**MVC**

**Introduction**  
View Engine renders the HTML to the browser. The view engine templates have a different syntax than the implementation. By default ASP.Net MVC supports ASPX and the Razor View Engine. There are many more third-party view engines, like Spark, Nhaml and so on also available for MVC. We can also write our own view engine.  
  
**ASPX View Engine**  
  
The syntax used for writing a view with the ASPX View Engine is the same as the syntax used in ASP.Net web forms. The file extensions are also the same as for ASP.NET web forms (like .aspx, .ascx, .master). This view engine is the default view engine for MVC 1.0 and MVC 2.0. Implementing the unit testing framework with the ASPX View Engine is very difficult. ASPX uses "<%= %>" or "<%: %>" to render server-side content. We can choose any language with the CodeDom provider. There are on demand or precompiled views supported by an ASPX View Engine. The ASPX View Engine is also known as the Web Form View Engine.  
  
Loop and condition example with ASPX View Engine:

<ul>  
    <%foreach (var item in Products)  
        {  %>  
             <% if (item.IsInStock)  
                  { %>  
                         <p><%=item.ProductName%> is in stock</p>  
                   <% }  
                 else  
                  { %>  
                         <p><%=item.ProductName%> is not in stock</p>  
                  <% } %>  
    <%} %>  
</ul>  
  
**Razor View Engine**  
  
The Razor View Engine is an advanced view engine, available with MVC 3.0 and later versions. Razor uses the "@" character instead of "<% %>" as used by the ASPX View Engine. Razor does not require the code block to be closed, the Razor View Engine parsed itself and it is able to decide during run time that it is a presentation element (content) and that it is a code element. The Razor View Engine is compatible with a unit testing framework. The Razor template does not require the controller or webserver to host it, so views written in Razor are fully testable. The file extension of a Razor view is cshtml (for C#) and vbhtml (for VB.NET). By default all text from the @ expression is HTML encoded. Razor is not a new language. It is easy to learn. The main advantage of Razor, is that there is less transition between HTML and code because Razor provides an optimized syntax to generate HTML using a code focused templating approach.  
  
Loop and condition example with Razor View Engine:  
  
<ul>  
    @foreach (var item in Products)  
    {  
               @if(item.IsinStock)  
               {    
                   @item.ProductName is in stock  
               } else {  
                   @item.ProductName is in stock  
               }  
    }  
</ul>  
  
One of the disadvantages of Razor is, it is not supported by visual editors like Dream Viewer.

**Layout /MasterPage**

* In Razor View Engine we use Layouts.
* In ASPX View Engine we use masterPages.

**PartialPage /WebUserControl**

* In Razor View Engine we use PartialPage.
* In ASPX View Engine we use WebUserControls.

**Extension**

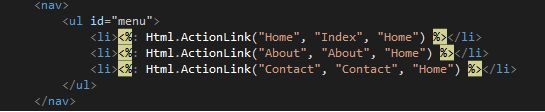
* Razor View Engine has .cshtml (with C#) and .vbhtml (with VB) extension for views, Layout and Partial views.
* ASPX View Engine has a similar extension as in a simple web application like .aspx for the views, .acsx for UserControls and .master for Master Pages.

**Performance**

* Razor Engine is a little slow compared to Aspx Engine.
* Aspx Engine is faster compared to Razor Engine.

**Syntax**

* ‘@’ symbol uses in Razor Engine to write the code. @Html.ActionLink("Login", "LoginView")
* ‘<%:’ delimiters use as starting point and ‘ %>’ use as ending point. You can write the code between them in ASPX Engine.

<%: Html.ActionLink("Login ", " LoginView ") %>   
  


**Cross-Site Scripting Attacks**

* Razor Engine prevents Cross-Site Scripting Attacks, in other words it encodes the script or HTML tags like <,> before rendering to view.
* ASPX Engine does not prevent Cross-Site Scripting Attacks, in other words any script saved in the database will be fired while rendering the page.

**Advantages of Razor View Engine**

* Easy to Learn: Razor is easy to learn. We can also use our existing HTML skills.
* It is Compact, Expressive, and Fluid. Razor helps us to minimize the coding and provide us a fast and fluid coding work flow.
* The parser (available with Razor) is smart enough. It is also able to decide at run time what is a code element and what is a content element.  
    
  For example, in the following code the @ character is also part of an email address, but Razor is smart enough to identify which is code and which is static content.  
    
  <p>  
          please contact to abc@gmail.com to more information  
         Current Date time : @DateTime.Now  
  </p>
* Razor is not a new language but it is markup so that we can also use Razor with any language like C# and VB.
* Razor also supports the concept of layout pages (the same as Master Pages in ASPX View Engine), that allows us to define a common site template, in other words a common look and feel across all the pages within a web site/application.
* Razor does not require any special tool to write markup. We can also write our markup code with any old plain text editor like Notepad.
* The Razor View Engine is designed such that it also supports unit test views without requiring a controller and web server. This can be hosted in any unit project. There is no special application domain required.
* ASP.NET MVC has HTML helpers that are methods that can be invoked within a code block. All existing HTML extension methods can be used with a Razor View Engine without any code changes.
* The code looks clean.
* Powerful built-in validation of markup that helps us to avoid unwanted runtime exceptions due to errors in the view.
* The Razor View Engine has the section concept that is equivalent to content placeholders in the ASPX View Engine and that can be optional.
* The @model directive provides a cleaner and more concise way to define a strongly typed model.  
    
  Example:  
  @model List<MyMVCapplication.EmployeeMaster>  
  or  
  @model MyMVCapplication.EmployeeDetails

**Razor View Engine VS ASPX View Engine**

|  |  |
| --- | --- |
| **Razor View Engine** | **ASPX View Engine (Web form view engine)** |
| The namespace used by the Razor View Engine is System.Web.Razor | The namespace used by the ASPX View Engine is System.Web.Mvc.WebFormViewEngine |
| The file extensions used by the Razor View Engine are different from a web form view engine. It uses cshtml with C# and vbhtml with vb for views, partial view, templates and layout pages. | The file extensions used by the Web Form View Engines are like ASP.Net web forms. It uses the ASPX extension to view the aspc extension for partial views or User Controls or templates and master extensions for layout/master pages. |
| The Razor View Engine is an advanced view engine that was introduced with MVC 3.0. This is not a new language but it is markup. | A web form view engine is the default view engine and available from the beginning of MVC |
| Razor has a syntax that is very compact and helps us to reduce typing. | The web form view engine has syntax that is the same as an ASP.Net forms application. |
| The Razor View Engine uses @ to render server-side content. | The ASPX/web form view engine uses "<%= %>" or "<%: %>" to render server-side content. |
| By default all text from an @ expression is HTML encoded. | There is a different syntax ("<%: %>") to make text HTML encoded. |
| Razor does not require the code block to be closed, the Razor View Engine parses itself and it is able to decide at runtime which is a content element and which is a code element. | A web form view engine requires the code block to be closed properly otherwise it throws a runtime exception. |
| The Razor View Engine prevents Cross Site Scripting (XSS) attacks by encoding the script or HTML tags before rendering to the view. | A web form View engine does not prevent Cross Site Scripting (XSS) attack. |
| The Razor Engine supports Test Driven Development (TDD). | Web Form view engine does not support Test Driven Development (TDD) because it depends on the System.Web.UI.Page class to make the testing complex. |
| Razor uses "@\* … \*@" for multiline comments. | The ASPX View Engine uses "<!--...-->" for markup and "/\* … \*/" for C# code. |
| There is only three transition characters with the Razor View Engine. | There are only three transition characters with the Razor View Engine. |
| The Razor View Engine is a bit slower than the ASPX View Engine. | |