Views and more...

ASP.NET MVC – REVIEW

SEPT 7, 2016

Plan

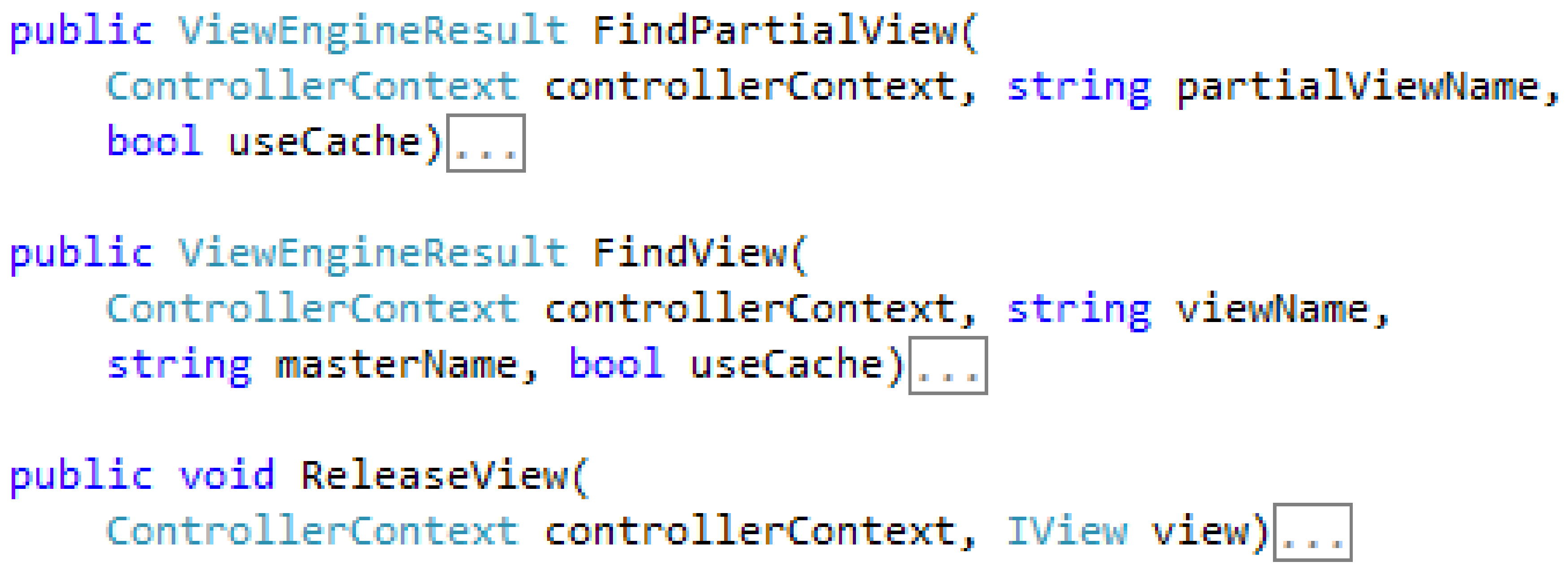
- Custom View Engine
- Razor View Engine
- Dynamic in Razor

Custom View Engine

public interface IViewEngine

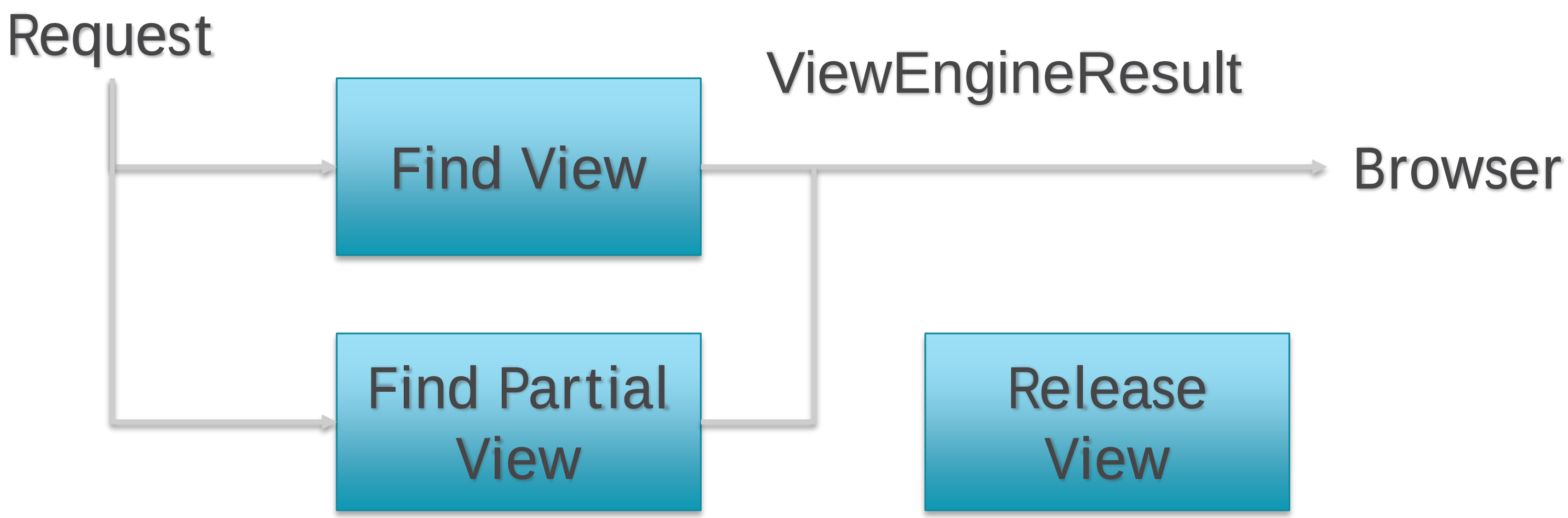
- Member of System.Web.Mvc
- Defines the methods that are required for a view engine.

Custom View Engine



```
public ViewEngineResult FindPartialView(
    ControllerContext controllerContext, string partialViewName,
    bool useCache)...

public ViewEngineResult FindView(
    ControllerContext controllerContext, string viewName,
    string masterName, bool useCache)...

public void ReleaseView(
    ControllerContext controllerContext, IView view)...
```

Custom View Engine



Custom View Engine

Note. View engines is implemented by the ‘ControllerActionInvoker’. If you have implemented your own action invoker or controller factory directly from the ‘IActionInvoker’ or ‘IControllerFactory’ interfaces you won’t have action to that feature.

Custom View Engine

ViewEngineResult

If view is not found:

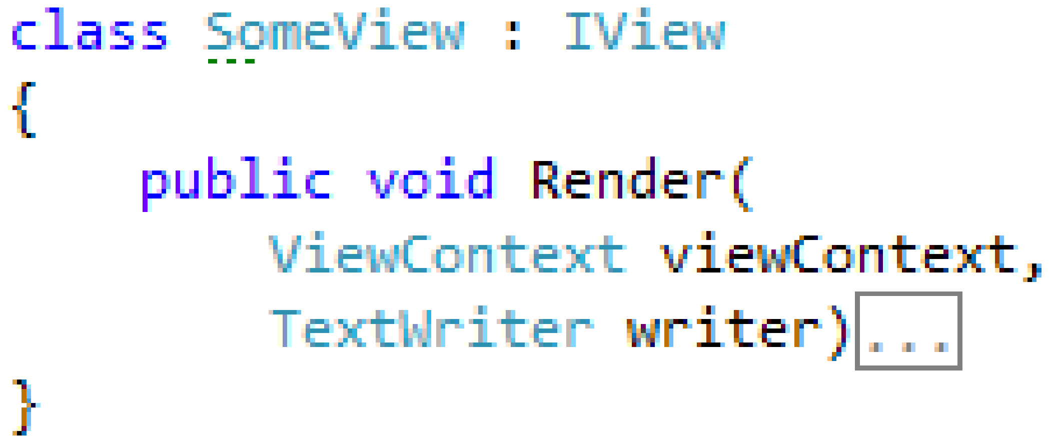
```
public ViewEngineResult(IEnumerable<string> searchedLocations)...
```

If view is found:

```
public ViewEngineResult(IView view, IViewEngine viewEngine)...
```

Custom View Engine

IView



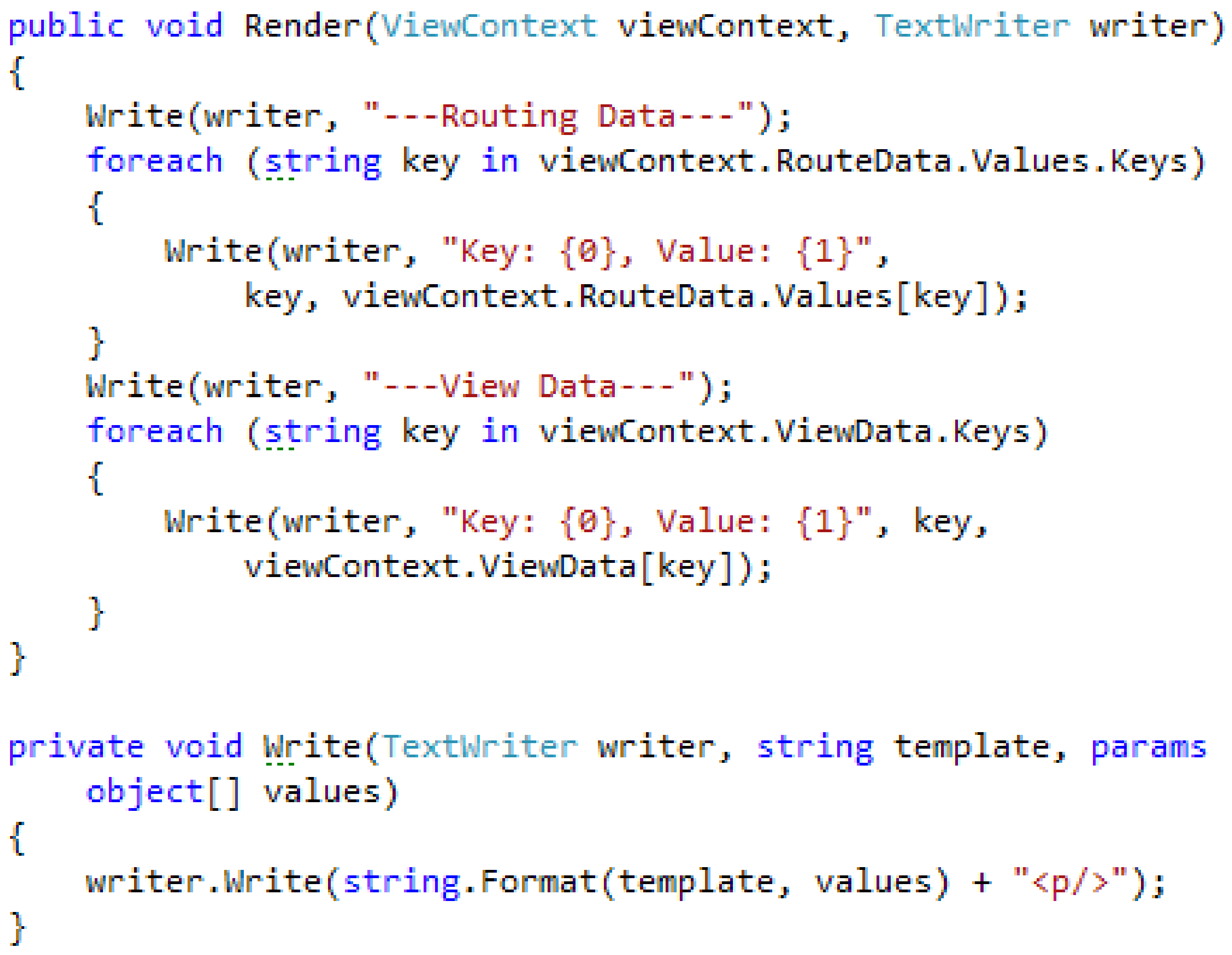
```
class SomeView : IView
{
     public void Render(
          ViewContext viewContext,
          TextWriter writer)...
}
```

Custom View Engine

- **Controller** - Returns the IController implementation that processed the current request
- **RequestContext** - Returns details of the current request
- **RouteData** - Returns the routing data for the current request
- **TempData** - Returns the temp data associated with the request
- **View** - Returns the implementation of the IView interface that will process the request.
- **ViewBag** - Returns an object that represents the view bag
- **ViewData** - Returns a dictionary of the view model data, which also contains the view bag and meta data for the model

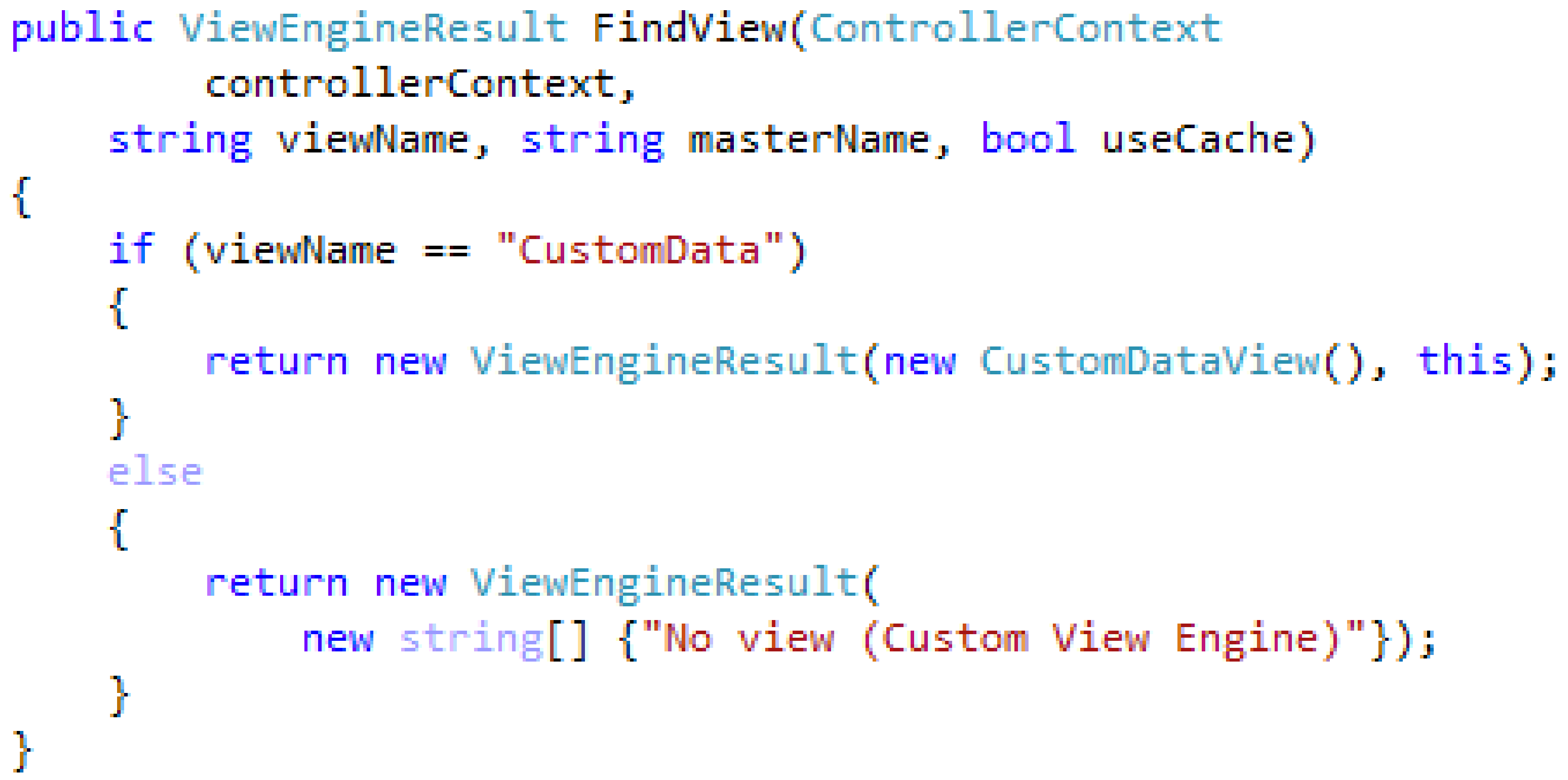
Project set up

Project set up – Custom Data View



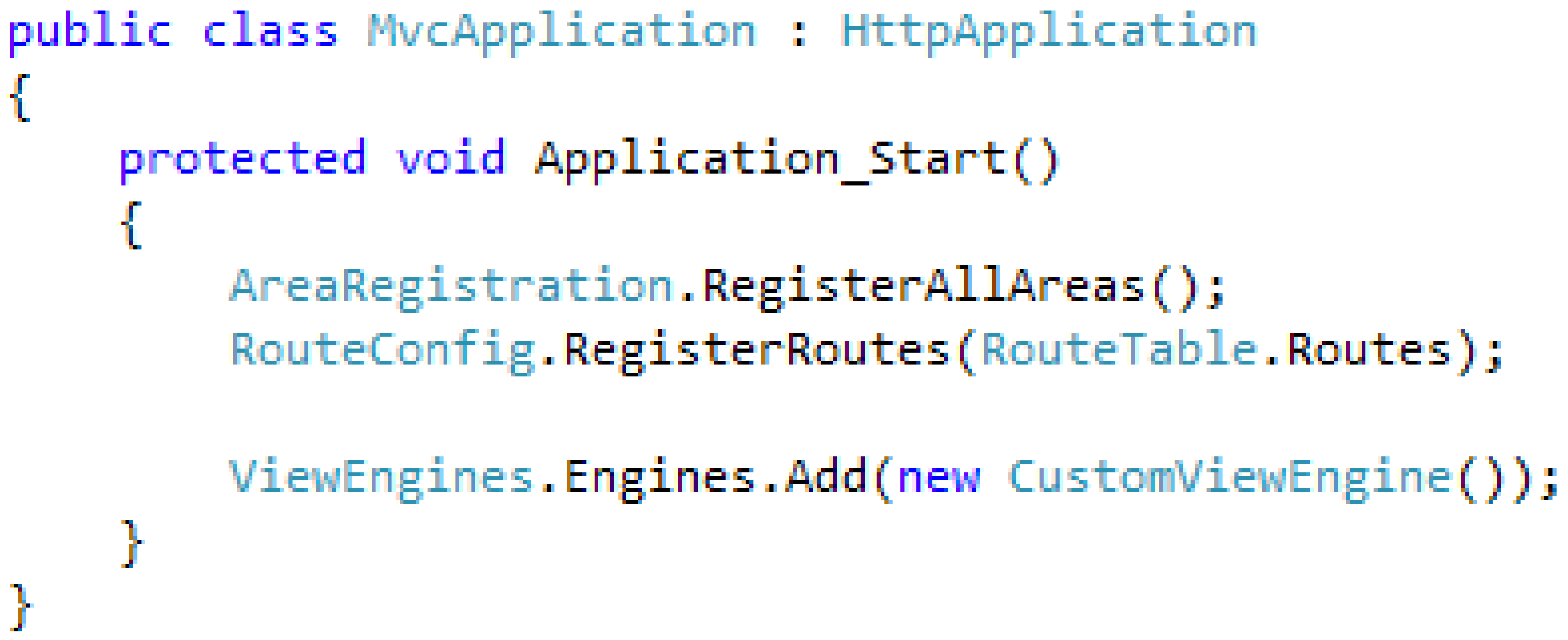
```
public void Render(ViewContext viewContext, TextWriter writer)
{
    Write(writer, "---Routing Data---");
    foreach (string key in viewContext.RouteData.Values.Keys)
    {
        Write(writer, "Key: {0}, Value: {1}",
            key, viewContext.RouteData.Values[key]);
    }
    Write(writer, "---View Data---");
    foreach (string key in viewContext.ViewData.Keys)
    {
        Write(writer, "Key: {0}, Value: {1}", key,
            viewContext.ViewData[key]);
    }
}

private void Write(TextWriter writer, string template, params
    object[] values)
{
    writer.Write(string.Format(template, values) + "<p/>");
}
```

Project set up – Custom View Engine



```
public ViewEngineResult FindView(ControllerContext
        controllerContext,
    string viewName, string masterName, bool useCache)
{
    if (viewName == "CustomData")
    {
        return new ViewEngineResult(new CustomDataView(), this);
    }
    else
    {
        return new ViewEngineResult(
            new string[] {"No view (Custom View Engine)"});
    }
}
```

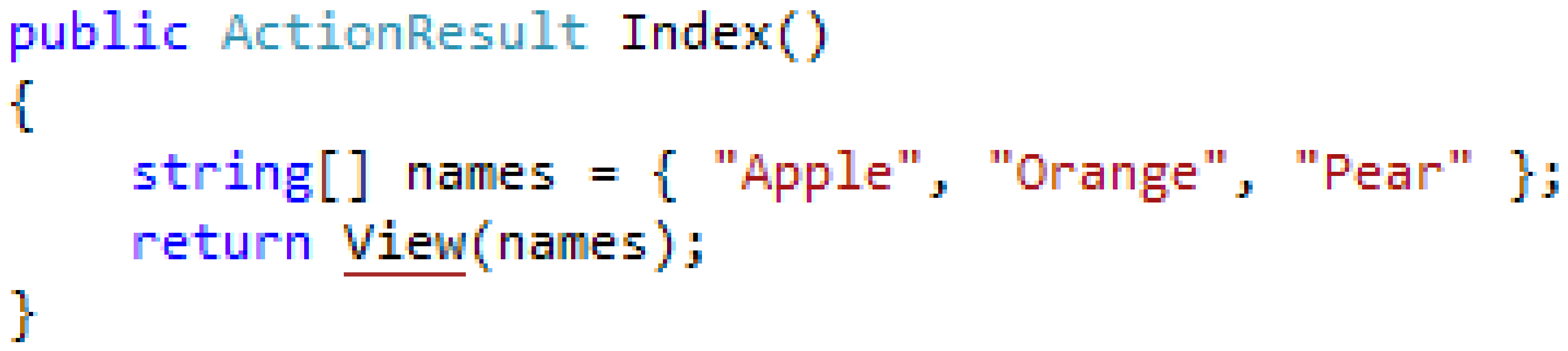
Project set up – Registering View Engine



```
public class MvcApplication : HttpApplication
{
    protected void Application_Start()
    {
        AreaRegistration.RegisterAllAreas();
        RouteConfig.RegisterRoutes(RouteTable.Routes);

        ViewEngines.Engines.Add(new CustomViewEngine());
    }
}
```

Working with Razor

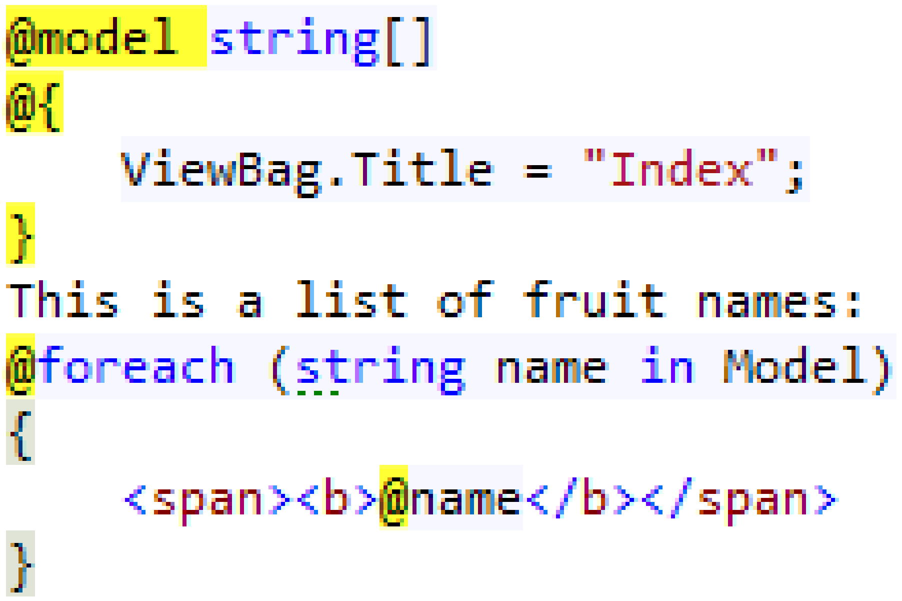
Create new home controller with Index action:



```
public ActionResult Index()
{
    string[] names = { "Apple", "Orange", "Pear" };
    return View(names);
}
```

Working with Razor

Add a simple index view to display results:



```
@model string[]
@{
    ViewBag.Title = "Index";
}
This is a list of fruit names:
@foreach (string name in Model)
{
    <span><b>@name</b></span>
}
```

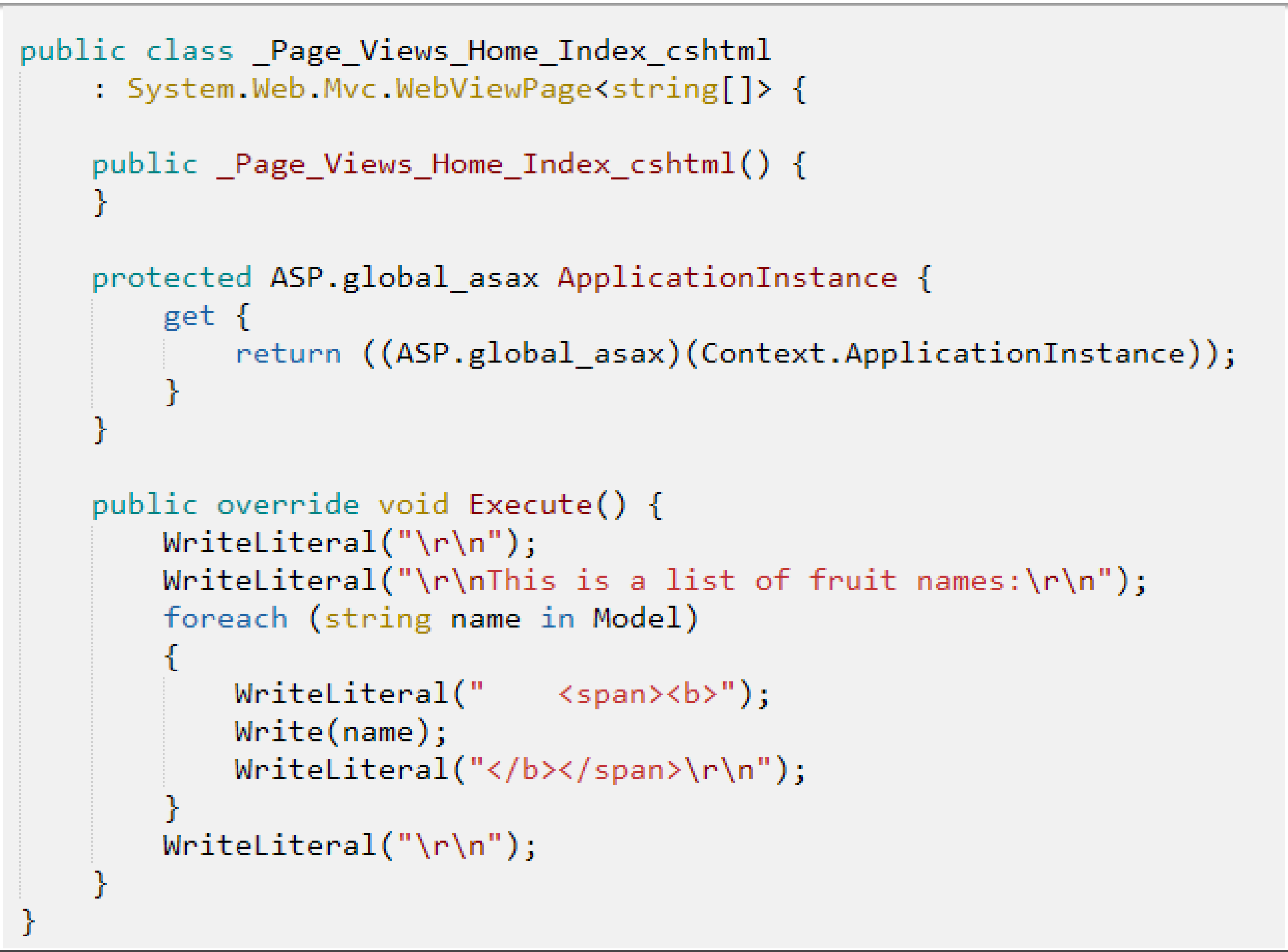
Working with Razor

Question:

- Were is the IndexView class that implements the IView interface?

- C:\Users\USERNAME\AppData\Local\Temp\Temporary ASP.NET Files\root\ *1f7230ac\ccb38794\App_Web_zy0hqxj0.0.cs*

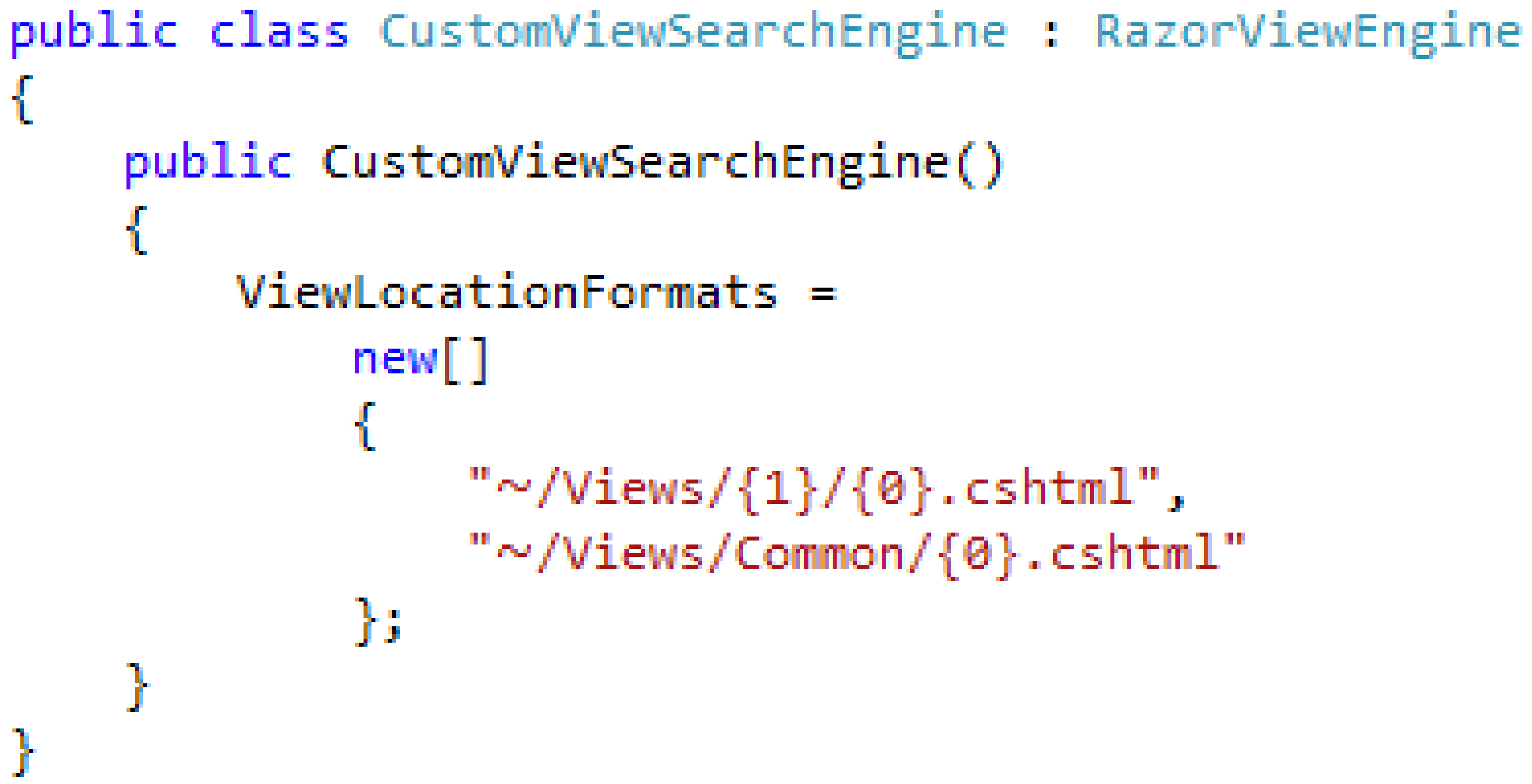
Working with Razor



```
public class _Page_Views_Home_Index_cshtml
    : System.Web.Mvc.WebViewPage<string[]> {

    public _Page_Views_Home_Index_cshtml() {
    }

    protected ASP.global_asax ApplicationInstance {
        get {
            return ((ASP.global_asax)(Context.ApplicationInstance));
        }
    }

    public override void Execute() {
        WriteLiteral("\r\n");
        WriteLiteral("\r\nThis is a list of fruit names:\r\n");
        foreach (string name in Model)
        {
            WriteLiteral("    <span><b>");
            Write(name);
            WriteLiteral("</b></span>\r\n");
        }
        WriteLiteral("\r\n");
    }
}
```

Configuring Razor

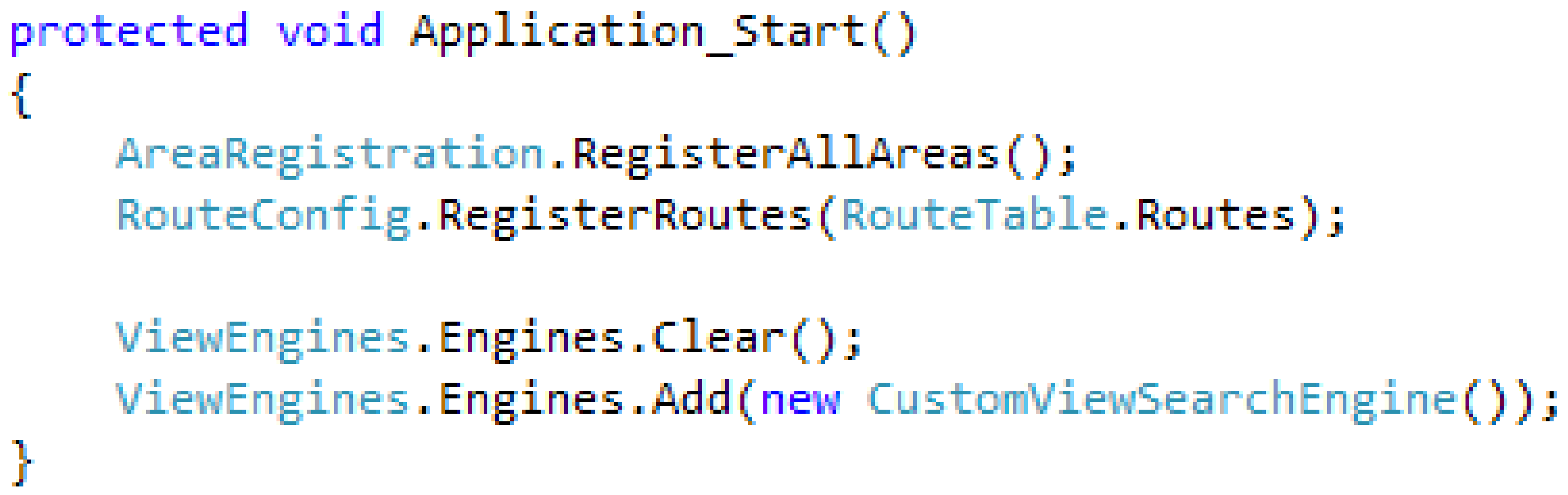
Changing view search locations:



```
public class CustomViewSearchEngine : RazorViewEngine
{
    public CustomViewSearchEngine()
    {
        ViewLocationFormats =
            new[]
            {
                "~/Views/{1}/{0}.cshtml",
                "~/Views/Common/{0}.cshtml"
            };
    }
}
```

Configuring Razor

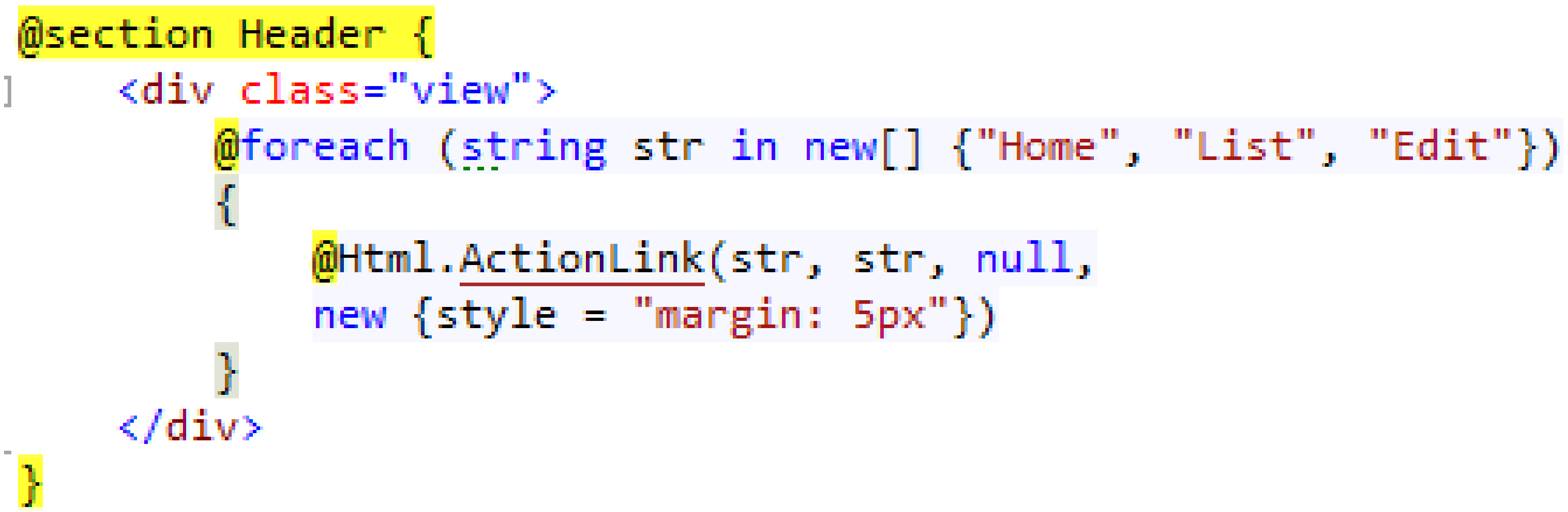
Changing view search locations:



```
protected void Application_Start()
{
    AreaRegistration.RegisterAllAreas();
    RouteConfig.RegisterRoutes(RouteTable.Routes);

    ViewEngines.Engines.Clear();
    ViewEngines.Engines.Add(new CustomViewSearchEngine());
}
```

Using Sections

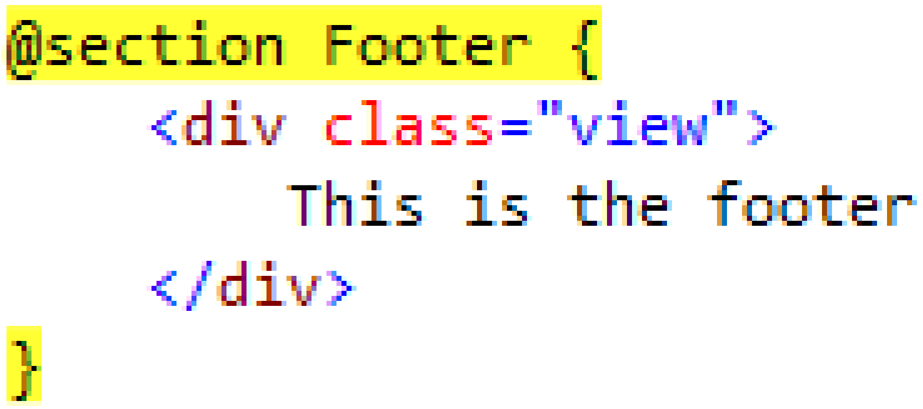
Sections will allow you to provide regions of content within a layout.



```
@section Header {
    <div class="view">
        @foreach (string str in new[] {"Home", "List", "Edit"})
        {
            @Html.ActionLink(str, str, null,
            new {style = "margin: 5px"})
        }
    </div>
}
```

Using Sections

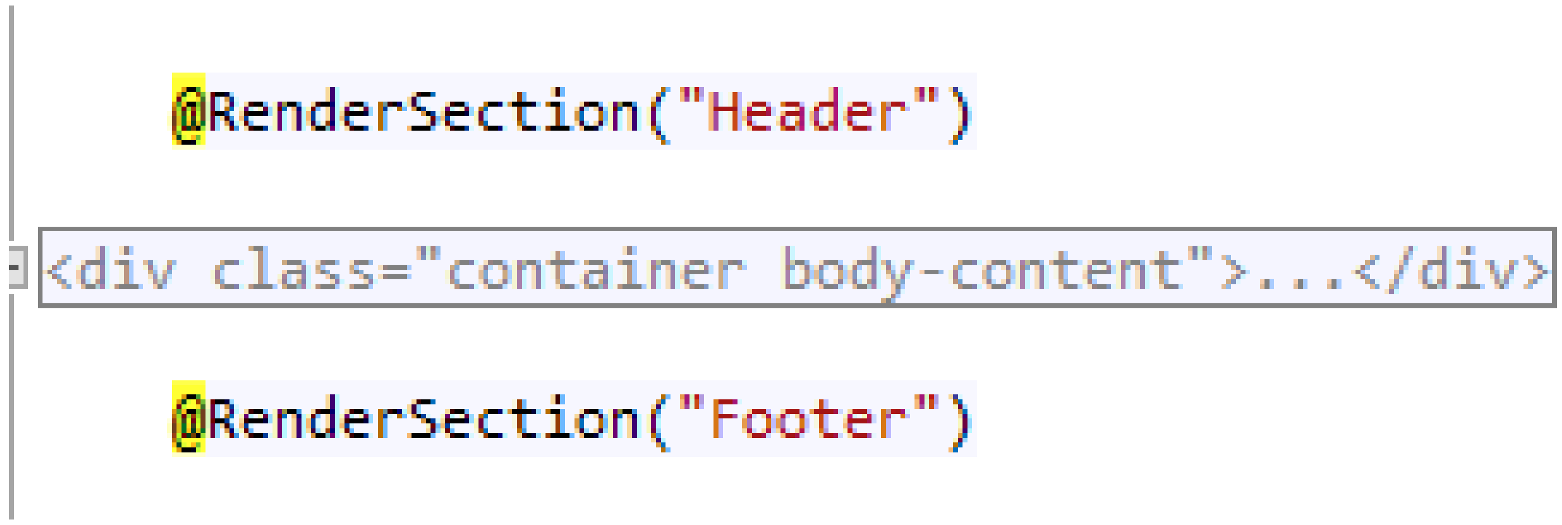
Sections will allow you to provide regions of content within a layout.



```
@section Footer {
    <div class="view">
        This is the footer
    </div>
}
```

Using Sections

Sections will allow you to provide regions of content within a layout.



```
@RenderSection("Header")
<div class="container body-content">...</div>
@RenderSection("Footer")
```

Using Sections

Sections will allow you to provide regions of content within a layout.

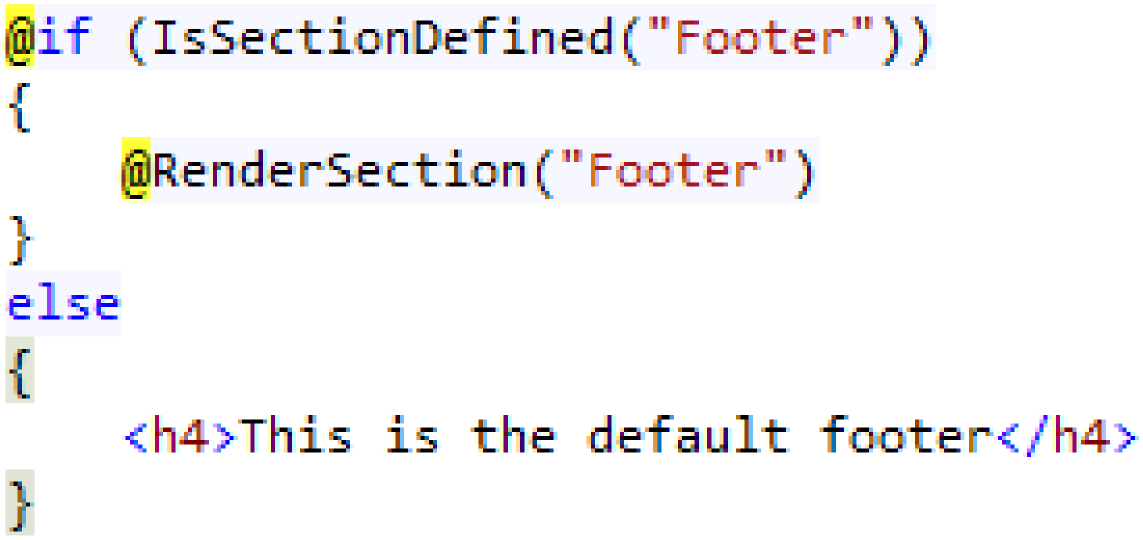
- RenderSection will be replaced with content from the appropriate section of triggered view.
- RenderBody will display everything that is not in sections.
- A view can define only the sections that are referred to in the layout.

Using Sections

- Best practice when defining section is not to mix them up with the rest of the view. Section should be defined either at the start of the view or on its end.
- In such case a common practice is to replace RenderBody with helper method RenderSection(“body”).

Using Sections

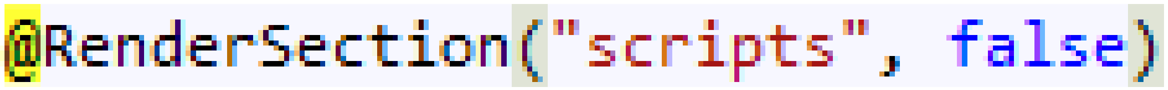
Verify that section exists:



```
@if (IsSectionDefined("Footer"))
{
    @RenderSection("Footer")
}
else
{
    <h4>This is the default footer</h4>
}
```

Using Sections

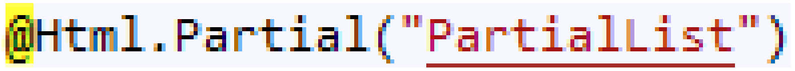
Display only if section exists:



```
@RenderSection("scripts", false)
```

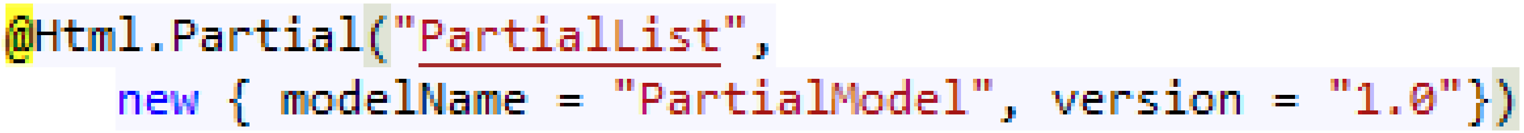
Razor and Partials

Calling Partial view:



```
@Html.Partial("PartialList")
```

Calling strongly typed Partial view:



```
@Html.Partial("PartialList",
    new { modelName = "PartialModel", version = "1.0"})
```

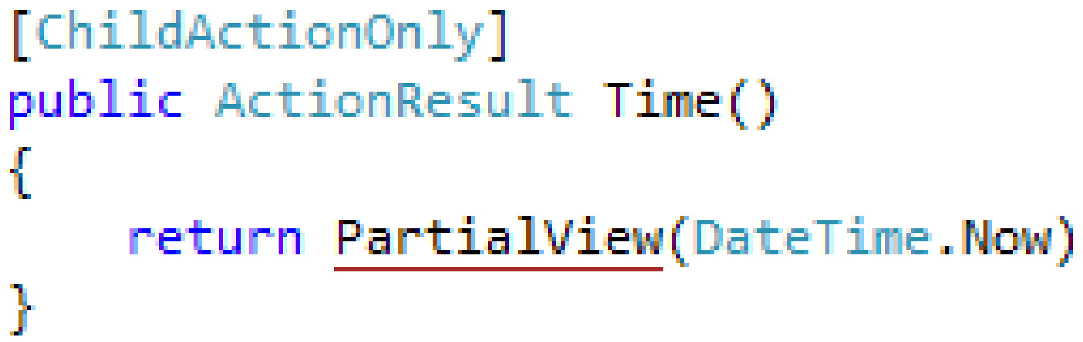
Note. The Razor View Engine looks for partial views in the same way that it looks for regular views (in the ~/Views/<controller> and ~/Views/Shared folders).

Razor and Child Actions

You can use a child action whenever you want to display some data-driven widget that appears on multiple pages and contains data unrelated to the main action that is running.

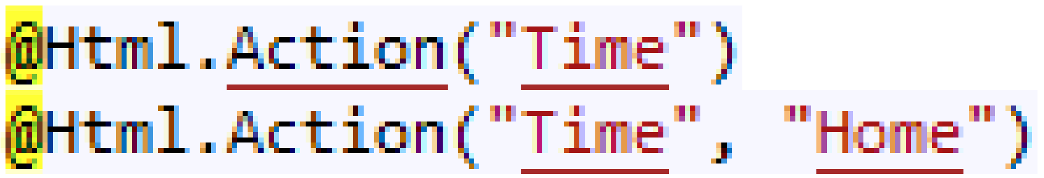
Razor and Child Actions

Controller part:



```
[ChildActionOnly]
public ActionResult Time()
{
    return PartialView(DateTime.Now)
}
```

View part:



```
@Html.Action("Time")
@Html.Action("Time", "Home")
```

Task

ASP.NET MVC Application

- Person View (shared):
    - Header (Partial) – Name of the faction, icon. (based on person faction)
    - Body – Some person info
    - Footer – Join other side yes/no
- Layout:
    - Navigation bar text – NavBar section, should be based on faction.
    - Scripts section:
        - If light side load light css class (make light layout). Footer action changes side.
        - If dark side load dark css class (make dark layout). Footer action calls popup message “LOL. There is no way out! PS. Your HR department has been contacted”.