[1. Main Razor Syntax Rules for C# 2](#_Toc378254061)

[2. Content Blocks 2](#_Toc378254062)

[3. Using a Layout Page 2](#_Toc378254063)

[4. Preventing Files from Being Browsed 3](#_Toc378254064)

[5. Logical Folder Structure 3](#_Toc378254065)

[6. URLs and Paths 4](#_Toc378254066)

[7. Global Initializations 5](#_Toc378254067)

[8. The Page Object 6](#_Toc378254068)

[9. Working with Text Files 7](#_Toc378254069)

[10. Displaying Data from Database 8](#_Toc378254070)

# Main Razor Syntax Rules for C#

The page contains ordinary HTML markup, with one addition: the @ marked Razor code.

* Razor code blocks are enclosed in @{ ... }
* Inline expressions (variables and functions) start with @
* Code statements end with semicolon
* Variables are declared with the var keyword
* Strings are enclosed with quotation marks
* C# code is case sensitive
* C# files have the extension .cshtml

# Content Blocks

With Web Pages you can use the **@RenderPage()** method to import content from separate files.

Example

<html>  
<body>  
@RenderPage("header.cshtml")  
<h1>Hello Web Pages</h1>   
<p>This is a paragraph</p>  
@RenderPage("footer.cshtml")  
</body>  
</html>

# Using a Layout Page

The layout page is just like a normal web page, except from a call to the **@RenderBody()** method where the content page will be included.

Each content page must start with a **Layout directive**.

## Layout Page:

<html>  
<body>  
<p>This is header text</p>  
@RenderBody()  
<p>&copy; 2012 W3Schools. All rights reserved.</p>  
</body>  
</html>

## Any Web Page:

@{Layout="Layout.cshtml";}  
  
<h1>Welcome to W3Schools</h1>  
  
<p>  
Lorem ipsum dolor sit amet, consectetur adipisicing elit,sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laborisnisi ut aliquip ex ea commodo consequat.  
</p>

# Preventing Files from Being Browsed

With ASP.NET, files with a name that starts with an underscore cannot be browsed from the web. The common way to hide sensitive information (database passwords, email passwords, etc.) is to keep the information in a separate file named "\_AppStart".

## \_AppStart.cshtml

@{  
WebMail.SmtpServer = "mailserver.example.com";  
WebMail.EnableSsl = true;  
WebMail.UserName = "username@example.com";  
WebMail.Password = "your-password";  
WebMail.From = "your-name-here@example.com";  
}

# Logical Folder Structure

* The "Account" folder contains logon and security files
* The "App\_Data" folder contains databases and data files
* The "Images" folder contains images
* The "Scripts" folder contains browser scripts
* The "Shared" folder contains common files (like layout and style files)

From the example above:

The virtual name of a web picture might be "Images/pic31.jpg".

But the physical name is "C:\Johnny\Documents\MyWebSites\Demo\Images\pic31.jpg"

# URLs and Paths

The root on a disk drive is written like C:\, but the root on a web site is  / (forward slash).

The virtual path of a web folder is (almost) never the same as the physical folder.

In your code you will, reference both the physical path and the virtual path, depending on what you are coding.

ASP.NET has 3 tools for working with folder paths: the ~ operator, the Server.MapPath method, and the Href method.

The ~ Operator

To specify the virtual root in programming code, use the ~ operator.

If you use the ~ operator, instead of a path, you can move your website to a different folder or location without changing any code:

var myImagesFolder = "~/images";  
var myStyleSheet = "~/styles/StyleSheet.css";

The Server.MapPath Method

The Server.MapPath method converts a virtual path (/default.cshtml) to a physical path that the server can understand (C:\Johnny\MyWebSited\Demo\default.cshtml).

You will use this method when you need to open data files located on the server (data files can only be accessed with a full physical path):

var pathName = "~/dataFile.txt";  
var fileName = Server.MapPath(pathName);

The Href Method

The Href method converts a path used in the code to a path that the browser can understand (the browser cannot understand the ~ operator).

You use the Href method to create paths to resources like image files, and CSS files.

You will often use this method in HTML <a>, <img>, and <link> elements:

@{var myStyleSheet = "~/Shared/Site.css";}  
  
<!-- This creates a link to the CSS file. -->  
<link rel="stylesheet" type="text/css" href="@Href(myStyleSheet)" />  
  
<!-- Same as : -->  
<link rel="stylesheet" type="text/css" href="/Shared/Site.css" />

The Href method is a method of the WebPage Object.

# Global Initializations

**Before Web Startup: \_AppStart -**  by creating a page named \_AppStart in the root of your site, you can have startup code executed before the site starts. If this page exists, ASP.NET runs it the first time any page in the site is requested.

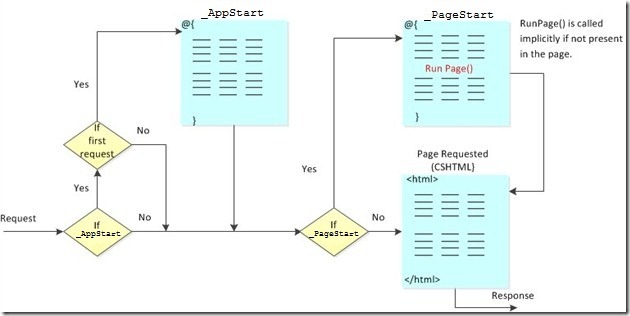
Typical use for \_AppStart is startup code and initialization of global values like counters and global names.

**Note 1:** \_AppStart should have the same file extension as your web pages, like: \_AppStart.cshtml.

**Note 2:**\_AppStart has an underscore prefix. Because of this, the files cannot be browsed directly.

**Before Every Page: \_PageStart** - For each folder in your web, you can add a file named \_PageStart.

Typical use for \_PageStart is setting the layout page for all pages in a folder, or checking that a user is logged in before running a page.



When a request comes in, ASP.NET checks whether \_AppStart exists. If so, and this is the first request to the site, \_AppStart runs.

Then ASP.NET checks whether \_PageStart exists. If so, \_PageStart runs, before the requested page.

If you include a call to RunPage() inside \_PageStart you specify where you want the requested page to run. If not, the \_PageStart runs before the requested page.

# The Page Object

Some Page Object Methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| href | Builds a URL using the specified parameters |
| RenderBody() | Renders the portion of a content page that is not within a named section (In layout pages) |
| RenderPage(*page*) | Renders the content of one page within another page |
| RenderSection(*section*) | Renders the content of a named section (In layout pages) |
| Write(*object*) | Writes the object as an HTML-encoded string |
| WriteLiteral | Writes an object without HTML-encoding it first. |

The Page property of the Page Object, provides property-like access to data shared between pages and layout pages.

You can use (add) your own properties to the Page property:

* Page.Title
* Page.Version
* Page.anythingyoulike

The pages property is very helpful. For instance, it makes it possible to set the page title in content files, and use it in the layout file

Home.cshtml

@{  
Layout="~/Shared/Layout.cshtml";  
Page.Title="Home Page"  
}  
  
<h1>Welcome to W3Schools</h1>   
  
<h2>Web Site Main Ingredients</h2>  
  
<p>A Home Page (Default.cshtml)</p>  
<p>A Layout File (Layout.cshtml)</p>  
<p>A Style Sheet (Site.css)</p>

Layout.cshtml

<!DOCTYPE html>  
<html>  
<head>  
<title>@Page.Title</title>  
</head>  
<body>  
@RenderBody()  
</body>  
</html

Some Page Object Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| isPost | Returns true if the HTTP data transfer method used by the client is a POST request |
| Layout | Gets or sets the path of a layout page |
| Page | Provides property-like access to data shared between pages and layout pages |
| Request | Gets the HttpRequest object for the current HTTP request |
| Server | Gets the HttpServerUtility object that provides web-page processing methods |

# Working with Text Files

## Example

Add the following content to the App\_data file:

## Persons.txt

George,Lucas  
Steven,Spielberg  
Alfred,Hitchcock

The example below shows how to display data from a text file:

@{  
var dataFile = Server.MapPath("~/App\_Data/Persons.txt");  
Array userData = File.ReadAllLines(dataFile);  
}  
  
<!DOCTYPE html>  
<html>  
<body>  
  
<h1>Reading Data from a File</h1>  
@foreach (string dataLine in userData)   
{  
  foreach (string dataItem in dataLine.Split(','))   
  {@dataItem <text>&nbsp;</text>}  
  <br />  
}  
</body>  
</html>

# Displaying Data from Database