Creating your own MVC View Engine For MVC Application

<http://www.codeproject.com/Articles/294297/Creating-your-own-MVC-View-Engine-into-MVC-Applica>

By **[Jigar Bagadai](http://www.codeproject.com/script/Membership/View.aspx?mid=7786956)**, 4 Dec 2011

|  |  |  |  |
| --- | --- | --- | --- |
| |  |  | | --- | --- | |  |  | |  |

Top of Form

* [**Download source - 2.33 MB**](http://www.codeproject.com/KB/aspnet/MyMVCViewEngine/CustomMVCViewEngine.zip)

Introduction

The View engines used in ASP.NET MVC Framework are the Razor View Engine and Web Form View Engine. Razor View engine used *.cshtml* and *.vbhtml*. While Web Form View Engine used *.aspx* to design the layout of the user interface.

The ASP.NET MVC Framework was designed to support alternative view engines and there are already several open source alternatives to the web forms view engine like Nhaml (pronounced enamel), spark, Brail, nVelocity. The different View Engines enable to write your view in different ways.

It is possible to use multiple view engines used in one MVC application. For that, it is required to registering multiple engines in the *global.aspx* file.

Custom View Engines

It is very simple to create your own custom view engine. When you create your own view engine, you have to just think about how you want to write your views.

The easiest approach to create custom view engine is just derive a new view engine from abstract VirtualPathProviderViewEngine Class. This base class can take care of the all low-level mechanics of finding and caching views.

Now take a simple example of MyViewEngine it will return simple view.

The first step is to create an empty MVC application. Then add a class file named *MyViewEngine.cs* and inherit that class from VirtualPathProviderViewEngine and override createview and createpartialviewmethods. These methods return an instance of the MYView Class. Your class will look like below:

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/294297/Creating-your-own-MVC-View-Engine-into-MVC-Applica)

public class MyViewEngine : VirtualPathProviderViewEngine

{

public MyViewEngine()

{

*// Define the location of the View file*

this.ViewLocationFormats = new string[]

{ "~/Views/{1}/{0}.myview", "~/Views/Shared/{0}.myview" };

this.PartialViewLocationFormats = new string[]

{ "~/Views/{1}/{0}.myview", "~/Views/Shared/{0}.myview" };

}

protected override IView CreatePartialView

(ControllerContext controllerContext, string partialPath)

{

var physicalpath = controllerContext.HttpContext.Server.MapPath(partialPath);

return new MyView(physicalpath);

}

protected override IView CreateView

(ControllerContext controllerContext, string viewPath, string masterPath)

{

var physicalpath = controllerContext.HttpContext.Server.MapPath(viewPath);

return new MyView(physicalpath);

}

}

Note that in the constructor, we set two properties of the base class. These properties indicate where the view engine should search to find a matching view or partial view.The parameter {1} represents the name of the controller and the parameter {0} represents the name of the action.

Now, create another class named MyView which implements IView interface. This class actually renders the view. MYView class code looks like below:

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/294297/Creating-your-own-MVC-View-Engine-into-MVC-Applica)

public class MyView : IView

{

private string \_viewPhysicalPath;

public MyView(string ViewPhysicalPath)

{

\_viewPhysicalPath = ViewPhysicalPath;

}

#region IView Members

public void Render(ViewContext viewContext, System.IO.TextWriter writer)

{

*//Load File*

string rawcontents = File.ReadAllText(\_viewPhysicalPath);

*//Perform Replacements*

string parsedcontents = Parse(rawcontents, viewContext.ViewData);

writer.Write(parsedcontents);

}

#endregion

public string Parse(string contents, ViewDataDictionary viewdata)

{

return Regex.Replace(contents, "\\{(.+)\\}", m => GetMatch(m,viewdata));

}

public virtual string GetMatch(Match m, ViewDataDictionary viewdata)

{

if (m.Success)

{

string key = m.Result("$1");

if (viewdata.ContainsKey(key))

{

return viewdata[key].ToString();

}

}

return string.Empty;

}

}

Render method of the class loads the view files and injects view data into the view, and writes that result into a text writer.

Before use, custom view engine is required to register view engine into *Global.asax* file using the following code:

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/294297/Creating-your-own-MVC-View-Engine-into-MVC-Applica)

protected void Application\_Start()

{

AreaRegistration.RegisterAllAreas();

RegisterGlobalFilters(GlobalFilters.Filters);

RegisterRoutes(RouteTable.Routes);

*//Register your View Engine Here.*

ViewEngines.Engines.Add(new MyViewEngine());

}

Now create a controller file into controller folder named myViewController and define index action into the controller. Your controller class will look like below:

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/294297/Creating-your-own-MVC-View-Engine-into-MVC-Applica)

public class myViewController : Controller

{

*//*

*// GET: /myView/*

public ActionResult Index()

{

ViewData["Message"] = "Hello World!";

return View();

}

}

Now add the simple HTML File into *View/MyView/* folder and give the name of the file like *index.myview*. Your view file markup looks like below:

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/294297/Creating-your-own-MVC-View-Engine-into-MVC-Applica)

<html>

<head>

<title>Index MyView </title>

</head>

<body>{message}

</body>

</html>

Now run the application and type URL /MyView /Index. You will get output of the Hello World! into the browser. The MyView class loads the *index.myview* file and replaces {message} with hello world! and renders the HTML Page.

Conclusion

After developing Custom View Engine, we can say that MVC team has done an awesome job at providing a very flexible framework for us to tweak and customize it so it fits our applications.

Bottom of Form