Layout Pages

Most sites feature the same content on every page, or within a large number of pages. Headers, footers, and navigation systems are just some examples. Site-wide scripts and style sheets also fall into this category. Adding the same header to every page in your site breaks the DRY principle (Don't Repeat Yourself). If you need to change the appearance of the header, you need to edit every page. The same applies to other common content, if you want to upgrade your client-side framework, for example. Some IDEs include tools for making replacements in multiple files, but that's not really a robust solution. The proper solution to this problem is the **Layout** page.

The layout page acts as a template for all pages that reference it. The pages that reference the layout page are called content pages. Content pages are not full web pages. They contain only the content that varies from one page to the next. The code example below illustrates a very simple layout page:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8" />
        <title></title>
        <link href="/css/site.css" rel="stylesheet" type="text/css" />
    </head>
    <body>
        @RenderBody()
    </body>
</html>
```

What makes this a layout page is the call to the `RenderBody` method. That is where the result from processing the content page will be placed. Content pages reference their layout page via the `Layout` property of the page, which can be assigned in a code block at the top of a content page to point to a relative location:

```
@{
    Layout = "/_Layout";
}
```

Layout pages are typically named *_Layout.cshtml*, the leading underscore preventing them from being browsed directly. Inclusion of the file extension is optional when supplying their location to the `Layout` property. Standard practice is to specify the location of the layout page in a *_ViewStart.cshtml* file, which affects all content pages in the folder in which it is placed, and all subfolders.

By default, the layout file is placed in the *Pages/Shared* folder, but it can be placed anywhere in the application folder structure. Use of the *_ViewStart* file to centralise the location of the layout makes updating to the new location easy:

```
@{
    Layout = "Themes/MyTheme/_Layout";
}
```

You can also specify the location of the layout in the Razor Page itself. This will override the instruction set in the *_ViewStart* file. If you don't want page to use the layout specified in the *_ViewStart* file, you can pass null to the `Layout` property:

```
@{
    Layout = null;
}
```

You might do this if your page is used to return XML, for example.

Locating a Layout

You do not need to provide a full file path for the layout file. The Razor Pages framework searches a set of predefined locations if you only provide the file name to the `Layout` property:

```
@{
    Layout = "_Layout";
}
```

The framework searches by walking up the directory tree from the location of the calling page looking for the file name that you pass in as long as you do not include the file extension, until it reaches the root *Pages* folder. Once this has been exhausted, the formally registered locations are searched. The default registered search paths are Pages/Shared (from ASP.NET Core 2.1 onwards) and Views/Shared (the default location for layout pages in an MVC application).

If the calling page is located in *Pages/Orders* the search for a layout named *_Layout.cshtml* will include the following locations:

```
Pages/Orders/_Layout.cshtml
Pages/_Layout.cshtml
Pages/Shared/_Layout.cshtml
Views/Shared/_Layout.cshtml
```

If the page calling the layout is located in an area, the search will also start in the currently executing page's folder, and then walk up the directory tree within the area. Once the area folder structure has been exhausted, registered layout locations are searched relative to the area's folder location (i.e. *Pages/Shared* and *Views/Shared* within the area). Finally, the registered locations themselves are searched.

The following search locations assume that the calling page in located at *Areas/Orders/Pages/Archive/Index.cshtml*:

```
Areas/Orders/Pages/Archive/_Layout.cshtml
Areas/Orders/Pages/_Layout.cshtml
Areas/Orders/Pages/Shared/_Layout.cshtml
Areas/Orders/Views/Shared/_Layout.cshtml
Pages/Shared/_Layout.cshtml
Views/Shared/_Layout.cshtml
```

This is the same discovery process as is used for **partial pages (/razor-pages/partial-pages)**.

Sections

The `RenderBody` method placement within the layout page determines where the content page will be rendered, but it is also possible to render other content supplied by the content page within a layout page. This is controlled by the placement of calls to the `RenderSection` method. The following example of a call to this method is taken from the layout page that forms part of the default template Razor Pages site:

```
@RenderSection("Scripts", required: false)
```

This call defines a section named "Scripts" - intended for page-specific script file references or blocks of JavaScript code so that they can be located just before the closing `</body>` tag. The second argument, `required` determines whether the content page must provide content for the named section. In this example, `required` is set to `false`, resulting in the section being optional. If the section is not optional, every content page that references the layout page must use the `@section` syntax to provide content for the section:

```
@section Scripts{
    // content here
}
```

In some cases, you might want to make a section optional, but you want to provide some default content in the event that the content page didn't provide anything for the section. You can use the `IsSectionDefined` method for this:

```
@if(IsSectionDefined("OptionalSection"))
{
    @RenderSection("OptionalSection")
}
else
{
    // default content
}
```

Nested Layouts

Layout pages can be nested, that is, it is perfectly legal to specify the layout for a layout page. The following example shows a master layout which contains the head and style references, and two sub-layout pages. One has a single column for content and the other has two columns, the second of which contains a section. Content pages can reference either of the two sub-layout pages and still benefit from the common markup provided by the master layout file.

_MasterLayout.cshtml

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8" />
        <title></title>
        <link href="/css/site.css" rel="stylesheet" type="text/css" />
    </head>
    <body>
        @RenderBody()
    </body>
</html>
```

SubLayout1.cshtml

```
@{
    Layout = "/_MasterLayout";
}
<div class="main-content-one-col">
@RenderBody()
</div>
```

SubLayout2.cshtml

```
@{
    Layout = "/_MasterLayout";
}
<div class="main-content-two-col">
@RenderBody()
</div>
<div>
@RenderSection("RightCol")
</div>
```

Any sections defined in the master layout should also be redefined in child layouts:

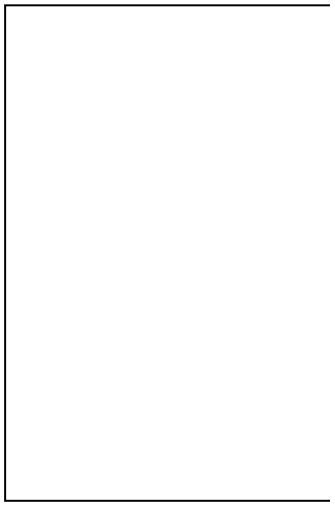
_MasterLayout.cshtml

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8" />
        <title></title>
        <link href="/css/site.css" rel="stylesheet" type="text/css" />
    </head>
    <body>
        @RenderBody()
        @RenderSection("scripts", required:false)
    </body>
</html>
```

_ChildLayout.cshtml

```
@{
    Layout = "/_MasterLayout";
}
<div class="main-content-two-col">
@RenderBody()
</div>
@section scripts {
  @RenderSection("scripts", required: false)
}
```

Last updated: 13/11/2020 10:55:52



On this page

Latest Updates

Caching in Razor Pages (https://www.learnrazorpages.com/razor-pages/caching)

Middleware in Razor Pages (https://www.learnrazorpages.com/middleware)

Using Cookies in Razor Pages (https://www.learnrazorpages.com/razor-pages/cookies)

The Razor _Layout.cshtml file (https://www.learnrazorpages.com/razor-pages/files/layout)

An Introduction To ASP.NET Core Razor Pages (https://www.learnrazorpages.com/)

Publishing and deploying a Razor Pages application to IIS on Windows (https://www.learnrazorpages.com/publishing/publish-to-iis)

Contact me at Outlook.com