Notes

Steps to create an application

1. Add Controller
2. Add View
3. Add Model
4. Accessing model’s data from a controller
5. Adding validation to the model

EF (Entity Framework)

1. Database First
2. Model First
3. Code First

Razor

A view-engine optimized around HTML generation using a code-focused template approach.

- **Compact, Expressive, and Fluid**: Razor minimizes the number of characters and keystrokes required in a file, and enables a fast, fluid coding workflow. Unlike most template syntaxes, you do not need to interrupt your coding to explicitly denote server blocks within your HTML. The parser is smart enough to infer this from your code. This enables a really compact and expressive syntax which is clean, fast and fun to type.

- **Easy to Learn**: Razor is easy to learn and enables you to quickly be productive with a minimum of concepts. You use all your existing language and HTML skills.

- **Is not a new language:** We consciously chose not to create a new imperative language with Razor. Instead we wanted to enable developers to use their existing C#/VB (or other) language skills with Razor, and deliver a template markup syntax that enables an awesome HTML construction workflow with your language of choice.

- **Works with any Text Editor**: Razor doesn’t require a specific tool and enables you to be productive in any plain old text editor (notepad works great).

- **Has great Intellisense**: While Razor has been designed to not require a specific tool or code editor, it will have awesome statement completion support within Visual Studio. We’ll be updating Visual Studio 2010 and Visual Web Developer 2010 to have full editor intellisense for it.

- **Unit Testable**: The new view engine implementation will support the ability to unit test views (without requiring a controller or web-server, and can be hosted in any unit test project – no special app-domain required).

Syntax/Sample	Razor	Web Forms Equivalent (or remarks)
Code Block	`@{` `  int x = 123;` `  string y = "because.";` `}`	`<%` `  int x = 123;` `  string y = "because.";` `%>`
Expression (Html Encoded)	`<span>@model.Message</span>`	`<span><%: model.Message %></span>`
Expression (Unencoded)	`<span>` `@Html.Raw(model.Message)` `</span>`	`<span><%= model.Message %></span>`
Combining Text and markup	`@foreach(var item in items) {` `  <span>@item.Prop</span>` `}`	`<% foreach(var item in items) { %>` `  <span><%: item.Prop %></span>` `<% } %>`
Mixing code and Plain text	`@if (foo) {` `  <text>Plain Text</text>` `}`	`<% if (foo) { %>` `  Plain Text` `<% } %>`
Mixing code and plain text (alternate)	`@if (foo) {` `  @:Plain Text is @bar` `}`	Same as above
Email Addresses	`Hi philha@example.com`	Razor recognizes basic email format and is smart enough not to treat the @ as a code delimiter
Explicit Expression	`<span>ISBN@(isbnNumber)</span>`	In this case, we need to be explicit about the expression by using parentheses.
Escaping the @ sign	`<span>In Razor, you use the` `@@foo to display the value` `of foo</span>`	@@ renders a single @ in the response.
Server side Comment	`@*` `This is a server side` `multiline comment` `*@`	`<%--` `This is a server side` `multiline comment` `--%>`
Calling generic method	`@(MyClass.MyMethod<AType>())`	Use parentheses to be explicit about what the expression is.
Creating a Razor Delegate	`@{` `  Func<dynamic, object> b =` `   @<strong>@item</strong>;` `}` `@b("Bold this")`	Generates a Func<T, HelperResult> that you can call from within Razor. See this blog post for more details.
Mixing expressions and text	`Hello @title. @name.`	`Hello <%: title %>. <%: name %>.`

Acronyms

1. EF: Entity Framework
2. ADO.NET: Active Directory Object
3. ORM: Object/Relational Mapping
4. DDL: Data definition language
5. POCO: Plain Old CLR Objects
6. CLR: Common Language Runtime
7. ASP: Active Server Pages

Glossary

1. Entity Framework: The Microsoft ADO.NET Entity Framework is an Object/Relational Mapping (ORM) framework that enables developers to work with relational data as domain-specific objects, eliminating the need for most of the data access plumbing code that developers usually need to write. Using the Entity Framework, developers issue queries using LINQ, then retrieve and manipulate data as strongly typed objects. The Entity Framework's ORM implementation provides services like change tracking, identity resolution, lazy loading, and query translation so that developers can focus on their application-specific business logic rather than the data access fundamentals.

References

1. Entity Framework, http://www.asp.net/entity-framework#Getting Started
2. Razor, http://weblogs.asp.net/scottgu/archive/2010/07/02/introducing-razor.aspx
3. Basic ASP.Net Tutorial, http://www.asp.net/mvc/tutorials/mvc-4/getting-started-with-aspnet-mvc4/intro-to-aspnet-mvc-4
4. Intermediate ASP.Net tutorial, http://www.asp.net/mvc/tutorials/getting-started-with-ef-using-mvc/creating-an-entity-framework-data-model-for-an-asp-net-mvc-application