Now - TimeSpan.FromDays(7)</p>

The code renders the following HTML:

HTMLCopy

<p>Last week: 7/7/2016 4:39:52 PM - TimeSpan.FromDays(7)</p>

Explicit expressions can be used to concatenate text with an expression result:

CSHTMLCopy

@{

var joe = new Person("Joe", 33);

}

<p>Age@(joe.Age)</p>

Without the explicit expression, <p>Age@joe.Age</p> is treated as an email address, and <p>Age@joe.Age</p> is rendered. When written as an explicit expression, <p>Age33</p> is rendered.

Explicit expressions can be used to render output from generic methods in .cshtml files. The following markup shows how to correct the error shown earlier caused by the brackets of a C# generic. The code is written as an explicit expression:

CSHTMLCopy

<p>@(GenericMethod<int>())</p>

**Expression encoding**

C# expressions that evaluate to a string are HTML encoded. C# expressions that evaluate to IHtmlContent are rendered directly through IHtmlContent.WriteTo. C# expressions that don't evaluate to IHtmlContent are converted to a string by ToString and encoded before they're rendered.

CSHTMLCopy

@("<span>Hello World</span>")

The preceding code renders the following HTML:

HTMLCopy

&lt;span&gt;Hello World&lt;/span&gt;

The HTML is shown in the browser as plain text:

<span>Hello World</span>

HtmlHelper.Raw output isn't encoded but rendered as HTML markup.

**Warning**

Using HtmlHelper.Raw on unsanitized user input is a security risk. User input might contain malicious JavaScript or other exploits. Sanitizing user input is difficult. Avoid using HtmlHelper.Raw with user input.

CSHTMLCopy

@Html.Raw("<span>Hello World</span>")

The code renders the following HTML:

HTMLCopy

<span>Hello World</span>

**Razor code blocks**

Razor code blocks start with @ and are enclosed by {}. Unlike expressions, C# code inside code blocks isn't rendered. Code blocks and expressions in a view share the same scope and are defined in order:

CSHTMLCopy

@{

var quote = "The future depends on what you do today. - Mahatma Gandhi";

}

<p>@quote</p>

@{

quote = "Hate cannot drive out hate, only love can do that. - Martin Luther King, Jr.";

}

<p>@quote</p>

The code renders the following HTML:

HTMLCopy

<p>The future depends on what you do today. - Mahatma Gandhi</p>

<p>Hate cannot drive out hate, only love can do that. - Martin Luther King, Jr.</p>

In code blocks, declare [local functions](https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/local-functions) with markup to serve as templating methods:

CSHTMLCopy

@{

void RenderName(string name)

{

<p>Name: <strong>@name</strong></p>

}

RenderName("Mahatma Gandhi");

RenderName("Martin Luther King, Jr.");

}

The code renders the following HTML:

HTMLCopy

<p>Name: <strong>Mahatma Gandhi</strong></p>

<p>Name: <strong>Martin Luther King, Jr.</strong></p>

**Implicit transitions**

The default language in a code block is C#, but the Razor Page can transition back to HTML:

CSHTMLCopy

@{

var inCSharp = true;

<p>Now in HTML, was in C# @inCSharp</p>

}

**Explicit delimited transition**

To define a subsection of a code block that should render HTML, surround the characters for rendering with the Razor <text> tag:

CSHTMLCopy

@for (var i = 0; i < people.Length; i++)

{

var person = people[i];

<text>Name: @person.Name</text>

}

Use this approach to render HTML that isn't surrounded by an HTML tag. Without an HTML or Razor tag, a Razor runtime error occurs.

The <text> tag is useful to control whitespace when rendering content:

* Only the content between the <text> tag is rendered.
* No whitespace before or after the <text> tag appears in the HTML output.

**Explicit line transition**

To render the rest of an entire line as HTML inside a code block, use @: syntax:

CSHTMLCopy

@for (var i = 0; i < people.Length; i++)

{

var person = people[i];

@:Name: @person.Name

}

Without the @: in the code, a Razor runtime error is generated.

Extra @ characters in a Razor file can cause compiler errors at statements later in the block. These extra @ compiler errors:

* Can be difficult to understand because the actual error occurs before the reported error.
* Is common after combining multiple implicit and explicit expressions into a single code block.

**Conditional attribute rendering**

Razor automatically omits attributes that aren't needed. If the value passed in is null or false, the attribute isn't rendered.

For example, consider the following razor:

CSHTMLCopy

<div class="@false">False</div>

<div class="@null">Null</div>

<div class="@("")">Empty</div>

<div class="@("false")">False String</div>

<div class="@("active")">String</div>

<input type="checkbox" checked="@true" name="true" />

<input type="checkbox" checked="@false" name="false" />

<input type="checkbox" checked="@null" name="null" />

The preceding Razor markup generates the following HTML:

HTMLCopy

<div>False</div>

<div>Null</div>

<div class="">Empty</div>

<div class="false">False String</div>

<div class="active">String</div>

<input type="checkbox" checked="checked" name="true">

<input type="checkbox" name="false">

<input type="checkbox" name="null">

**Control structures**

Control structures are an extension of code blocks. All aspects of code blocks (transitioning to markup, inline C#) also apply to the following structures:

**Conditionals @if, else if, else, and @switch**

@if controls when code runs:

CSHTMLCopy

@if (value % 2 == 0)

{

<p>The value was even.</p>

}

else and else if don't require the @ symbol:

CSHTMLCopy

@if (value % 2 == 0)

{

<p>The value was even.</p>

}

else if (value >= 1337)

{

<p>The value is large.</p>

}

else

{

<p>The value is odd and small.</p>

}

The following markup shows how to use a switch statement:

CSHTMLCopy

@switch (value)

{

case 1:

<p>The value is 1!</p>

break;

case 1337:

<p>Your number is 1337!</p>

break;

default:

<p>Your number wasn't 1 or 1337.</p>

break;

}

**Looping @for, @foreach, @while, and @do while**

Templated HTML can be rendered with looping control statements. To render a list of people:

CSHTMLCopy

@{

var people = new Person[]

{

new Person("Weston", 33),

new Person("Johnathon", 41),

...

};

}

The following looping statements are supported:

@for

CSHTMLCopy

@for (var i = 0; i < people.Length; i++)

{

var person = people[i];

<p>Name: @person.Name</p>

<p>Age: @person.Age</p>

}

@foreach

CSHTMLCopy

@foreach (var person in people)

{

<p>Name: @person.Name</p>

<p>Age: @person.Age</p>

}

@while

CSHTMLCopy

@{ var i = 0; }

@while (i < people.Length)

{

var person = people[i];

<p>Name: @person.Name</p>

<p>Age: @person.Age</p>

i++;

}

@do while

CSHTMLCopy

@{ var i = 0; }

@do

{

var person = people[i];

<p>Name: @person.Name</p>

<p>Age: @person.Age</p>

i++;

} while (i < people.Length);

**Compound @using**

In C#, a using statement is used to ensure an object is disposed. In Razor, the same mechanism is used to create HTML Helpers that contain additional content. In the following code, HTML Helpers render a <form> tag with the @using statement:

CSHTMLCopy

@using (Html.BeginForm())

{

<div>

Email: <input type="email" id="Email" value="">

<button>Register</button>

</div>

}

**@try, catch, finally**

Exception handling is similar to C#:

CSHTMLCopy

@try

{

throw new InvalidOperationException("You did something invalid.");

}

catch (Exception ex)

{

<p>The exception message: @ex.Message</p>

}

finally

{

<p>The finally statement.</p>

}

**@lock**

Razor has the capability to protect critical sections with lock statements:

CSHTMLCopy

@lock (SomeLock)

{

// Do critical section work

}

**Comments**

Razor supports C# and HTML comments:

CSHTMLCopy

@{

/\* C# comment \*/

// Another C# comment

}

<!-- HTML comment -->

The code renders the following HTML:

HTMLCopy

<!-- HTML comment -->

Razor comments are removed by the server before the webpage is rendered. Razor uses @\* \*@ to delimit comments. The following code is commented out, so the server doesn't render any markup:

CSHTMLCopy

@\*

@{

/\* C# comment \*/

// Another C# comment

}

<!-- HTML comment -->

\*@

**Directives**

Razor directives are represented by implicit expressions with reserved keywords following the @ symbol. A directive typically changes the way a view is compiled or functions.

Understanding how Razor generates code for a view makes it easier to understand how directives work.

CSHTMLCopy

@{

var quote = "Getting old ain't for wimps! - Anonymous";

}

<div>Quote of the Day: @quote</div>

The code generates a class similar to the following:

C#Copy

public class \_Views\_Something\_cshtml : RazorPage<dynamic>

{

public override async Task ExecuteAsync()

{

var output = "Getting old ain't for wimps! - Anonymous";

WriteLiteral("/r/n<div>Quote of the Day: ");

Write(output);

WriteLiteral("</div>");

}

}

Later in this article, the section [Inspect the Razor C# class generated for a view](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-8.0#inspect-the-razor-c-class-generated-for-a-view) explains how to view this generated class.

**@attribute**

The @attribute directive adds the given attribute to the class of the generated page or view. The following example adds the [Authorize] attribute:

CSHTMLCopy

@attribute [Authorize]

The @attribute directive can also be used to supply a constant-based route template in a Razor component. In the following example, the @page directive in a component is replaced with the @attribute directive and the constant-based route template in Constants.CounterRoute, which is set elsewhere in the app to "/counter":

diffCopy

- @page "/counter"

+ @attribute [Route(Constants.CounterRoute)]

**@code**

*This scenario only applies to Razor components (.razor).*

The @code block enables a [Razor component](https://learn.microsoft.com/en-us/aspnet/core/blazor/components/?view=aspnetcore-8.0) to add C# members (fields, properties, and methods) to a component:

razorCopy

@code {

// C# members (fields, properties, and methods)

}

For Razor components, @code is an alias of [@functions](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-8.0#functions) and recommended over @functions. More than one @code block is permissible.

**@functions**

The @functions directive enables adding C# members (fields, properties, and methods) to the generated class:

CSHTMLCopy

@functions {

// C# members (fields, properties, and methods)

}

In [Razor components](https://learn.microsoft.com/en-us/aspnet/core/blazor/components/?view=aspnetcore-8.0), use @code over @functions to add C# members.

For example:

CSHTMLCopy

@functions {

public string GetHello()

{

return "Hello";

}

}

<div>From method: @GetHello()</div>

The code generates the following HTML markup:

HTMLCopy

<div>From method: Hello</div>

The following code is the generated Razor C# class:

C#Copy

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc.Razor;

public class \_Views\_Home\_Test\_cshtml : RazorPage<dynamic>

{

// Functions placed between here

public string GetHello()

{

return "Hello";

}

// And here.

#pragma warning disable 1998

public override async Task ExecuteAsync()

{

WriteLiteral("\r\n<div>From method: ");

Write(GetHello());

WriteLiteral("</div>\r\n");

}

#pragma warning restore 1998

@functions methods serve as templating methods when they have markup:

CSHTMLCopy

@{

RenderName("Mahatma Gandhi");

RenderName("Martin Luther King, Jr.");

}

@functions {

private void RenderName(string name)

{

<p>Name: <strong>@name</strong></p>

}

}

The code renders the following HTML:

HTMLCopy

<p>Name: <strong>Mahatma Gandhi</strong></p>

<p>Name: <strong>Martin Luther King, Jr.</strong></p>

**@implements**

The @implements directive implements an interface for the generated class.

The following example implements [System.IDisposable](https://learn.microsoft.com/en-us/dotnet/api/system.idisposable) so that the [Dispose](https://learn.microsoft.com/en-us/dotnet/api/system.idisposable.dispose) method can be called:

CSHTMLCopy

@implements IDisposable

<h1>Example</h1>

@functions {

private bool \_isDisposed;

...

public void Dispose() => \_isDisposed = true;

}

**@inherits**

The @inherits directive provides full control of the class the view inherits:

CSHTMLCopy

@inherits TypeNameOfClassToInheritFrom

The following code is a custom Razor page type:

C#Copy

using Microsoft.AspNetCore.Mvc.Razor;

public abstract class CustomRazorPage<TModel> : RazorPage<TModel>

{

public string CustomText { get; } =

"Gardyloo! - A Scottish warning yelled from a window before dumping" +

"a slop bucket on the street below.";

}

The CustomText is displayed in a view:

CSHTMLCopy

@inherits CustomRazorPage<TModel>

<div>Custom text: @CustomText</div>

The code renders the following HTML:

HTMLCopy

<div>

Custom text: Gardyloo! - A Scottish warning yelled from a window before dumping

a slop bucket on the street below.

</div>

@model and @inherits can be used in the same view. @inherits can be in a \_ViewImports.cshtml file that the view imports:

CSHTMLCopy

@inherits CustomRazorPage<TModel>

The following code is an example of a strongly-typed view:

CSHTMLCopy

@inherits CustomRazorPage<TModel>

<div>The Login Email: @Model.Email</div>

<div>Custom text: @CustomText</div>

If "rick@contoso.com" is passed in the model, the view generates the following HTML markup:

HTMLCopy

<div>The Login Email: rick@contoso.com</div>

<div>

Custom text: Gardyloo! - A Scottish warning yelled from a window before dumping

a slop bucket on the street below.

</div>

**@inject**

The @inject directive enables the Razor Page to inject a service from the [service container](https://learn.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-8.0) into a view. For more information, see [Dependency injection into views](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/dependency-injection?view=aspnetcore-8.0).

**@layout**

*This scenario only applies to Razor components (.razor).*

The @layout directive specifies a layout for routable Razor components that have an [@page](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-8.0#page) directive. Layout components are used to avoid code duplication and inconsistency. For more information, see [ASP.NET Core Blazor layouts](https://learn.microsoft.com/en-us/aspnet/core/blazor/components/layouts?view=aspnetcore-8.0).

**@model**

*This scenario only applies to MVC views and Razor Pages (.cshtml).*

The @model directive specifies the type of the model passed to a view or page:

CSHTMLCopy

@model TypeNameOfModel

In an ASP.NET Core MVC or Razor Pages app created with individual user accounts, Views/Account/Login.cshtml contains the following model declaration:

CSHTMLCopy

@model LoginViewModel

The class generated inherits from RazorPage<LoginViewModel>:

C#Copy

public class \_Views\_Account\_Login\_cshtml : RazorPage<LoginViewModel>

Razor exposes a Model property for accessing the model passed to the view:

CSHTMLCopy

<div>The Login Email: @Model.Email</div>

The @model directive specifies the type of the Model property. The directive specifies the T in RazorPage<T> that the generated class that the view derives from. If the @model directive isn't specified, the Model property is of type dynamic. For more information, see [Strongly typed models and the @model keyword](https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/adding-model?view=aspnetcore-8.0#strongly-typed-models-and-the--keyword).

**@namespace**

The @namespace directive:

* Sets the namespace of the class of the generated Razor page, MVC view, or Razor component.
* Sets the root derived namespaces of a pages, views, or components classes from the closest imports file in the directory tree, \_ViewImports.cshtml (views or pages) or \_Imports.razor (Razor components).

CSHTMLCopy

@namespace Your.Namespace.Here

For the Razor Pages example shown in the following table:

* Each page imports Pages/\_ViewImports.cshtml.
* Pages/\_ViewImports.cshtml contains @namespace Hello.World.
* Each page has Hello.World as the root of it's namespace.

Expand table

| **Page** | **Namespace** |
| --- | --- |
| Pages/Index.cshtml | Hello.World |
| Pages/MorePages/Page.cshtml | Hello.World.MorePages |
| Pages/MorePages/EvenMorePages/Page.cshtml | Hello.World.MorePages.EvenMorePages |

The preceding relationships apply to import files used with MVC views and Razor components.

When multiple import files have a @namespace directive, the file closest to the page, view, or component in the directory tree is used to set the root namespace.

If the EvenMorePages folder in the preceding example has an imports file with @namespace Another.Planet (or the Pages/MorePages/EvenMorePages/Page.cshtml file contains @namespace Another.Planet), the result is shown in the following table.

Expand table

| **Page** | **Namespace** |
| --- | --- |
| Pages/Index.cshtml | Hello.World |
| Pages/MorePages/Page.cshtml | Hello.World.MorePages |
| Pages/MorePages/EvenMorePages/Page.cshtml | Another.Planet |

**@page**

The @page directive has different effects depending on the type of the file where it appears. The directive:

* In a .cshtml file indicates that the file is a Razor Page. For more information, see [Custom routes](https://learn.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-8.0#custom-routes) and [Introduction to Razor Pages in ASP.NET Core](https://learn.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-8.0).
* Specifies that a Razor component should handle requests directly. For more information, see [ASP.NET Core Blazor routing and navigation](https://learn.microsoft.com/en-us/aspnet/core/blazor/fundamentals/routing?view=aspnetcore-8.0).

**@preservewhitespace**

*This scenario only applies to Razor components (.razor).*

When set to false (default), whitespace in the rendered markup from Razor components (.razor) is removed if:

* Leading or trailing within an element.
* Leading or trailing within a RenderFragment parameter. For example, child content passed to another component.
* It precedes or follows a C# code block, such as @if or @foreach.

**@rendermode**

*This scenario only applies to Razor components (.razor).*

Sets the render mode of a Razor component:

* InteractiveServer: Applies interactive server rendering using Blazor Server.
* InteractiveWebAssembly: Applies interactive WebAssembly rendering using Blazor WebAssembly.
* InteractiveAuto: Initially applies interactive WebAssembly rendering using Blazor Server, and then applies interactive WebAssembly rendering using WebAssembly on subsequent visits after the Blazor bundle is downloaded.

For a component instance:

razorCopy

<... @rendermode="InteractiveServer" />

In the component definition:

razorCopy

@rendermode InteractiveServer

**Note**

Blazor templates include a static using directive for [**RenderMode**](https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.components.web.rendermode) in the app's \_Imports file (Components/\_Imports.razor) for shorter @rendermode syntax:

razorCopy

@using static Microsoft.AspNetCore.Components.Web.RenderMode

Without the preceding directive, components must specify the static [**RenderMode**](https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.components.web.rendermode) class in @rendermode syntax explicitly:

razorCopy

<Dialog @rendermode="RenderMode.InteractiveServer" />

For more information, including guidance on disabling prerendering with the directive/directive attribute, see [ASP.NET Core Blazor render modes](https://learn.microsoft.com/en-us/aspnet/core/blazor/components/render-modes?view=aspnetcore-8.0).

**@section**

*This scenario only applies to MVC views and Razor Pages (.cshtml).*

The @section directive is used in conjunction with [MVC and Razor Pages layouts](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/layout?view=aspnetcore-8.0) to enable views or pages to render content in different parts of the HTML page. For more information, see [Layout in ASP.NET Core](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/layout?view=aspnetcore-8.0).

**@typeparam**

*This scenario only applies to Razor components (.razor).*

The @typeparam directive declares a [generic type parameter](https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/generics/generic-type-parameters) for the generated component class:

razorCopy

@typeparam TEntity

Generic types with [where](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/where-generic-type-constraint) type constraints are supported:

razorCopy

@typeparam TEntity where TEntity : IEntity

For more information, see the following articles:

* [ASP.NET Core Razor component generic type support](https://learn.microsoft.com/en-us/aspnet/core/blazor/components/generic-type-support?view=aspnetcore-8.0)
* [ASP.NET Core Blazor templated components](https://learn.microsoft.com/en-us/aspnet/core/blazor/components/templated-components?view=aspnetcore-8.0)

**@using**

The @using directive adds the C# using directive to the generated view:

CSHTMLCopy

@using System.IO

@{

var dir = Directory.GetCurrentDirectory();

}

<p>@dir</p>

In [Razor components](https://learn.microsoft.com/en-us/aspnet/core/blazor/components/?view=aspnetcore-8.0), @using also controls which components are in scope.

**Directive attributes**

Razor directive attributes are represented by implicit expressions with reserved keywords following the @ symbol. A directive attribute typically changes the way an element is compiled or functions.

**@attributes**

*This scenario only applies to Razor components (.razor).*

@attributes allows a component to render non-declared attributes. For more information, see [ASP.NET Core Blazor attribute splatting and arbitrary parameters](https://learn.microsoft.com/en-us/aspnet/core/blazor/components/splat-attributes-and-arbitrary-parameters?view=aspnetcore-8.0).

**@bind**

*This scenario only applies to Razor components (.razor).*

Data binding in components is accomplished with the @bind attribute. For more information, see [ASP.NET Core Blazor data binding](https://learn.microsoft.com/en-us/aspnet/core/blazor/components/data-binding?view=aspnetcore-8.0).

**@bind:culture**

*This scenario only applies to Razor components (.razor).*

Use the @bind:culture attribute with the [@bind](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-8.0#bind) attribute to provide a [System.Globalization.CultureInfo](https://learn.microsoft.com/en-us/dotnet/api/system.globalization.cultureinfo) for parsing and formatting a value. For more information, see [ASP.NET Core Blazor globalization and localization](https://learn.microsoft.com/en-us/aspnet/core/blazor/globalization-localization?view=aspnetcore-8.0#globalization).

**@formname**

*This scenario only applies to Razor components (.razor).*

@formname assigns a form name to a Razor component's plain HTML form or a form based on [EditForm](https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.components.forms.editform) ([Editform documentation](https://learn.microsoft.com/en-us/aspnet/core/blazor/forms/binding?view=aspnetcore-8.0#editformeditcontext-model)). The value of @formname should be unique, which prevents form collisions in the following situations:

* A form is placed in a component with multiple forms.
* A form is sourced from an external class library, commonly a NuGet package, for a component with multiple forms, and the app author doesn't control the source code of the library to set a different external form name than a name used by another form in the component.

For more information and examples, see [ASP.NET Core Blazor forms overview](https://learn.microsoft.com/en-us/aspnet/core/blazor/forms/?view=aspnetcore-8.0).

**@on{EVENT}**

*This scenario only applies to Razor components (.razor).*

Razor provides event handling features for components. For more information, see [ASP.NET Core Blazor event handling](https://learn.microsoft.com/en-us/aspnet/core/blazor/components/event-handling?view=aspnetcore-8.0).

**@on{EVENT}:preventDefault**

*This scenario only applies to Razor components (.razor).*

Prevents the default action for the event.

**@on{EVENT}:stopPropagation**

*This scenario only applies to Razor components (.razor).*

Stops event propagation for the event.

**@key**

*This scenario only applies to Razor components (.razor).*

The @key directive attribute causes the components diffing algorithm to guarantee preservation of elements or components based on the key's value. For more information, see [Retain element, component, and model relationships in ASP.NET Core Blazor](https://learn.microsoft.com/en-us/aspnet/core/blazor/components/element-component-model-relationships?view=aspnetcore-8.0).

**@ref**

*This scenario only applies to Razor components (.razor).*

Component references (@ref) provide a way to reference a component instance so that you can issue commands to that instance. For more information, see [ASP.NET Core Razor components](https://learn.microsoft.com/en-us/aspnet/core/blazor/components/?view=aspnetcore-8.0#capture-references-to-components).

**Templated Razor delegates**

*This scenario only applies to MVC views and Razor Pages (.cshtml).*

Razor templates allow you to define a UI snippet with the following format:

CSHTMLCopy

@<tag>...</tag>

The following example illustrates how to specify a templated Razor delegate as a [Func<T,TResult>](https://learn.microsoft.com/en-us/dotnet/api/system.func-2). The [dynamic type](https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/types/using-type-dynamic) is specified for the parameter of the method that the delegate encapsulates. An [object type](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/object) is specified as the return value of the delegate. The template is used with a [List<T>](https://learn.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1) of Pet that has a Name property.

C#Copy

public class Pet

{

public string Name { get; set; }

}

CSHTMLCopy

@{

Func<dynamic, object> petTemplate = @<p>You have a pet named <strong>@item.Name</strong>.</p>;

var pets = new List<Pet>

{

new Pet { Name = "Rin Tin Tin" },

new Pet { Name = "Mr. Bigglesworth" },

new Pet { Name = "K-9" }

};

}

The template is rendered with pets supplied by a foreach statement:

CSHTMLCopy

@foreach (var pet in pets)

{

@petTemplate(pet)

}

Rendered output:

HTMLCopy

<p>You have a pet named <strong>Rin Tin Tin</strong>.</p>

<p>You have a pet named <strong>Mr. Bigglesworth</strong>.</p>

<p>You have a pet named <strong>K-9</strong>.</p>

You can also supply an inline Razor template as an argument to a method. In the following example, the Repeat method receives a Razor template. The method uses the template to produce HTML content with repeats of items supplied from a list:

CSHTMLCopy

@using Microsoft.AspNetCore.Html

@functions {

public static IHtmlContent Repeat(IEnumerable<dynamic> items, int times,

Func<dynamic, IHtmlContent> template)

{

var html = new HtmlContentBuilder();

foreach (var item in items)

{

for (var i = 0; i < times; i++)

{

html.AppendHtml(template(item));

}

}

return html;

}

}

Using the list of pets from the prior example, the Repeat method is called with:

* [List<T>](https://learn.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1) of Pet.
* Number of times to repeat each pet.
* Inline template to use for the list items of an unordered list.

CSHTMLCopy

<ul>

@Repeat(pets, 3, @<li>@item.Name</li>)

</ul>

Rendered output:

HTMLCopy

<ul>

<li>Rin Tin Tin</li>

<li>Rin Tin Tin</li>

<li>Rin Tin Tin</li>

<li>Mr. Bigglesworth</li>

<li>Mr. Bigglesworth</li>

<li>Mr. Bigglesworth</li>

<li>K-9</li>

<li>K-9</li>

<li>K-9</li>

</ul>

**Tag Helpers**

*This scenario only applies to MVC views and Razor Pages (.cshtml).*

There are three directives that pertain to [Tag Helpers](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/intro?view=aspnetcore-8.0).

Expand table

| **Directive** | **Function** |
| --- | --- |
| [@addTagHelper](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/intro?view=aspnetcore-8.0#add-helper-label) | Makes Tag Helpers available to a view. |
| [@removeTagHelper](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/intro?view=aspnetcore-8.0#remove-razor-directives-label) | Removes Tag Helpers previously added from a view. |
| [@tagHelperPrefix](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/intro?view=aspnetcore-8.0#prefix-razor-directives-label) | Specifies a tag prefix to enable Tag Helper support and to make Tag Helper usage explicit. |

**Razor reserved keywords**

**Razor keywords**

* page
* namespace
* functions
* inherits
* model
* section
* helper (Not currently supported by ASP.NET Core)

Razor keywords are escaped with @(Razor Keyword) (for example, @(functions)).

**C# Razor keywords**

* case
* do
* default
* for
* foreach
* if
* else
* lock
* switch
* try
* catch
* finally
* using
* while

C# Razor keywords must be double-escaped with @(@C# Razor Keyword) (for example, @(@case)). The first @ escapes the Razor parser. The second @ escapes the C# parser.

**Reserved keywords not used by Razor**

* class

**Inspect the Razor C# class generated for a view**

The [Razor SDK](https://learn.microsoft.com/en-us/aspnet/core/razor-pages/sdk?view=aspnetcore-8.0) handles compilation of Razor files. By default, the generated code files aren't emitted. To enable emitting the code files, set the EmitCompilerGeneratedFiles directive in the project file (.csproj) to true:

XMLCopy

<PropertyGroup>

<EmitCompilerGeneratedFiles>true</EmitCompilerGeneratedFiles>

</PropertyGroup>

When building a 6.0 project (net6.0) in the Debug build configuration, the Razor SDK generates an obj/Debug/net6.0/generated/ directory in the project root. Its subdirectory contains the emitted Razor page code files.

**View lookups and case sensitivity**

The Razor view engine performs case-sensitive lookups for views. However, the actual lookup is determined by the underlying file system:

* File based source:
  + On operating systems with case insensitive file systems (for example, Windows), physical file provider lookups are case insensitive. For example, return View("Test") results in matches for /Views/Home/Test.cshtml, /Views/home/test.cshtml, and any other casing variant.
  + On case-sensitive file systems (for example, Linux, OSX, and with EmbeddedFileProvider), lookups are case-sensitive. For example, return View("Test") specifically matches /Views/Home/Test.cshtml.
* Precompiled views: With ASP.NET Core 2.0 and later, looking up precompiled views is case insensitive on all operating systems. The behavior is identical to physical file provider's behavior on Windows. If two precompiled views differ only in case, the result of lookup is non-deterministic.

Developers are encouraged to match the casing of file and directory names to the casing of:

* Area, controller, and action names.
* Razor Pages.

Matching case ensures the deployments find their views regardless of the underlying file system.

**Imports used by Razor**

The following imports are generated by the ASP.NET Core web templates to support Razor Files:

Copy

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.AspNetCore.Mvc.ViewFeatures;