ASP.NET: Razor Syntax

Razor Syntax

Razor Syntax allows you to embed code (C#) into page views through the use of a few keywords (such as “@”), and then have the C# code be processed and converted at runtime to HTML. In other words, rather than coding static HTML syntax in the page view, a user can code in the view in C# and have the Razor engine convert the C# code into HTML at runtime, creating a dynamically generated HTML web page.

```
@page
@model IndexModel
<h2>Welcome</h2>

<ul>
  @for (int i = 0; i < 3; i++)
  {
    <li>@i</li>
  }
</ul>
```

The @page Directive

In a Razor view page (**.cshtml**), the `@page` directive indicates that the file is a Razor Page.
In order for the page to be treated as a Razor Page, and have ASP.NET parse the view syntax with the Razor engine, the directive `@page` should be added at the top of the file.
There can be empty space before the `@page` directive, but there cannot be any other characters, even an empty code block.

```
@page
@model IndexModel

<h1>Welcome to my Razor Page</h1>
<p>Title: @Model.Title</p>
```

The @model Directive

The page model class, i.e. the data and methods that hold the functionality associated with a view page, is made available to the view page via the `@model` directive.
By specifying the model in the view page, Razor exposes a `Model` property for accessing the model passed to the view page. We can then access properties and functions from that model by using the keyword `Model` or render its property values on the browser by prefixing the property names with `@Model`, e.g. `@Model.PropertyName`.

```
@page
@model PersonModel

// Rendering the value of FirstName in PersonModel
<p>@Model.FirstName</p>

<ul>
  // Accessing the value of FavoriteFoods in PersonModel
  @foreach (var food in Model.FavoriteFoods)
  {
    <li>@food</li>
  }
</ul>
```

Razor Markup

Razor pages use the @ symbol to transition from HTML to C#. C# expressions are evaluated and then rendered in the HTML output. You can use Razor syntax under the following conditions:

1. Anything immediately following the @ is assumed to be C# code.
2. Code blocks must appear within @{ ... } brackets.
3. A single line of code that uses spaces should be surrounded by parentheses, ( ).

```
@page
@model PersonModel

// Using the `@` symbol:
<h1>My name is @Model.FirstName and I am @Model.Age years old
</h1>

// Using a code block:
@{
  var greet = "Hey threre!";
  var name = "John";
  <h1>@greet I'm @name!</h1>
}

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

Razor Conditionals

Conditionals in Razor code can be written pretty much the same way you would in regular C# code. The only exception is to prefix the keyword `if` with the `@` symbol. Afterward, any `else` or `else if` conditions doesn't need to be preprended with the `@` symbol.

```
// if-else if-else statment:
@{ var time = 9; }

@if (time < 10)
{
  <p>Good morning, the time is: @time</p>
}
else if (time < 20)
{
  <p>Good day, the time is: @time</p>
}
else
{
  <p>Good evening, the time is: @time</p>
}
```

Razor Switch Statements

In Razor Pages, a switch statement begins with the `@` symbol followed by the keyword `switch`. The condition is then written in parentheses and finally the switch cases are written within curly brackets, `{}`.

```
@{ string day = "Monday"; }
@switch (day)
{
  case "Saturday":
    <p>Today is Saturday</p>
    break;
  case "Sunday":
    <p>Today is Sunday</p>
    break;
  default:
    <p>Today is @day... Looking forward to the weekend</p>
    break;
}
```

Razor For Loops

In Razor Pages, a `for` loop is prepended by the `@` symbol followed by a set of conditions wrapped in parentheses. The `@` symbol must be used when referring to C# code.

```
@{
  List<string> avengers = new List<string>()
  {
    "Spiderman",
    "Iron Man",
    "Hulk",
    "Thor",
  };
}


<h1>The Avengers Are:</h1>

@for (int i = 0; i < @avengers.Count; i++)
{
  <p>@avengers[i]</p>
}
```

Razor Foreach Loops

In Razor Pages, a `foreach` loop is prepended by the `@` symbol followed by a set of conditions wrapped in parentheses. Within the conditions, we can create a variable that will be used when rendering its value on the browser.

```
@{
  List<string> avengers = new List<string>()
  {
    "Spiderman",
    "Iron Man",
    "Hulk",
    "Thor",
  };
}


<h1>The Avengers Are:</h1>

@foreach (var avenger in avengers)
{
    <p>@avenger</p>
}
```

Razor While Loops

A while loop repeats the execution of a sequence of statements as long as a set of conditions is true , once the condition becomes false we break out of the loop. When writing a while loop, we must prepend the keyword while with the @ symbol and write the condition within parentheses.

```
@{ int i = 0; }


@while (i < 5)
{
  <p>@i</p>
  i++;
}
```

Razor View Data

In Razor Pages, you can use the ViewData property to pass data from a Page Model to its corresponding view page, as well as share it with the layout, and any partial views.
ViewData is a dictionary that can contain key-value pairs where each key must be a string. The values can be accessed in the view page using the @ symbol.
A huge benefit of using ViewData comes when working with layout pages. We can easily pass information from each individual view page such as the title , into the layout by storing it in the ViewData dictionary in a view page:
@{ ViewData["Title"] = "Homepage" }
We can then access it in the layout like so: ViewData["Title"] . This way, we don’t need to hardcode certain information on each individual view page.

```
// Page Model: Index.cshtml.cs
public class IndexModel : PageModel
{
  public void OnGet()
  {
    ViewData["Message"] = "Welcome to my page!";
    ViewData["Date"] = DateTime.Now();
  }
}

// View Page: Index.cshtml
@page
@model IndexModel

<h1>@ViewData["Message"]</h1>
<h2>Today is: @ViewData["Date"]</h2>
```

Razor Shared Layouts

In Razor Pages, you can reduce code duplication by sharing layouts between view pages. A default layout is set up for your application in the **_Layout.cshtml** file located in **Pages/Shared/**.
Inside the **_Layout.cshtml** file there is a method call: `RenderBody()` . This method specifies the point at which the content from the view page is rendered relative to the layout defined.
If you want your view page to use a specific Layout page you can define it at the top by specifying the filename without the file extension: `@{ Layout = "LayoutPage" }`

```
// Layout: _LayoutExample.cshtml
<body>
    ...
<div class="container body-content">
  @RenderBody()
  <footer>
    <p>@DateTime.Now.Year - My ASP.NET Application</p>
  </footer>
</div>
</body>

// View Page: Example.cshtml
@page
@model ExampleModel

@{ Layout = "_LayoutExample" }

<h1>This content will appear where @RenderBody is called!
</h1>
```

Razor Tag Helpers

In Razor Pages, Tag Helpers change and enhance existing HTML elements by adding specific attributes to them. The elements they target are based on the element name, the attribute name, or the parent tag.
ASP.NET provides us with numerous built-in Tag Helpers that can be used for common tasks - such as creating forms, links, loading assets, and more.

```
// Page Model: Example.cshtml.cs
public class ExampleModel : PageModel
{
  public string Language { get; set; }

  public List<SelectListItem> Languages { get; } = new
List<SelectListItem>
  {
    new SelectListItem { Value = "C#", Text = "C#" },
    new SelectListItem { Value = "Javascript", Text =
"Javascript" },
    new SelectListItem { Value = "Ruby", Text = "Ruby"  },
  };
}

// View Page: Example.cshtml
<h1>Select your favorite language!</h1>
<form method="post">
  // asp-for:  The name of the specified model property.
  // asp-items: A collection of SelectListItemoptions that
appear in the select list.
  <select asp-for="Language" asp-items="Model.Languages">
</select>
  <br />
  <button type="submit">Register</button>
</form>
```

```
// HTML Rendered:
<form method="post">
  <select id="Language" name="Language">
    <option value="C#">C#</option>
    <option value="Javascript">Javascript</option>
    <option value="Ruby">Ruby</option>
    <br>
  </select>
  <button type="submit">Register</button>
</form>
```

Razor View Start File

When creating a template with ASP.NET, a **ViewStart.cshtml** file is automatically generated under the **/Pages** folder.
The **ViewStart.cshtml** file is generally used to define the layout for the website but can be used to define common view code that you want to execute at the start of each View's rendering. The generated file contains code to set up the main layout for the application.

```
// ViewStart.cshtml
@{
  Layout: "_Layout"
}
```

Razor Partials

Partial views are an effective way of breaking up large views into smaller components and reduce complexity. A partial consists of fragments of HTML and server-side code to be included in any number of pages or layouts.
We can use the Partial Tag Helper, `<partial>` , in order to render a partial's content in a view page.

```
// _MyPartial.cshtml
<form method="post">
  <input type="email" name="emailaddress">
  <input type="submit">
</form>

// Example.cshtml
<h1> Welcome to my page! </h1>
<h2> Fill out the form below to subscribe!:</h2>
<partial name="_MyPartial" />
```

Razor View Imports

The **_ViewImports.cshtml** file is automatically generated under **/Pages** when we create a template with ASP.NET.
Just like the **_ViewStart.cshtml** file, **_ViewImports.cshtml** is invoked for all your view pages before they are rendered.
The purpose of the file is to write common directives that our view pages need.
ASP.NET currently supports a few directives that can be added such as: `@namespace` , `@using` , `@addTagHelpers` , and `@inject` amongst a few other ones. Instead of having to add them individually to each page, we can place the directives here and they'll be available globally throughout the application.

```
// _ViewImports.cshtml
@using YourProject
@namespace YourProject.Pages
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```