

RSCG nr 15 : RazorBlade

Info

Nuget : <https://www.nuget.org/packages/RazorBlade/>

You can find more details at : <https://github.com/ltrzesniewski/RazorBlade>

Author : Lucas Trzesniewski


Source : <https://github.com/ltrzesniewski/RazorBlade>

About

Fast templating with Razor syntax Do not forget to put into AdditionalFiles section of csproj file

How to use

Add reference to the RazorBlade in the csproj




```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net7.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="RazorBlade" Version="0.4.3" PrivateAssets="all"
ReferenceOutputAssembly="false" OutputItemType="Analyzer" />
  </ItemGroup>
  <ItemGroup>
    <AdditionalFiles Include="PersonDisplay.cshtml" />
  </ItemGroup>
  <PropertyGroup>
    <EmitCompilerGeneratedFiles>true</EmitCompilerGeneratedFiles>

    <CompilerGeneratedFilesOutputPath>$(BaseIntermediateOutputPath)\GX</CompilerGeneratedF
ilesOutputPath>
  </PropertyGroup>
</Project>
```

This was for me the starting code

I have **coded** the file Program.cs



```
using RazorBladeDemo;

Console.WriteLine("Hello, World!");
Person p = new();
p.FirstName = "Andrei";
p.LastName = "Ignat";

var template = new PersonDisplay(p);
var result = template.Render();
Console.WriteLine(result);
```

I have **coded** the file Person.cs



```
• namespace RazorBladeDemo;  
  
public class Person  
{  
    public string? FirstName { get; set; }  
    public string? LastName { get; set; }  
    public string FullName()  
    {  
        return FirstName + " "+LastName;  
    }  
}
```

I have **coded** the file PersonDisplay.cshtml



```
@using RazorBladeDemo;  
@inherits RazorBlade.HtmlTemplate<Person>;  
  
This is the @Model.FirstName @Model.LastName  
  
<br />  
  
This should be full name of @Model.FullName( )
```

And here are the *generated* files

The file *generated* is Attributes.g.cs

```

•// This file is part of the RazorBlade library.

#nullable enable

using System;

namespace RazorBlade.Support;

/// <summary>
/// Specifies that this constructor needs to be provided by the generated template
/// class.
/// </summary>
[AttributeUsage(AttributeTargets.Constructor)]
internal sealed class TemplateConstructorAttribute : Attribute
{
}

/// <summary>
/// Specifies if a method should be used depending on the template being sync or
/// async.
/// </summary>
[AttributeUsage(AttributeTargets.Method)]
internal sealed class ConditionalOnAsyncAttribute : Attribute
{
    /// <summary>
    /// The message to display.
    /// </summary>
    public string? Message { get; set; }

    /// <summary>
    /// Marks a method as meant to be used in a sync or async template.
    /// </summary>
    /// <param name="async">True for methods meant to be used in async templates, and
    false for methods meant to be used for sync templates.</param>
    public ConditionalOnAsyncAttribute(bool async)
    {
    }
}

```

The file *generated* is HtmlHelper.g.cs

```

// This file is part of the RazorBlade library.

#nullable enable

using System;
using System.Diagnostics.CodeAnalysis;
using System.Text;

namespace RazorBlade;

// ReSharper disable once RedundantDisableWarningComment
#pragma warning disable CA1822

/// <summary>
/// Utilities for HTML Razor templates.
/// </summary>
[SuppressMessage("ReSharper", "MemberCanBeMadeStatic.Global")]
internal sealed class HtmlHelper
{
    internal static HtmlHelper Instance { get; } = new();

    /// <summary>
    /// Returns markup that is not HTML encoded.
    /// </summary>
    /// <param name="value">The HTML markup.</param>
    public HtmlString Raw(object? value)
        => new(value?.ToString());

    /// <summary>
    /// HTML-encodes the provided value.
    /// </summary>
    /// <param name="value">Value to HTML-encode.</param>
    public string Encode(object? value)
    {
        var valueString = value?.ToString();
        if (valueString is null or "")
            return string.Empty;

#if NET6_0_OR_GREATER
        var valueSpan = valueString.AsSpan();
        var sb = new StringBuilder();

        while (true)
        {
            var idx = valueSpan.IndexOfAny("<<\"'");
            if (idx < 0)
                break;

            if (idx != 0)
                sb.Append(valueSpan[..idx]);

            sb.Append(valueSpan[idx] switch
            {
                '&' => "&amp;",
                '<' => "&lt;",
                '>' => "&gt;",
                '"' => "&quot;",
                '\'' => "&#x27;",
                _ => c.ToString() // Won't happen
            });

            valueSpan = valueSpan[(idx + 1)..];
        }

        if (valueSpan.Length != 0)
            sb.Append(valueSpan);

        return sb.ToString();
#else
        return valueString.Replace("&", "&amp;")
            .Replace("<", "&lt;")
            .Replace(">", "&gt;")
            .Replace("\"", "&quot;")
            .Replace("'", "&#x27;");
#endif
    }
}

```

The file *generated* is `HtmlString.g.cs`

```
•// This file is part of the RazorBlade library.

#nullable enable

using System.IO;

namespace RazorBlade;

/// <summary>
/// Represents an HTML-encoded string that should not be encoded again.
/// </summary>
internal sealed class HtmlString : IEncodedContent
{
    private readonly string _value;

    /// <summary>
    /// Creates a HTML-encoded string.
    /// </summary>
    public HtmlString(string? value)
        => _value = value ?? string.Empty;

    /// <inheritdoc />
    public override string ToString()
        => _value;

    void IEncodedContent.WriteTo(TextWriter textWriter)
        => textWriter.Write(_value);
}
```

The file *generated* is HtmlTemplate.g.cs

[illegible]

The file *generated* is IEncodedContent.g.cs


```
•// This file is part of the RazorBlade library.

#nullable enable

using System.IO;

namespace RazorBlade;

/// <summary>
/// Encoded content to be written to the output as-is.
/// </summary>
internal interface IEncodedContent
{
    /// <summary>
    /// Writes the content to the provided <see cref="TextWriter"/>.
    /// </summary>
    /// <param name="textWriter"><see cref="TextWriter"/> to write the content to.
    </param>
    void WriteTo(TextWriter textWriter);
}
```

The file *generated* is PlainTextTemplate.g.cs

```

// This file is part of the RazorBlade library.

#nullable enable

using System;
using RazorBlade.Support;

namespace RazorBlade;

/// <summary>
/// Base class for plain text templates.
/// </summary>
/// <remarks>
/// Values will be written as-is, without escaping.
/// </remarks>
internal abstract class PlainTextTemplate : RazorTemplate
{
    private string? _currentAttributeSuffix;

    /// <inheritdoc />
    protected override void Write(object? value)
    {
        if (value is IEncodedContent encodedContent)
            encodedContent.WriteTo(Output);
        else
            Output.Write(value);
    }

    /// <inheritdoc />
    protected override void BeginWriteAttribute(string name, string prefix, int
    prefixOffset, string suffix, int suffixOffset, int attributeValuesCount)
    {
        WriteLiteral(prefix);
        _currentAttributeSuffix = suffix;
    }

    /// <inheritdoc />
    protected override void WriteAttributeValue(string prefix, int prefixOffset,
    object? value, int valueOffset, int valueLength, bool isLiteral)
    {
        WriteLiteral(prefix);

        if (isLiteral)
            WriteLiteral(value?.ToString());
        else
            Write(value);
    }

    /// <inheritdoc />
    protected override void EndWriteAttribute()
    {
        WriteLiteral(_currentAttributeSuffix);
        _currentAttributeSuffix = null;
    }
}

/// <summary>
/// Base class for plain text templates with a model.
/// </summary>
/// <remarks>
/// Values will be written as-is, without escaping.
/// </remarks>
/// <typeparam name="TModel">The model type.</typeparam>
internal abstract class PlainTextTemplate<TModel> : PlainTextTemplate
{
    /// <summary>
    /// The model for the template.
    /// </summary>
    public TModel Model { get; }

    /// <summary>
    /// Initializes a new instance of the template.
    /// </summary>
    /// <param name="model">The model for the template.</param>
    [TemplateConstructor]
    protected PlainTextTemplate(TModel model)
    {
        Model = model;
    }

    /// <summary>
    /// This constructor is provided for the designer only. Do not use.
    /// </summary>
    protected PlainTextTemplate()
    {
        throw new NotSupportedException("Use the constructor overload that takes a
    model.");
    }
}

```

The file *generated* is RazorTemplate.g.cs

```

•// This file is part of the RazorBlade library.

#nullable enable

using System.ComponentModel;
using System.IO;
using System.Threading;
using System.Threading.Tasks;
using RazorBlade.Support;

namespace RazorBlade;

/// <summary>
/// Base class for Razor templates.
/// </summary>
internal abstract class RazorTemplate : IEncodedContent
{
    /// <summary>
    /// The <see cref="TextWriter"/> which receives the output.
    /// </summary>
    protected TextWriter Output { get; set; } = new StreamWriter(Stream.Null);

    /// <summary>
    /// The cancellation token.
    /// </summary>
    protected CancellationToken CancellationToken { get; private set; }

    /// <summary>
    /// Renders the template synchronously and returns the result as a string.
    /// </summary>
    /// <param name="cancellationToken">The cancellation token.</param>
    /// <remarks>
    /// Use this only if the template does not use <c>@async</c> directives.
    /// </remarks>
    [ConditionalOnAsync(false, Message = $"The generated template is async. Use {nameof(RenderAsync)} instead.")]
    public string Render(CancellationToken cancellationToken = default)
    {
        cancellationToken.ThrowIfCancellationRequested();

        var renderTask = RenderAsync(cancellationToken);
        if (renderTask.IsCompleted)
            return renderTask.Result;

        return Task.Run(async () => await renderTask.ConfigureAwait(false),
            CancellationToken.None).GetAwaiter().GetResult();
    }

    /// <summary>
    /// Renders the template synchronously to the given <see cref="TextWriter"/>.
    /// </summary>
    /// <param name="textWriter">The <see cref="TextWriter"/> to write to.</param>
    /// <param name="cancellationToken">The cancellation token.</param>
    /// <remarks>
    /// Use this only if the template does not use <c>@async</c> directives.
    /// </remarks>
    [ConditionalOnAsync(false, Message = $"The generated template is async.

```

The file *generated* is RazorBladeDemo.PersonDisplay.Razor.g.cs

```

•#pragma checksum
"C:\test\RSCG_Examples\v2\rscg_examples\RazorBlade\src\RazorBladeDemo\PersonDisplay.cs
html" "{ff1816ec-aa5e-4d10-87f7-6f4963833460}"
"0ee9a5bcc623252570e9d97efdeb7e3c5a8d6350"
// <auto-generated/>
#pragma warning disable 1591
namespace RazorBladeDemo
{
    #line hidden
    #nullable restore
    #line 1
    "C:\test\RSCG_Examples\v2\rscg_examples\RazorBlade\src\RazorBladeDemo\PersonDisplay.cs
html"
    using RazorBladeDemo;

    #line default
    #line hidden
    #nullable disable
    #nullable restore
    internal partial class PersonDisplay : RazorBlade.HtmlTemplate<Person>
    #nullable disable
    {
        #pragma warning disable 1998
        protected async override global::System.Threading.Tasks.Task ExecuteAsync()
        {
            WriteLiteral("\r\nThis is the ");
        }
    #nullable restore
    #line (4,14)-(4,29) 6
    "C:\test\RSCG_Examples\v2\rscg_examples\RazorBlade\src\RazorBladeDemo\PersonDisplay.cs
html"
    Write(Model.FirstName);

    #line default
    #line hidden
    #nullable disable
        WriteLiteral(" ");
    #nullable restore
    #line (4,31)-(4,45) 6
    "C:\test\RSCG_Examples\v2\rscg_examples\RazorBlade\src\RazorBladeDemo\PersonDisplay.cs
html"
    Write(Model.LastName);

    #line default
    #line hidden
    #nullable disable
        WriteLiteral("\r\n\r\n<br />\r\n\r\nThis should be full name of ");
    #nullable restore
    #line (8,30)-(8,46) 6
    "C:\test\RSCG_Examples\v2\rscg_examples\RazorBlade\src\RazorBladeDemo\PersonDisplay.cs
html"
    Write(Model.FullName());

    #line default
    #line hidden
    #nullable disable
    }
    #pragma warning restore 1998
}
#pragma warning restore 1591

```

The file *generated* is RazorBladeDemo.PersonDisplay.RazorBlade.g.cs

```
• // <auto-generated/>

#nullable restore

namespace RazorBladeDemo
{
    partial class PersonDisplay
    {
        /// <inheritdoc cref="M:RazorBlade.HtmlTemplate`1.#ctor(`0)" />
        public PersonDisplay(global::RazorBladeDemo.Person model)
            : base(model)
        {
        }
    }
}
```

You can download the code and this page as pdf from https://ignatandrei.github.io/RSCG_Examples/v2/docs/RazorBlade

You can see the whole list at https://ignatandrei.github.io/RSCG_Examples/v2/docs/List-of-RSCG