Using Razor outside of MVC in .NET Core

Asked 3 years, 3 months ago Active 7 months ago Viewed 24k times

52

I would like to use Razor as a templating engine in a .NET console application that I'm writing in .NET Core.

The standalone Razor engines I've come across (RazorEngine, RazorTemplates) all require full .NET. I'm looking for a solution that works with .NET Core.

c# .net razor .net-core

28

edited Jul 7 '16 at 19:05

Nate Barbettini
32k 17 94 124

asked Jul 7 '16 at 13:42

Christof Jans
561 1 6 9

2 github.com/aspnet/Razor only requires the core runtime (using the .NET standard library) – haim770 Jul 7 '16 at 13:50

5 Answers

33

Recently I've created a library called RazorLight.

It has no redundant dependencies, like ASP.NET MVC parts and can be used in console applications. For now it only supports .NET Core (NetStandard1.6) - but that's exactly what you need.

Here is a short example:

```
IRazorLightEngine engine = EngineFactory.CreatePhysical("Path-to-your-views");
```

```
// Strings and anonymous models
string stringResult = engine.ParseString("Hello @Model.Name", new { Name = "John" });
```

edited Feb 21 '17 at 10:08

answered Jul 25 '16 at 13:16

Toddams

1,038 12 20

2 This was pretty easy to implement however it has pretty terrible performance. I created a loop that generated around a 1000 lines of html. It took around 12 seconds everytime. Just creating a single page with 200 lines took about 1-2 seconds. In an MVC project 1 page took about 20 milliseconds. So If you are not worried about performance this is a viable option. – DeadlyChambers Jul 7 '17 at 13:32

Well, if you use ParseString - templates are not cached, that's why you experience performance issues. Use Parse instead with appropriate template manager (for files or embedded resources) - this way template will only be compiled once and next time will be taken from cache. And you'll see the same numbers as in MVC project – Toddams Sep 19 '17 at 8:24

4 Update: 2.0 version caches templates built from strings – Toddams Dec 28 '17 at 11:25

2 @Toddams This is not going to work on production because views are precompiled on publish. Can you please add mvc project sample which support production(precompiled views) and development environment. I am not able to make it work for both environments together :((– Freshblood Jun 19 '18 at 13:52

@Toddams I'm in the same boat for the precompiled views; *RazorLight* is not working at all when we're using `dotnet publish` to build the app. – bchhun Apr 15 at 20:08

26

Here is a sample code that only depends on Razor (for parsing and C# code generation) and Roslyn (for C# code compilation, but you could use the old CodeDom as well).

There is no MVC in that piece of code, so, no View, no .cshtml files, no Controller, just Razor source parsing and compiled runtime execution. There is still the notion of Model though.

You will only need to add following nuget packages: `Microsoft.AspNetCore.Razor.Language` (v2.1.1), `Microsoft.AspNetCore.Razor.Runtime` (v2.1.1) and `Microsoft.CodeAnalysis.CSharp` (v2.8.2) nugets.

This C# source code is compatible with NETCore, NETStandard 2 and .NET Framework. To test it just create a .NET framework or .NET core console app, paste it, and add the nugets.

```
using System;
```

```
using Microsoft.AspNetCore.Razor.Hosting;
using Microsoft.AspNetCore.Razor.Language;
using Microsoft.AspNetCore.Razor.Language.Extensions;
using Microsoft.CodeAnalysis;
using Microsoft.CodeAnalysis.CSharp;

namespace RazorTemplate
{
    class Program
    {
        static void Main(string[] args)
        {
            // points to the local path
            var fs = RazorProjectFileSystem.Create(".");

            // customize the default engine a little bit
            var engine = RazorProjectEngine.Create(RazorConfiguration.Default, fs,
(builder) =>
            {
                InheritsDirective.Register(builder);
                builder.SetNamespace("MyNamespace"); // define a namespace for the
Template class
            });

            // get a razor-templated file. My "hello.txt" template file is defined like
this:
            //
            // @inherits RazorTemplate.MyTemplate
            // Hello @Model.Name, welcome to Razor World!
            //

            var item = fs.GetItem("hello.txt");

            // parse and generate C# code, outputs it on the console
            //var cs = te.GenerateCode(item);
            //Console.WriteLine(cs.GeneratedCode);

            var codeDocument = engine.Process(item);
            var cs = codeDocument.GetCSharpDocument();

            // now, use roslyn, parse the C# code
            var tree = CSharpSyntaxTree.ParseText(cs.GeneratedCode);

            // define the dll
            const string dllName = "hello";
            var compilation = CSharpCompilation.Create(dllName, new[] { tree },
```

```
        // include corlib

        MetadataReference.CreateFromFile(typeof(RazorCompiledItemAttribute).Assembly.Location),
        // include Microsoft.AspNetCore.Razor.Runtime

        MetadataReference.CreateFromFile(Assembly.GetExecutingAssembly().Location), // this file
        (that contains the MyTemplate base class)

                        // for some reason on .NET core, I need to add this... this is not
        needed with .NET framework

        MetadataReference.CreateFromFile(Path.Combine(Path.GetDirectoryName(typeof(object).Assembl
         "System.Runtime.dll")),

                        // as found out by @Isantipov, for some other reason on .NET Core
        for Mac and Linux, we need to add this... this is not needed with .NET framework

        MetadataReference.CreateFromFile(Path.Combine(Path.GetDirectoryName(typeof(object).Assembl
         "netstandard.dll"))
                    },
                    new CSharpCompilationOptions(OutputKind.DynamicallyLinkedLibrary)); //
        we want a dll


                // compile the dll
                string path = Path.Combine(Path.GetFullPath("."), dllName + ".dll");
                var result = compilation.Emit(path);
                if (!result.Success)
                {
                    Console.WriteLine(string.Join(Environment.NewLine, result.Diagnostics));
                    return;
                }

                // load the built dll
                Console.WriteLine(path);
                var asm = Assembly.LoadFile(path);

                // the generated type is defined in our custom namespace, as we asked.
        "Template" is the type name that razor uses by default.
                var template =
        (MyTemplate)Activator.CreateInstance(asm.GetType("MyNamespace.Template"));

                // run the code.
                // should display "Hello Killroy, welcome to Razor World!"
                template.ExecuteAsync().Wait();
            }
```

```
    public class MyModel
    {
        // this will map to @Model.Name
        public string Name => "Killroy";
    }

    // the sample base template class. It's not mandatory but I think it's much easier.
    public abstract class MyTemplate
    {
        // this will map to @Model (property name)
        public MyModel Model => new MyModel();

        public void WriteLiteral(string literal)
        {
            // replace that by a text writer for example
            Console.Write(literal);
        }

        public void Write(object obj)
        {
            // replace that by a text writer for example
            Console.Write(obj);
        }

        public async virtual Task ExecuteAsync()
        {
            await Task.Yield(); // whatever, we just need something that compiles...
        }
    }
}
```

edited Aug 23 '18 at 6:01

answered Dec 11 '17 at 15:45

Simon Mourier

105k 13 191 242

2 Nice work, thanks! In order to get it working in netstandard 2.0 class library running in netcore2 app on mac and linux I had to add an additional reference to netstandard dll:
`MetadataReference.CreateFromFile(Path.Combine(Path.GetDirectoryName(typeof(object).Assembly.Location),"netstandard.dll")),` – lsantipov Dec 19 '17 at 11:03

2 @lsantipov - ok, thanks for pointing that out, I had not tested this on other platforms than Windows. I've updated the answer. – Simon Mourier Dec 19 '17 at 17:37

null? – James Wilkins Feb 26 '18 at 17:43

2 Well if I recall, the actual RazorViewEngine is in MVC, so I guess that makes Razor nothing more than a parser and compiler I guess. ;) – James Wilkins Feb 27 '18 at 7:27

1 @Dave - I have updated my answer. It should work with the newest versions now. – Simon Mourier Jul 20 '18 at 12:39

16

There's a working example for .NET Core 1.0 at aspnet/Entropy/samples/Mvc.RenderViewToString. Since this might change or go away, I'll detail the approach I'm using in my own applications here.

Tl;dr - Razor works really well outside of MVC! This approach can handle more complex rendering scenarios like partial views and injecting objects into views as well, although I'll just demonstrate a simple example below.

The core service looks like this:

RazorViewToStringRenderer.cs

```
using System;
using System.IO;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Abstractions;
using Microsoft.AspNetCore.Mvc.ModelBinding;
using Microsoft.AspNetCore.Mvc.Razor;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.AspNetCore.Mvc.ViewFeatures;
using Microsoft.AspNetCore.Routing;

namespace RenderRazorToString
{
    public class RazorViewToStringRenderer
    {
        private readonly IRazorViewEngine _viewEngine;
        private readonly ITempDataProvider _tempDataProvider;
        private readonly IServiceProvider _serviceProvider;

        public RazorViewToStringRenderer(
            IRazorViewEngine viewEngine,
            ITempDataProvider tempDataProvider,
```

```
            _tempDataProvider = tempDataProvider;
            _serviceProvider = serviceProvider;
        }

        public async Task<string> RenderViewToString<TModel>(string name, TModel model)
        {
            var actionContext = GetActionContext();

            var viewEngineResult = _viewEngine.FindView(actionContext, name, false);

            if (!viewEngineResult.Success)
            {
                throw new InvalidOperationException(string.Format("Couldn't find view
'{0}'", name));
            }

            var view = viewEngineResult.View;

            using (var output = new StringWriter())
            {
                var viewContext = new ViewContext(
                    actionContext,
                    view,
                    new ViewDataDictionary<TModel>(
                        metadataProvider: new EmptyModelMetadataProvider(),
                        modelState: new ModelStateDictionary())
                    {
                        Model = model
                    },
                    new TempDataDictionary(
                        actionContext.HttpContext,
                        _tempDataProvider),
                    output,
                    new HtmlHelperOptions());

                await view.RenderAsync(viewContext);

                return output.ToString();
            }
        }

        private ActionContext GetActionContext()
        {
            var httpContext = new DefaultHttpContext
            {
                RequestServices =  serviceProvider
```

```
ActionDescriptor());
        }
    }
}
```

A simple test console app just needs to initialize the service (and some supporting services), and call it:

Program.cs

```
using System;
using System.Diagnostics;
using System.IO;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Hosting.Internal;
using Microsoft.AspNetCore.Mvc.Razor;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.FileProviders;
using Microsoft.Extensions.ObjectPool;
using Microsoft.Extensions.PlatformAbstractions;

namespace RenderRazorToString
{
    public class Program
    {
        public static void Main()
        {
            // Initialize the necessary services
            var services = new ServiceCollection();
            ConfigureDefaultServices(services);
            var provider = services.BuildServiceProvider();

            var renderer = provider.GetRequiredService<RazorViewToStringRenderer>();

            // Build a model and render a view
            var model = new EmailViewModel
            {
                UserName = "User",
                SenderName = "Sender"
            };
            var emailContent = renderer.RenderViewToString("EmailTemplate",
model).GetAwaiter().GetResult();

            Console.WriteLine(emailContent);
            Console.ReadLine();
```

```
        {
            var applicationEnvironment = PlatformServices.Default.Application;
            services.AddSingleton(applicationEnvironment);

            var appDirectory = Directory.GetCurrentDirectory();

            var environment = new HostingEnvironment
            {
                WebRootFileProvider = new PhysicalFileProvider(appDirectory),
                ApplicationName = "RenderRazorToString"
            };
            services.AddSingleton<IHostingEnvironment>(environment);

            services.Configure<RazorViewEngineOptions>(options =>
            {
                options.FileProviders.Clear();
                options.FileProviders.Add(new PhysicalFileProvider(appDirectory));
            });

            services.AddSingleton<ObjectPoolProvider, DefaultObjectPoolProvider>();

            var diagnosticSource = new DiagnosticListener("Microsoft.AspNetCore");
            services.AddSingleton<DiagnosticSource>(diagnosticSource);

            services.AddLogging();
            services.AddMvc();
            services.AddSingleton<RazorViewToStringRenderer>();
        }
    }
}
```

This assumes that you have a view model class:

EmailViewModel.cs

```
namespace RenderRazorToString
{
    public class EmailViewModel
    {
        public string UserName { get; set; }

        public string SenderName { get; set; }
    }
}
```

Views/_Layout.cshtml

```
<!DOCTYPE html>

<html>
<body>
    <div>
        @RenderBody()
    </div>
    <footer>
Thanks,<br />
@Model.SenderName
    </footer>
</body>
</html>
```

Views/EmailTemplate.cshtml

```
@model RenderRazorToString.EmailViewModel
@{
    Layout = "_EmailLayout";
}

Hello @Model.UserName,

<p>
    This is a generic email about something.<br />
    <br />
</p>
```

edited Apr 27 '18 at 10:11

razon
2,133 1 18 28

answered Jul 7 '16 at 19:03

Nate Barbettini
32k 17 94 124

2 @dustinmoris If I recall correctly, it does do some caching for you. I haven't tried it in a while. – Nate Barbettini Mar 26 '17 at 17:17

2 Awesome! But, this doesn't work in .net core 2.0 :(It seems that the dependencies can't be loaded: `The type 'Attribute' is defined in an assembly that is not referenced. You must add a reference to assembly 'netstandard, Version=2.0.0.0, Culture=neutral, PublicKeyToken=cc7b13ffcd2ddd51'` - I'm not sure how to tell razor to load all the dependencies it needs - any ideas? – Matt Roberts Sep 6 '17 at 11:09

1 @MehdiDehghani Razor always takes a long time to compile on the fly (the first time). You might be interested in the new Razor library support and compile-at-build feature in ASP.NET Core 2.1: blogs.msdn.microsoft.com/webdev/2018/02/02/… – Nate Barbettini Mar 24 '18 at 22:47

2 For anyone getting the `The type 'Attribute' is defined in an assembly that is not referenced` error, adding `<PreserveCompilationContext>true</PreserveCompilationContext>` resolved the issue. – Mark G May 9 '18 at 4:00

6

Here is a class to get Nate's answer working as a scoped service in an ASP.NET Core 2.0 project.

```
using System;
using System.IO;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Abstractions;
using Microsoft.AspNetCore.Mvc.ModelBinding;
using Microsoft.AspNetCore.Mvc.Razor;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.AspNetCore.Mvc.ViewFeatures;
using Microsoft.AspNetCore.Routing;

namespace YourNamespace.Services
{
    public class ViewRender : IViewRender
    {
        private readonly IRazorViewEngine _viewEngine;
        private readonly ITempDataProvider _tempDataProvider;
        private readonly IServiceProvider _serviceProvider;

        public ViewRender(
            IRazorViewEngine viewEngine,
            ITempDataProvider tempDataProvider,
            IServiceProvider serviceProvider)
        {
            _viewEngine = viewEngine;
            _tempDataProvider = tempDataProvider;
            _serviceProvider = serviceProvider;
        }

        public async Task<string> RenderAsync(string name)
        {
            return await RenderAsync<object>(name, null);
        }
```

```
            var actionContext = GetActionContext();

            var viewEngineResult = _viewEngine.FindView(actionContext, name, false);

            if (!viewEngineResult.Success)
            {
                throw new InvalidOperationException(string.Format("Couldn't find view
'{0}'", name));
            }

            var view = viewEngineResult.View;

            using (var output = new StringWriter())
            {
                var viewContext = new ViewContext(
                    actionContext,
                    view,
                    new ViewDataDictionary<TModel>(
                        metadataProvider: new EmptyModelMetadataProvider(),
                        modelState: new ModelStateDictionary())
                    {
                        Model = model
                    },
                    new TempDataDictionary(
                        actionContext.HttpContext,
                        _tempDataProvider),
                    output,
                    new HtmlHelperOptions());

                await view.RenderAsync(viewContext);

                return output.ToString();
            }
        }

        private ActionContext GetActionContext()
        {
            var httpContext = new DefaultHttpContext {RequestServices =
_serviceProvider};
            return new ActionContext(httpContext, new RouteData(), new
ActionDescriptor());
        }
    }

    public interface IViewRender
    {
```

```
    }
}
```

In Startup.cs

```
public void ConfigureServices(IServiceCollection services)
{
     services.AddScoped<IViewRender, ViewRender>();
}
```

In a controller

```
public class VenuesController : Controller
{
    private readonly IViewRender _viewRender;

    public VenuesController(IViewRender viewRender)
    {
        _viewRender = viewRender;
    }

    public async Task<IActionResult> Edit()
    {
        string html = await _viewRender.RenderAsync("Emails/VenuePublished",
venue.Name);
        return Ok();
    }
}
```

answered Oct 7 '17 at 0:09

ArcadeRenegade
508 6 11

I am not able to get this to work. I get the error: Unable to resolve service for type 'Microsoft.AspNetCore.Mvc.Razor.IRazorViewEngine' while attempting to activate 'Mvc.RenderViewToString.RazorViewToStringRenderer'.' – Kjensen May 11 '18 at 21:01

1 This is a very nice and simple answer and mostly great because it works in Linux backed Docker images. A lot of other solutions do not work due to some Linux specific issues.... This one does. Thank you! This should be the answer for ASPNET CORE 2+ – Piotr Kula Dec 19 '18 at 14:33

Does this use caching at all? – ijxtra Feb 24 at 16:04

0

I spent several days fiddling with razor light, but it has a number of deficiencies such as not having html helpers (@Html.*) or url helpers, and other quirks.

Here is a solution that is encapsulated for usage outside of an mvc app. It does require package references to aspnet core and mvc, but those are easy to add to a service or console application. No controllers or web server are needed. RenderToStringAsync is the method to call to render a view to a string.

The advantage is that you can write your views the same way you would in a .net core web project. You can use the same @Html and other helper functions and methods.

You can replace or add to the physical file provider in the razor view options setup with your own custom provider to load views from database, web service call, etc. Tested with .net core 2.2 on Windows and Linux.

Please note that your .csproj file must have this as the top line:

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


using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Dynamic;
using System.IO;
using System.Linq;
using System.Threading.Tasks;

using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Hosting.Internal;
using Microsoft.AspNetCore.Mvc.Razor;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.FileProviders;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.ObjectPool;

namespace RazorRendererNamespace
{
    /// <summary>
    /// Renders razor pages with the absolute minimum setup of MVC, easy to use in
console application, does not require any other classes or setup.
    /// </summary>
    public class RazorRenderer : ILoggerFactory, ILogger
    {
```

```
        {
            private static readonly System.Net.IPAddress localIPAddress =
System.Net.IPAddress.Parse("127.0.0.1");

            private readonly Dictionary<string, object> tempData = new
Dictionary<string, object>(StringComparer.OrdinalIgnoreCase);
            private readonly IRazorViewEngine _viewEngine;
            private readonly ITempDataProvider _tempDataProvider;
            private readonly IServiceProvider _serviceProvider;
            private readonly IHttpContextAccessor _httpContextAccessor;

            public ViewRenderService(IRazorViewEngine viewEngine,
                IHttpContextAccessor httpContextAccessor,
                ITempDataProvider tempDataProvider,
                IServiceProvider serviceProvider)
            {
                _viewEngine = viewEngine;
                _httpContextAccessor = httpContextAccessor;
                _tempDataProvider = tempDataProvider ?? this;
                _serviceProvider = serviceProvider ?? this;
            }

            public void Dispose()
            {

            }

            public async Task<string> RenderToStringAsync<TModel>(string viewName,
TModel model, ExpandoObject viewBag = null, bool isMainPage = false)
            {
                HttpContext httpContext;
                if (_httpContextAccessor?.HttpContext != null)
                {
                    httpContext = _httpContextAccessor.HttpContext;
                }
                else
                {
                    DefaultHttpContext defaultContext = new DefaultHttpContext {
RequestServices = _serviceProvider };
                    defaultContext.Connection.RemoteIpAddress = localIPAddress;
                    httpContext = defaultContext;
                }
                var actionContext = new ActionContext(httpContext, new RouteData(), new
ActionDescriptor());
                using (var sw = new StringWriter())
                {
```

```
                    if (viewResult.View == null)
                    {
                        viewResult = _viewEngine.GetView("~/", viewName, isMainPage);
                    }

                    if (viewResult.View == null)
                    {
                        return null;
                    }

                    var viewDictionary = new ViewDataDictionary(new
EmptyModelMetadataProvider(), new ModelStateDictionary())
                    {
                        Model = model
                    };
                    if (viewBag != null)
                    {
                        foreach (KeyValuePair<string, object> kv in (viewBag as
IDictionary<string, object>))
                        {
                            viewDictionary.Add(kv.Key, kv.Value);
                        }
                    }
                    var viewContext = new ViewContext(
                        actionContext,
                        viewResult.View,
                        viewDictionary,
                        new TempDataDictionary(actionContext.HttpContext,
_tempDataProvider),
                        sw,
                        new HtmlHelperOptions()
                    );

                    await viewResult.View.RenderAsync(viewContext);
                    return sw.ToString();
                }
            }

            object IServiceProvider.GetService(Type serviceType)
            {
                return null;
            }

            IDictionary<string, object> ITempDataProvider.LoadTempData(HttpContext
context)
            {
```

```
            void ITempDataProvider.SaveTempData(HttpContext context, IDictionary<string,
object> values)
            {
            }
        }

        private readonly string rootPath;
        private readonly ServiceCollection services;
        private readonly ServiceProvider serviceProvider;
        private readonly ViewRenderService viewRenderer;

        public RazorRenderer(string rootPath)
        {
            this.rootPath = rootPath;
            services = new ServiceCollection();
            ConfigureDefaultServices(services);
            serviceProvider = services.BuildServiceProvider();
            viewRenderer = new
ViewRenderService(serviceProvider.GetRequiredService<IRazorViewEngine>(), null, null,
serviceProvider);
        }

        private void ConfigureDefaultServices(IServiceCollection services)
        {
            var environment = new HostingEnvironment
            {
                WebRootFileProvider = new PhysicalFileProvider(rootPath),
                ApplicationName = typeof(RazorRenderer).Assembly.GetName().Name,
                ContentRootPath = rootPath,
                WebRootPath = rootPath,
                EnvironmentName = "DEVELOPMENT",
                ContentRootFileProvider = new PhysicalFileProvider(rootPath)
            };
            services.AddSingleton<IHostingEnvironment>(environment);
            services.Configure<RazorViewEngineOptions>(options =>
            {
                options.FileProviders.Clear();
                options.FileProviders.Add(new PhysicalFileProvider(rootPath));
            });
            services.AddSingleton<ObjectPoolProvider, DefaultObjectPoolProvider>();
            services.AddSingleton<ILoggerFactory>(this);
            var diagnosticSource = new DiagnosticListener(environment.ApplicationName);
            services.AddSingleton<DiagnosticSource>(diagnosticSource);
            services.AddMvc();
        }
```

```
        public Task<string> RenderToStringAsync<TModel>(string viewName, TModel model,
ExpandoObject viewBag = null, bool isMainPage = false)
        {
            return viewRenderer.RenderToStringAsync(viewName, model, viewBag,
isMainPage);
        }

        void ILoggerFactory.AddProvider(ILoggerProvider provider)
        {

        }

        IDisposable ILogger.BeginScope<TState>(TState state)
        {
            throw new NotImplementedException();
        }

        ILogger ILoggerFactory.CreateLogger(string categoryName)
        {
            return this;
        }

        bool ILogger.IsEnabled(Microsoft.Extensions.Logging.LogLevel logLevel)
        {
            return false;
        }

        void ILogger.Log<TState>(Microsoft.Extensions.Logging.LogLevel logLevel, EventId
eventId, TState state, Exception exception, Func<TState, Exception, string> formatter)
        {
        }
    }
}
```

edited Mar 2 at 20:53

answered Mar 2 at 15:36

jjxtra

12.5k 12 69 115