Rick Strahl's Web Log
Wind, waves, code and everything in between...
ASP.NET • C# • HTML5 • JavaScript • AngularJs

Articles

Rendering ASP.NET MVC Views to String

May 30, 2012 - from Hood River, Oregon

Tweet 17 comments

sponsored by

It's not uncommon in my applications that I require longish text output that does not have to be rendered into the HTTP output stream. The most common scenario I have for 'template driven' non-Web text is for emails of all sorts. Logon confirmations and verifications, email confirmations for things like orders, status updates or scheduler notifications - all of which require merged text output both within and sometimes outside of Web applications. On other occasions I also need to capture the output from certain views for logging purposes.

Rather than creating text output in code, it's much nicer to use the rendering mechanism that ASP.NET MVC already provides by way of it's ViewEngines - using Razor or WebForms views - to render output to a string. This is nice because it uses the same familiar rendering mechanism that I already use for my HTTP output and it also solves the problem of where to store the templates for rendering this content in nothing more than perhaps a separate view folder.

The good news is that ASP.NET MVC's rendering engine is much more modular than the full ASP.NET runtime engine which was a real pain in the butt to coerce into rendering output to string. With MVC the rendering engine has been separated out from core ASP.NET runtime, so it's actually a lot easier to get View output into a string.

Getting View Output from within an MVC Application

If you need to generate string output from an MVC and pass some model data to it, the process to capture this output is fairly straight forward and involves only a handful of lines of code. The catch is that this particular approach requires that you have an active ControllerContext that can be passed to the view. This means that the following approach is limited to access from within Controller methods.

Here's a class that wraps the process and provides both instance and static methods to handle the rendering:

```
/// <summary>
/// Class that renders MVC views to a string using the
/// standard MVC View Engine to render the view.
/// </summary>
public class ViewRenderer
{
    /// <summary>
    /// Required Controller Context
    /// </summary>
    protected ControllerContext Context { get; set; }

    /// <summary>
    /// Initializes the ViewRenderer with a Context.
    /// </summary>
    /// <param name="controllerContext">
    /// If you are running within the context of an ASP.NET MVC request pass in
    /// the controller's context.
    /// Only leave out the context if no context is otherwise available.
    /// </param>
    public ViewRenderer(ControllerContext controllerContext = null)
    {
        // Create a known controller from HttpContext if no context is passed
        if (controllerContext == null)
        {
            if (HttpContext.Current != null)
                controllerContext = CreateController<ErrorController>().ControllerContext;
            else
                throw new InvalidOperationException(
                    "ViewRenderer must run in the context of an ASP.NET " +
                    "Application and requires HttpContext.Current to be present.");
        }
        Context = controllerContext;
    }

    /// <summary>
    /// Renders a full MVC view to a string. Will render with the full MVC
    /// View engine including running _ViewStart and merging into _Layout
    /// </summary>
    /// <param name="viewPath">
    /// The path to the view to render. Either in same controller, shared by
    /// name or as fully qualified ~/ path including extension
    /// </param>
    /// <param name="model">The model to render the view with</param>
    /// <returns>String of the rendered view or null on error</returns>
    public string RenderView(string viewPath, object model)
    {
        return RenderViewToStringInternal(viewPath, model, false);
    }


    /// <summary>
    /// Renders a partial MVC view to string. Use this method to render
    /// a partial view that doesn't merge with _Layout and doesn't fire
    /// _ViewStart.
    /// </summary>
    /// <param name="viewPath">
    /// The path to the view to render. Either in same controller, shared by
    /// name or as fully qualified ~/ path including extension
    /// </param>
    /// <param name="model">The model to pass to the viewRenderer</param>
    /// <returns>String of the rendered view or null on error</returns>
    public string RenderPartialView(string viewPath, object model)
    {
        return RenderViewToStringInternal(viewPath, model, true);
    }

    /// <summary>
    /// Renders a partial MVC view to string. Use this method to render
    /// a partial view that doesn't merge with _Layout and doesn't fire
    /// _ViewStart.
    /// </summary>
    /// <param name="viewPath">
    /// The path to the view to render. Either in same controller, shared by
    /// name or as fully qualified ~/ path including extension
    /// </param>
    /// <param name="model">The model to pass to the viewRenderer</param>
    /// <param name="controllerContext">Active Controller context</param>
    /// <returns>String of the rendered view or null on error</returns>
    public static string RenderView(string viewPath, object model,
                                    ControllerContext controllerContext)
    {
        ViewRenderer renderer = new ViewRenderer(controllerContext);
        return renderer.RenderView(viewPath, model);
    }

    /// <summary>
    /// Renders a partial MVC view to string. Use this method to render
    /// a partial view that doesn't merge with _Layout and doesn't fire
    /// _ViewStart.
    /// </summary>
    /// <param name="viewPath">
    /// The path to the view to render. Either in same controller, shared by
    /// name or as fully qualified ~/ path including extension
    /// </param>
    /// <param name="model">The model to pass to the viewRenderer</param>
    /// <param name="controllerContext">Active Controller context</param>
    /// <param name="errorMessage">optional out parameter that captures an error message i
    /// <returns>String of the rendered view or null on error</returns>
    public static string RenderView(string viewPath, object model,
                                    ControllerContext controllerContext,
                                    out string errorMessage)
    {
        errorMessage = null;
        try
        {
            ViewRenderer renderer = new ViewRenderer(controllerContext);
            return renderer.RenderView(viewPath, model);
        }
        catch (Exception ex)
```

```
        {
            errorMessage = ex.GetBaseException().Message;
        }
        return null;
    }

    /// <summary>
    /// Renders a partial MVC view to string. Use this method to render
    /// a partial view that doesn't merge with _Layout and doesn't fire
    /// _ViewStart.
    /// </summary>
    /// <param name="viewPath">
    /// The path to the view to render. Either in same controller, shared by
    /// name or as fully qualified ~/ path including extension
    /// </param>
    /// <param name="model">The model to pass to the viewRenderer</param>
    /// <param name="controllerContext">Active controller context</param>
    /// <returns>String of the rendered view or null on error</returns>
    public static string RenderPartialView(string viewPath, object model,
                                           ControllerContext controllerContext)
    {
        ViewRenderer renderer = new ViewRenderer(controllerContext);
        return renderer.RenderPartialView(viewPath, model);
    }

    /// <summary>
    /// Renders a partial MVC view to string. Use this method to render
    /// a partial view that doesn't merge with _Layout and doesn't fire
    /// _ViewStart.
    /// </summary>
    /// <param name="viewPath">
    /// The path to the view to render. Either in same controller, shared by
    /// name or as fully qualified ~/ path including extension
    /// </param>
    /// <param name="model">The model to pass to the viewRenderer</param>
    /// <param name="controllerContext">Active controller context</param>
    /// <param name="errorMessage">optional output parameter to receive an error message o
    /// <returns>String of the rendered view or null on error</returns>
    public static string RenderPartialView(string viewPath, object model,
                                           ControllerContext controllerContext,
                                           out string errorMessage)
    {
        errorMessage = null;
        try
        {
            ViewRenderer renderer = new ViewRenderer(controllerContext);
            return renderer.RenderPartialView(viewPath, model);
        }
        catch (Exception ex)
        {
            errorMessage = ex.GetBaseException().Message;
        }
        return null;
    }
```

```
/// <summary>
/// Internal method that handles rendering of either partial or
/// or full views.
/// </summary>
/// <param name="viewPath">
/// The path to the view to render. Either in same controller, shared by
/// name or as fully qualified ~/ path including extension
/// </param>
/// <param name="model">Model to render the view with</param>
/// <param name="partial">Determines whether to render a full or partial view</param>
/// <returns>String of the rendered view</returns>
protected string RenderViewToStringInternal(string viewPath, object model,
                                            bool partial = false)
{
    // first find the ViewEngine for this view
    ViewEngineResult viewEngineResult = null;
    if (partial)
        viewEngineResult = ViewEngines.Engines.FindPartialView(Context, viewPath);
    else
        viewEngineResult = ViewEngines.Engines.FindView(Context, viewPath, null);

    if (viewEngineResult == null)
        throw new FileNotFoundException(Resources.ViewCouldNotBeFound);

    // get the view and attach the model to view data
    var view = viewEngineResult.View;
    Context.Controller.ViewData.Model = model;

    string result = null;

    using (var sw = new StringWriter())
    {
        var ctx = new ViewContext(Context, view,
                                    Context.Controller.ViewData,
                                    Context.Controller.TempData,
                                    sw);
        view.Render(ctx, sw);
        result = sw.ToString();
    }

    return result;
}


/// <summary>
/// Creates an instance of an MVC controller from scratch
/// when no existing ControllerContext is present
/// </summary>
/// <typeparam name="T">Type of the controller to create</typeparam>
/// <returns></returns>
public static T CreateController<T>(RouteData routeData = null)
            where T : Controller, new()
{
    T controller = new T();
```

```
        // Create an MVC Controller Context
        HttpContextBase wrapper = null;
        if (HttpContext.Current != null)
            wrapper = new HttpContextWrapper(System.Web.HttpContext.Current);
        //else
        //    wrapper = CreateHttpContextBase(writer);


        if (routeData == null)
            routeData = new RouteData();

        if (!routeData.Values.ContainsKey("controller") && !routeData.Values.ContainsKey('
            routeData.Values.Add("controller", controller.GetType().Name
                                                    .ToLower()
                                                    .Replace("controller", ""));

        controller.ControllerContext = new ControllerContext(wrapper, routeData, controlle
        return controller;
    }
}
```

The key is the RenderViewToStringInternal method. The method first tries to find the view to render based on its path which can either be in the current controller's view path or the shared view path using its simple name (PasswordRecovery) or alternately by its full virtual path (~/Views/Templates/PasswordRecovery.cshtml). This code should work both for Razor and WebForms views although I've only tried it with Razor Views. Note that WebForms Views might actually be better for plain text as Razor adds all sorts of white space into its output when there are code blocks in the template. The Web Forms engine provides more accurate rendering for raw text scenarios.

Once a view engine is found the view to render can be retrieved. Views in MVC render based on data that comes off the controller like the ViewData which contains the model along with the actual ViewData and ViewBag. From the View and some of the Context data a ViewContext is created which is then used to render the view with. The View picks up the Model and other data from the ViewContext internally and processes the View the same it would be processed if it were to send its output into the HTTP output stream. The difference is that we can override the ViewContext's output stream which we provide and capture into a StringWriter(). After rendering completes the result holds the output string.

If an error occurs the error behavior is similar what you see with regular MVC errors - you get a full yellow screen of death including the view error information with the line of error highlighted. It's your responsibility to handle the error - or let it bubble up to your regular Controller Error filter if you have one.

To use the simple class you only need a single line of code if you call the static methods. Here's an example of some Controller code that is used to send a user notification to a customer via email in one of my applications:

```
[HttpPost]
public ActionResult ContactSeller(ContactSellerViewModel model)
{
    InitializeViewModel(model);

    var entryBus = new busEntry();
    var entry = entryBus.LoadByDisplayId(model.EntryId);
         if ( string.IsNullOrEmpty(model.Email) )
        entryBus.ValidationErrors.Add("Email address can't be empty.","Email");
    if ( string.IsNullOrEmpty(model.Message))
        entryBus.ValidationErrors.Add("Message can't be empty.","Message");

    model.EntryId = entry.DisplayId;
    model.EntryTitle = entry.Title;

    if (entryBus.ValidationErrors.Count > 0)
    {
        ErrorDisplay.AddMessages(entryBus.ValidationErrors);
        ErrorDisplay.ShowError("Please correct the following:");
    }
    else
    {
        string message = ViewRenderer.RenderView("~/views/template/ContactSellerEmail.csht
                                         ControllerContext);

        string title = entry.Title + " (" + entry.DisplayId + ") - " + App.Configuration.A
        AppUtils.SendEmail(title, message, model.Email,
                           entry.User.Email, false, false))
    }

    return View(model);
}
```

Simple!

The view in this case is just a plain MVC view and in this case it's a very simple plain text email message (edited for brevity here) that is created and sent off:

```
@model ContactSellerViewModel
@{
    Layout = null;
}re: @Model.EntryTitle
@Model.ListingUrl

@Model.Message



** SECURITY ADVISORY - AVOID SCAMS
** Avoid: wiring money, cross-border deals, work-at-home
** Beware: cashier checks, money orders, escrow, shipping
** More Info: @(App.Configuration.ApplicationBaseUrl)scams.html
```

Obviously this is a very simple view (I edited out more from this page to keep it brief) - but other template views are much more complex HTML documents or long messages that are occasionally updated and they are a perfect fit for Razor rendering. It even works with nested partial views and _layout pages.

Partial Rendering

Notice that I'm rendering a full View here. In the view I explicitly set the Layout=null to avoid pulling in _layout.cshtml for this view. This can also be controlled externally by calling the RenderPartial method instead:

string message = ViewRenderer.RenderPartialView("~/views/template/ContactSellerEmail.cshtml",model, ControllerContext);

with this line of code no layout page (or _viewstart) will be loaded, so the output generated is just what's in the view. I find myself using Partials most of the time when rendering templates, since the target of templates usually tend to be emails or other HTML fragment like output, so the RenderPartialView() method is definitely useful to me.

Rendering without a ControllerContext

The preceding class is great when you're need template rendering from within MVC controller actions or anywhere where you have access to the request Controller. But if you don't have a controller context handy - maybe inside a utility function that is static, a non-Web application, or an operation that runs asynchronously in ASP.NET - which makes using the above code impossible. I haven't found a way to manually create a Controller context to provide the ViewContext() what it needs from outside of the MVC infrastructure.

However, there are ways to accomplish this, but they are a bit more complex. It's possible to host the RazorEngine on your own, which side steps all of the MVC framework and HTTP and just deals with the raw rendering engine. I wrote about this process in **Hosting the Razor Engine in Non-Web Applications** a long while back. It's quite a process to create a custom Razor engine and runtime, but it allows for all sorts of flexibility. There's also a **RazorEngine CodePlex project** that does something similar. I've been meaning to check out the latter but haven't gotten around to it since I have my o code to do this. The trick to hosting the RazorEngine to have it behave properly inside of an ASP.NE application and properly cache content so templates aren't constantly rebuild and reparsed.

Anyway, in the same app as above I have one scenario where no ControllerContext is available: I have a background scheduler running inside of the app that fires on timed intervals. This process could be external but because it's lightweight we decided to fire it right inside of the ASP.NET app on a separate thread.

In my app the code that renders these templates does something like this:

```
var model = new SearchNotificationViewModel()
{
    Entries = entries,
    Notification = notification,
    User = user
};


// TODO: Need logging for errors sending
string razorError = null;
```

```
var result = AppUtils.RenderRazorTemplate("~/views/template/SearchNotificationTemplate.csh
                                          razorError);
```

which references a couple of helper functions that set up my RazorFolderHostContainer class:

```
public static string RenderRazorTemplate(string virtualPath, object model,string errorMess
{
    var razor = AppUtils.CreateRazorHost();

    var path = virtualPath.Replace("~/", "").Replace("~", "").Replace("/", "\\");
    var merged = razor.RenderTemplateToString(path, model);
    if (merged == null)
        errorMessage = razor.ErrorMessage;

    return merged;
}

/// <summary>
/// Creates a RazorStringHostContainer and starts it
/// Call .Stop() when you're done with it.
///
/// This is a static instance
/// </summary>
/// <param name="virtualPath"></param>
/// <param name="binBasePath"></param>
/// <param name="forceLoad"></param>
/// <returns></returns>
public static RazorFolderHostContainer CreateRazorHost(string binBasePath = null,
                                                       bool forceLoad = false)
{
    if (binBasePath == null)
    {
        if (HttpContext.Current != null)
            binBasePath = HttpContext.Current.Server.MapPath("~/");
        else
            binBasePath = AppDomain.CurrentDomain.BaseDirectory;
    }

    if (_RazorHost == null || forceLoad)
    {
        if (!binBasePath.EndsWith("\\"))
            binBasePath += "\\";

        //var razor = new RazorStringHostContainer();
        var razor = new RazorFolderHostContainer();
        razor.TemplatePath = binBasePath;
        binBasePath += "bin\\";
        razor.BaseBinaryFolder = binBasePath;
        razor.UseAppDomain = false;
        razor.ReferencedAssemblies.Add(binBasePath + "ClassifiedsBusiness.dll");
        razor.ReferencedAssemblies.Add(binBasePath + "ClassifiedsWeb.dll");
        razor.ReferencedAssemblies.Add(binBasePath + "Westwind.Utilities.dll");
        razor.ReferencedAssemblies.Add(binBasePath + "Westwind.Web.dll");
```

```
            razor.ReferencedAssemblies.Add(binBasePath + "Westwind.Web.Mvc.dll");
            razor.ReferencedAssemblies.Add("System.Web.dll");
            razor.ReferencedNamespaces.Add("System.Web");
            razor.ReferencedNamespaces.Add("ClassifiedsBusiness");
            razor.ReferencedNamespaces.Add("ClassifiedsWeb");
            razor.ReferencedNamespaces.Add("Westwind.Web");
            razor.ReferencedNamespaces.Add("Westwind.Utilities");

            _RazorHost = razor;
            _RazorHost.Start();

            //_RazorHost.Engine.Configuration.CompileToMemory = false;
        }

        return _RazorHost;
    }
```

The RazorFolderHostContainer essentially is a full runtime that mimics a folder structure like a typical Web app does including caching semantics and compiling code only if code changes on disk. It maps a folder hierarchy to views using the ~/ path syntax. The host is then configured to add assemblies and namespaces. Unfortunately the engine is not exactly like MVC's Razor - the expression expansion and code execution are the same, but some of the support methods like sections, helpers etc. are not all there so templates have to be a bit simpler. There are other folder hosts provided as well to directly execute templates from strings (using RazorStringHostContainer).

The following is an example of an HTML email template

```
@inherits RazorHosting.RazorTemplateFolderHost <ClassifiedsWeb.SearchNotificati
<html>
    <head>
        <title>Search Notifications</title>
        <style>
            body { margin: 5px;font-family: Verdana, Arial; font-size: 10pt;}
            h3 { color: SteelBlue; }
            .entry-item { border-bottom: 1px solid grey; padding: 8px; margin-t
        </style>
    </head>
    <body>

        Hello @Model.User.Name,<br />
        <p>Below  are your Search Results for the search phrase:</p>
        <h3>@Model.Notification.SearchPhrase</h3>
        <small>since @TimeUtils.ShortDateString(Model.Notification.LastSearch)<
        <hr />
```

You can see that the syntax is a little different. Instead of the familiar @model header the raw Razor

@inherits tag is used to specify the template base class (which you can extend). I took a quick look through the feature set of RazorEngine on CodePlex (now Github I guess) and the template implementation they use is closer to MVC's razor but there are other differences. In the end don't expect exact behavior like MVC templates if you use an external Razor rendering engine.

This is not what I would consider an ideal solution, but it works well enough for this project. My biggest concern is the overhead of hosting a second razor engine in a Web app and the fact that here the differences in template rendering between 'real' MVC Razor views and another RazorEngine really are noticeable.

You win some, you lose some

It's extremely nice to see that if you have a ControllerContext handy (which probably addresses 99% of Web app scenarios) rendering a view to string using the native MVC Razor engine is pretty simple. Kudos on making that happen - as it solves a problem I see in just about every Web application I work on.

But it is a bummer that a ControllerContext is required to make this simple code work. It'd be really sweet if there was a way to render views without being so closely coupled to the ASP.NET or MVC infrastructure that requires a ControllerContext. Alternately it'd be nice to have a way for an MVC based application to create a minimal ControllerContext from scratch - maybe somebody's been down that path. I tried for a few hours to come up with a way to make that work but gave up in the soup of nested contexts (MVC/Controller/View/Http). I suspect going down this path would be similar to hosting the ASP.NET runtime requiring a WorkerRequest. Brrr….

The sad part is that it seems to me that a View should really not require much 'context' of any kind to render output to string. Yes there are a few things that clearly are required like paths to the virtual and possibly the disk paths to the root of the app, but beyond that view rendering should not require much. But, no such luck. For now custom RazorHosting seems to be the only way to make Razor rendering go outside of the MVC context…

Resources

- **Full ViewRenderer.cs source code from Westwind.Web.Mvc library**
- **Hosting the Razor Engine for Non-Web Applications**
- **RazorEngine on GitHub**
- **Related discussion on StackOverFlow**

Other Posts you might also like

- **First Steps: Exploring .NET Core and ASP.NET Core**
- **Adding minimal OWIN Identity Authentication to an Existing ASP.NET MVC Application**
- **Strongly Typed Configuration Settings in ASP.NET Core**
- **Upgrading to ASP.NET Core RTM from RC2**

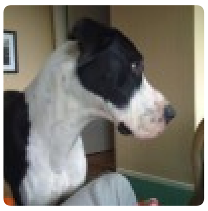
Donate Bitcoins

Donate with PayPal

Posted in **ASP.NET** **ASP.NET MVC**

Tweet

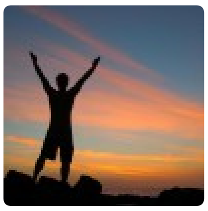
The Voices of Reason



RickB
June 05, 2012

# re: Rendering ASP.NET MVC Views to String

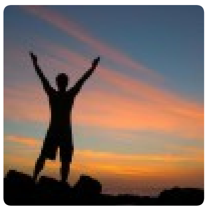
I am doing somehting similar to handle a simple CMS website wher eI loop over a list of controllers/views and call Renderpage on each one. My issue is that I want each of my partial views to be able to specify a scripts section that will be accumulated and rendered at the bottom of the page. Any thoughts on how to handle this?



Rick Strahl
June 05, 2012

# re: Rendering ASP.NET MVC Views to String

@Rickb - hmmm... why? You could have each of the views render their scripts inline and that will actually work fine (assuming there are no naming conflicts). Other than that I don't think you can do that because each view rendered is effectively independent of the other so there's no interaction. Once the view is rendered you can't track what the content of the other is. One way you can potentially handle this is have one 'master view' that renders all the others with a @section command which effectively gets embedded into the parent, but that requires that you know what the partials are upfront (ie. a static master view).



Rick Strahl

June 06, 2012

# re: Rendering ASP.NET MVC Views to String

@ms440 - haven't tried it, but it looks like it solves a whole other set of problems than typical razor engines which tend to be more dynamic. RazorGenerator seems to be more geared to packaging and including compiled templates into projects so the ,cshtml files can be left behind. That can be very useful, but serves a different purpose...

Alex

September 22, 2012

# re: Rendering ASP.NET MVC Views to String

Awesome. Thanks Rick.

I wrote a controller-method for this, but then realized I need to render a view outside of a controller context (for an email), your post saved my life.

PS. In case anyone needs the "controller-way" for this:

```
public class BaseController : Controller
{
    public string RenderRazorViewToString(string viewName, object model)
    {
        ViewData.Model = model;
        using (var sw = new System.IO.StringWriter())
        {
            var viewResult = ViewEngines.Engines.FindPartialView(ControllerContex
            var viewContext = new ViewContext(ControllerContext, viewResult.View,
            viewResult.View.Render(viewContext, sw);
            viewResult.ViewEngine.ReleaseView(ControllerContext, viewResult.View)
            return sw.GetStringBuilder().ToString();
        }
    }
}
```

Mau

November 09, 2012

# re: Rendering ASP.NET MVC Views to String

Thanks a lot Rick, this is way better than doing stringbuilders on code.

Chris Green
January 10, 2013

# re: Rendering ASP.NET MVC Views to String

Thanks Rick. Was battling with this for a while from other examples. The difference with yours is that the explanations as I worked through it enabled me to overcome various issues I experience.

Great stuff.

Joao Albuquerque
July 15, 2013

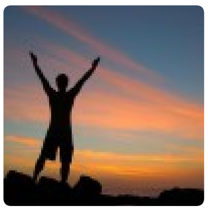
# re: Rendering ASP.NET MVC Views to String

Hi Rick! Any clue why am I having this result from a partial view (any view)?

<$A$><div>Teste! Texto da PartialView</div></$A$>

What are these tags <$A$> B, C, D etc...? It breaks my HTML layout.
I am using ASP.NET MVC 4.

Thanks.



Rick Strahl

July 15, 2013

# re: Rendering ASP.NET MVC Views to String

@Joao - What are you rendering from? Using ViewRenderer? The external Razor engine? and what does the razor code look like that's causing this?

Joao Albuquerque

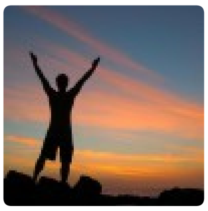
July 15, 2013

# re: Rendering ASP.NET MVC Views to String

Hi Rick! Thanks for your reply. Well, I was using my code and the result was the same. Then, I picked up your code here (w/ no modifications) for testing and got the very same result. I am using a clean MVC project with .NET FrameWork 4.5 (also used 4.5.1). RazorEngine is referenced in the project as v4.0.30319 (RazorEngine - Core Framework).

The result string returned from a simple View is this:

```
<$C$><!DOCTYPE html>
<html>
<head>
    <meta</$C$><$D$> charset="utf-8"</$D$><$E$> />
    <meta</$E$><$F$> name="viewport"</$F$><$G$> content="width=device-width, init
    <title>Teste</title>
</head>
<body>
</$H$><$I$>    </$I$><$J$><$A$>
</$A$><$B$><div>Test!</div></$B$></$J$><$K$>
</body>
</html>
</$K$>
```



Rick Strahl
July 15, 2013

# re: Rendering ASP.NET MVC Views to String

@Joao - I've never seen that, but it sure looks like a tokenizer string that's used to parse the razor code. Not sure why you'd ever end up with tokenizer code instead of final parsed output though. You might want to check with Andrew Nurse (@anurse on Twitter), he's the Razor Wizard at Microsoft and he might have some ideas.

Rich
August 11, 2014

# re: Rendering ASP.NET MVC Views to String

Just what the doctor ordered! Thanks for sharing!

Vicente Mantrana
January 14, 2015

# re: Rendering ASP.NET MVC Views to String

Hi:

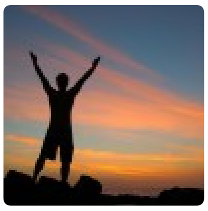
Is there any way to extend this functionality in a way that it takes a partial view name and a _layout name and dynamically apply that layout to the partial view and render the resulting view as a string?

justin
May 17, 2015

# re: Rendering ASP.NET MVC Views to String

Hi, thanks for the article.

Just wonder is there a way to keep the css style after rendering the view to string?
looks like all styles are gone.



Rick Strahl
May 17, 2015

# re: Rendering ASP.NET MVC Views to String

@Justin - the renderer just creates text. if you have relative links for CSS and images you need to fix up those links dependening on the new location from which the HTML is served. Either make the URLs of the original page absolute or convert them to absolute paths when you're rendering.

freedomn-m
May 19, 2016

# re: Rendering ASP.NET MVC Views to String

@Vicente Mantrana - the line 'ViewEngines.Engines.FindView' takes masterName as the last argument (null in the code here) - pass your _layout path instead of the null, eg: "~/Views/Shared/_LayoutEmail.cshtml"

Travis
July 16, 2016

# re: Rendering ASP.NET MVC Views to String

Hi Rick.

Thanks for this Class. Just a quick question:

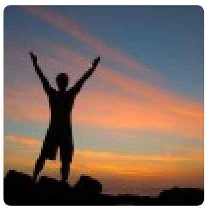
I have a 'Basic' view template (using Razor) for a basic email. It contains a bit of html jazz an then @Model.EmailMessage (to substitute my message). I then populate like so:

```
var EmailModel = new BasicEmailViewModel()
{
    EmailMessage = "(Message from: " + model.Name + ")<br><br>" + model.Message
};

string messageBody = ViewRenderer.RenderView("~/Views/Emails/Basic.cshtml", Email
```

It works great. However the "<br><br>" tags display in the email as "<br><br>" instead of 2 line spaces. It's as if they are being HTML encoded. Is there any way to add some basic HTML formatting like this and not encode it when I render the view?

Hope this makes sense! Thanks!



Rick Strahl
July 17, 2016

# re: Rendering ASP.NET MVC Views to String

@Travis - Razor encodes all expressions by default. If you want raw HTML strings you have the use the @Html.Raw(yourExpression) to get the raw unexpanded text to be rendered.

Add a Comment