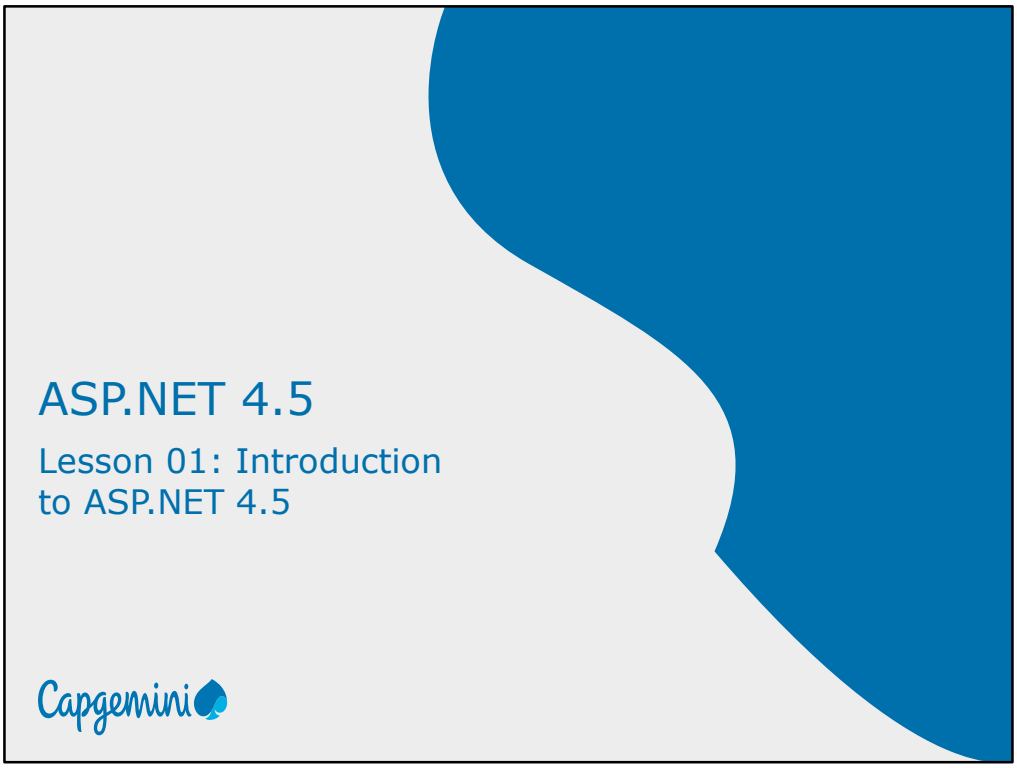

ASP.NET 4.5
Lesson 01: Introduction
to ASP.NET 4.5
Capgemini

Lesson Objectives

- In this lesson, you will learn:
  - Basic ASP.NET concepts, ASP.NET Framework
  - Web Forms and their features
  - Life Cycle of a Web Form
  - Advantages of ASP.NET over ASP
  - Concept of CodeBehind, Web Configuration File
  - Page Life Cycle

1.1: Introduction to ASP.NET

Introduction

- ASP.NET is a unified Web development platform that provides the services necessary for developers to build enterprise-class Web applications
- ASP.NET is a part of the Microsoft .NET framework, and is a powerful tool for creating dynamic and interactive web pages
- ASP.NET is a sophisticated engine that uses Managed Code for front to back processing of Web Requests

Introduction to ASP.NET:

- ASP.NET is more than the next version of **Active Server Pages (ASP)**. It is a unified Web development platform that provides the services necessary for developers to build enterprise-class Web applications. The evolution of ASP.NET starts right from when **Active Server Pages 3.0** was introduced. The evolution continued with **ASP.NET 1.0** to **1.1** and then now we have **ASP.NET 4.5**.
- The primary goal of ASP.NET is to help you build powerful, secure, and dynamic applications using the least possible code.
- ASP.NET is a compiled .NET-based environment. You can author applications in any .NET compatible language, including Visual Basic, C# and JScript. Additionally, the entire .NET Framework is available to any ASP.NET application. Developers can easily access the benefits of these technologies, which include a managed Common Language Runtime environment, type safety, inheritance, and so on.
- ASP.NET has been designed to work seamlessly with **WYSIWYG HTML** editors and other programming tools, including **Microsoft Visual Studio.NET**. Not only does this make Web development easier, but also provides all the benefits that these tools have to offer. This includes a GUI that developers can use to drop server controls onto a Web page, as well as fully integrated debugging support
- **ASP.NET 4.5** was a major release of the product and was an integral part of the **.NET 4.5 Framework**. ASP.NET 4.5 was developed keeping in mind developers productivity, administration and management, as well as scalability and performance.

1.2: ASP.NET versus ASP

A Comparison

- ASP.NET provides for the following enhancements over ASP:
  - Enhanced performance through Compiled Code.
  - Extensive set of controls and class libraries
  - Runtime error handling through try-catch blocks
  - Scalability
  - Flexibility of using languages – C#, VB.Net
  - Caching facilities

ASP.NET versus ASP:
ASP.NET not only takes advantage of performance enhancements found in the .NET Framework and runtime, but also has been designed to offer significant performance improvements over ASP and other Web development platforms.

ASP.NET and ASP can be compared based on the following points:

- **Enhanced performance through Compiled Code:** All ASP.NET code is compiled rather than interpreted, which allows early binding, strong typing, and just-in-time (JIT) compiling to native code. Compiled code means applications run faster with more design-time errors trapped at the development stage.

- **Extensive set of controls and class libraries:** An extensive set of controls and class libraries allow rapid building of applications, plus user-defined controls allow commonly used templates, such as **menus**. Layout of these controls on a page is easier because most of it can be done visually in most editors. WYSIWYG editing, drag-and-drop server controls, and automatic deployment are just a few of the features that this powerful tool provides.

- **Runtime error handling through try catch blocks:** ASP.NET provides for improved run-time error handling, making use of exception handling with try-catch blocks.

1.2: ASP.NET versus ASP

A Comparison

ASP.NET	ASP
Compiled	Interpreted
Uses ADO.NET for DB Connectivity	Uses ADO for DB Connectivity
Code-behind feature	Mixed HTML and Coding
Purely object-oriented	Partially object-oriented
Full support for XML	No built-in support for XML

ASP.NET versus ASP:
Scalability: ASP.NET has been designed with scalability in mind, providing features specifically tailored to improve performance in clustered and multiprocessor environments. Furthermore, processes are closely monitored and managed by the ASP.NET runtime, so that if one process misbehaves (leaks, deadlocks), a new process can be created in its place, which helps keep your application constantly available to handle requests.

Flexibility of using language: ASP.NET leverages the multi-language capabilities of the .NET CLR, allowing web pages to be coded in VB.NET, C#, J#. Since ASP.NET is based on the common language runtime, the power and flexibility of that entire platform is available to Web application developers. The .NET Framework class library, Messaging, and Data Access solutions are all seamlessly accessible from the Web.
ASP.NET is also language-independent, so you can choose the language that best applies to your application or partition your application across many languages. Furthermore, common language runtime interoperability guarantees that your existing investment in COM-based development is preserved while migrating to ASP.NET.

Caching Facilities: ASP.NET provides the ability to cache the whole page or just parts of it to improve performance.
ASP.NET offers complete syntax and processing compatibility with ASP applications. Developers simply need to change file extensions from .asp to .aspx to migrate their files to the ASP.NET framework. They can also add ASP.NET functionality to their applications with ease, sometimes by simply adding just a few lines of code to their ASP files.

1.3: Features of ASP.NET

Salient Features

- Let us see some of the salient features of ASP.NET:
  - Configuration files in XML format
  - Ability to use the code-behind development model to separate business logic from presentation
  - Web Server Controls
  - Default authorization and authentication schemes for Web applications

Features of ASP.NET:

- Apart from the comparative features of ASP.NET and ASP, ASP.NET also offers a lot of other features.
- ASP.NET configuration settings are stored in XML-based files, which are human readable and writable. Each of your applications can have a distinct configuration file and you can extend the configuration scheme to suit your requirements. Since ASP.NET employs a text-based, hierarchical configuration system, it simplifies applying settings to your server environment and Web applications. Since configuration information is stored as plain text, new settings may be applied without the aid of local administration tools. This "zero local administration" philosophy extends to deploying ASP.NET applications as well. An ASP.NET application is deployed to a server simply by copying the necessary files to the server. No server restart is required, even to deploy or replace running compiled code.
- ASP.NET provides the ability to use the **code-behind** development model to separate business logic from presentation. ASP.NET provides a simple framework that enables Web developers to write logic that runs at the application level. Developers can write this code in either the **global.asax** text file or in a compiled class deployed as an assembly.
- ASP.NET provides the Web Server Controls. These are controls introduced by ASP.NET for providing the UI for the web form. These controls are state managed controls and are **WYSIWYG (What You See Is What You Get)** controls.
- The .NET Framework and ASP.NET provide default authorization and authentication schemes for Web applications. You can easily remove, add, or replace these schemes depending upon the needs of your application. With built in Windows authentication and per-application configuration, you can be assured that your applications are secure.
- ASP.NET delivers a well-factored architecture that allows developers to "**plug-in**" their code at the appropriate level. In fact, it is possible to extend or replace any subcomponent of the ASP.NET runtime with your own custom-written component. Implementing custom authentication or state services has never been easier.

1.4: An Entire look at ASP.NET

A Bird's Eye View

- At a high level, ASP.NET is a collection of .NET classes that collaborate to process an HTTP request and generate an HTTP response, classes are loaded from system assemblies, GAC assemblies, and local assemblies
- Developers can choose from two programming models while creating an ASP.NET application, or combine them as required They are as follows :
  - Web Forms
  - Web Services

An Entire Look at ASP.NET:

- At a high level, ASP.NET is a collection of .NET classes that collaborate to process an HTTP request and generate an HTTP response, classes are loaded from system assemblies, GAC assemblies, and local assemblies.
- To work with ASP.NET, you must build your own classes that integrate into the existing class structure. Developers of ASP.NET can choose from two programming models or combine them as required:
  - **Web Form (.aspx):** Web Forms allow you to build powerful forms-based Web pages. While building these pages, you can use ASP.NET server controls to create common UI elements and program them for common tasks. These controls allow you to rapidly build up a Web Form out of reusable built-in or custom components, simplifying the code of a page.
  - **Web Services(.asmx):** A Web service is a way to access server functionality remotely. Using services, businesses can expose programmatic interfaces to their data or business logic, which in turn can be obtained and manipulated by client and server applications. Web services enable the exchange of data in **client-server** or **server-server** scenarios, using standards like HTTP and XML messaging to move data across firewalls. Web services are not tied to a particular **component technology** or **object-calling convention**. As a result, programs written in any language, using any component model, and running on any operating system can access Web services.

- Though a developer will primarily start of with a WebForm, she/he will eventually work with other components of the application. In other words, ASP.NET is not just about Web Forms or Web Services.
- We will now take a look at how an ASP.NET application works.

1.5: Working of a typical ASP.NET Application

Flow in ASP.NET

- A request starts on the browser where the user types in a URL, or clicks on a hyperlink, or submits an HTML form (a GET/POST request)
- ASP.NET interfaces with IIS through an ISAPI extension at the lowest level
- ASP.NET routes the request to an appropriate handler which is responsible for handling the request

Working of a typical ASP.NET Application:

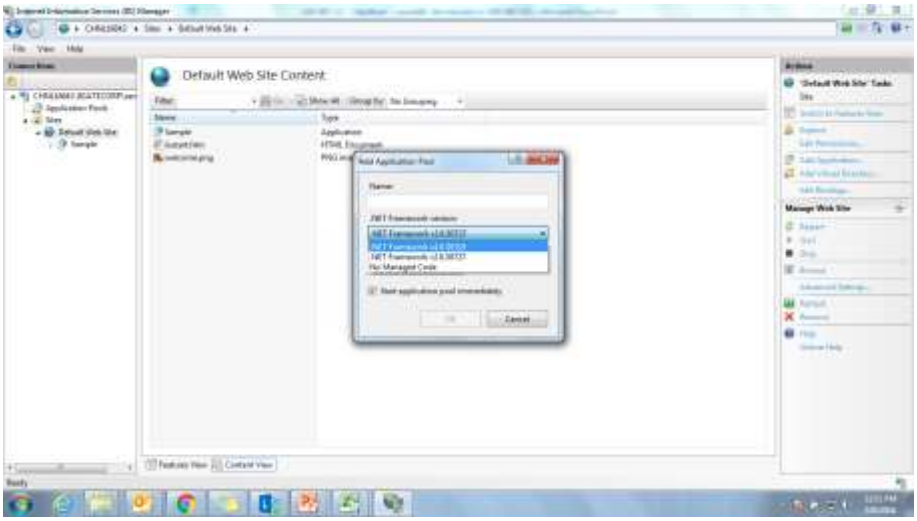
- ASP.NET is a request processing engine. It takes an incoming request and passes it through its internal pipeline to an end point where you as a developer can attach code to process that request.
- The runtime provides a complex yet very elegant mechanism for routing requests through this pipeline.
- The entire ASP.NET engine was completely built in managed code and all of the extensibility functionalities were provided via managed code extensions. This is a testament to the power of the .NET framework in its ability to build sophisticated and very performance oriented architectures.
- Above all, though the most impressive part of ASP.NET is the thoughtful design that makes the architecture easy to work with, yet it hooks into just about any part of the request processing.

Working of a typical ASP.NET Application (contd.):

- Let us see the working of a typical ASP.NET application:
  - A request starts on the browser where the user types in a URL, clicks on a hyperlink or submits an HTML form (a GET/POST request).
  - With a **GET request** the parameters are encoded in the URL, with a **POST request** they are transmitted in the body.
    - A GET request does not have a body whereas POST has body within which all data is passed.
  - On the Web Server, the **Internet Information Server (IIS)**picks up the request. At the lowest level, ASP.NET interfaces with **IIS** through an **ISAPI** extension. With ASP.NET this request usually is routed to a page with an **.aspx** extension. However, the manner in which the process works depends entirely on the implementation of the **HTTP Handler** that is set up to handle the specified extension.
  - In IIS, the **.aspx** is mapped through an "**Application Extension**" that is mapped to the **ASP.NET ISAPI dll** - aspnet_isapi.dll.
    - Every request that fires ASP.NET must go through an extension that is registered and points at **aspnet_isapi.dll**.
    - The request to an appropriate handler that is responsible for picking up requests depending on the extension which determines whether the request was for a Web Form or Web Service.
    - For ASP.NET the ISAPI dll is very lean and acts merely as a routing mechanism to pipe the inbound request into the ASP.NET runtime.
    - All the heavy lifting and processing, happens inside of the ASP.NET engine and your code.

1.6: Setting the .NET Version

Process Steps

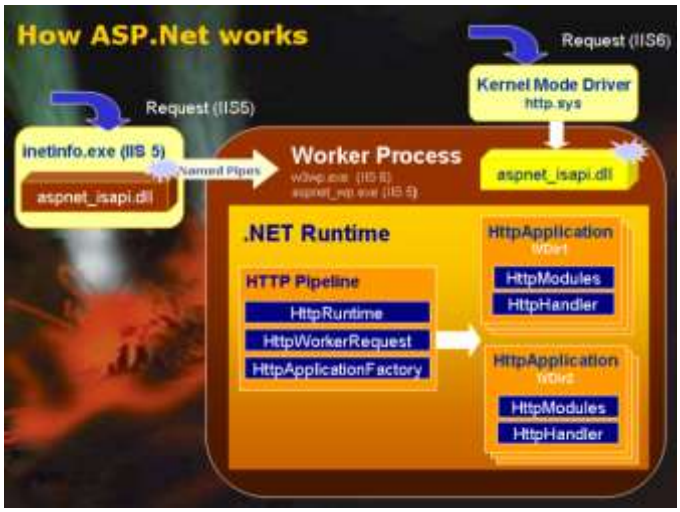


Setting the .Net Version:

- **ISAPI** is the initial code point that marks the beginning of an ASP.NET request. ASP.NET maps various **extensions** to its ISAPI extension which lives in the .NET Framework directory:
  - <.NET FrameworkDir>\aspnet_isapi.dll
- You should not set these **extensions** manually, since .NET requires a number of them. Instead, use the **aspnet_regiis.exe** utility to ensure that all the various scriptmaps get registered properly:
  - cd <.NetFrameworkDirectory>
  - aspnet_regiis – i
- Each version of the .NET framework has its own version of **aspnet_regiis**, and you need to run the appropriate one to register a site or virtual directory for a specific version of the .NET framework. Starting with ASP.NET 2.0, an **IIS ASP.NET configuration page** lets you pick the .NET version interactively in the IIS management console. The figure in the above slide shows the setting for .NET version.

1.7: Request Flow of ASP.NET

Process Steps



Once IIS intercepts the request and maps it to the worker process, the request follows a very specific path through the pipeline. The outline of the request's path through IIS 5.x and 6.x is this:

1. The request lands in IIS.
2. IIS routes the request to aspnet_isapi.dll.
3. ASP.NET packages the request context into an instance of *HttpContext*.
4. ASP.NET pipes the request through an instance of an *HttpApplication* object (or an *HttpApplication*-derived object).
5. If the application object is interested in receiving any of the request preprocessing events, *HttpApplication* fires the events to the application object. Any *HttpModules* that have subscribed to these events will receive the notifications as well.
6. Runtime instantiates a handler and handles the request.

HTTP Handler

HTTP Handler is the process which runs in response to a HTTP request. So whenever user requests a file it is processed by the handler based on the extension. So, custom http handlers are created when you need to specify handling based on the file name extension.

The ASP.NET page handler is only one type of handler. ASP.NET includes several other built-in handlers such as the Web service handler for .asmx files.

HTTP Modules

An HTTP module is an assembly that is called on every request that is made to your application.

So when a request is processed it is passed through all the modules in the pipeline of the request.

In IIS 6.0, the ASP.NET request pipeline is separate from the Web server request pipeline. In IIS 7.0, the ASP.NET request pipeline and the Web server request pipeline can be integrated into a common request pipeline. In IIS 7.0, this is referred to as *Integrated* mode.

Handler	Description
ASP.NET page handler (*.aspx)	The default HTTP handler for all ASP.NET pages.
Web service handler (*.asmx)	The default HTTP handler for Web service pages created as .asmx files in ASP.NET.
Generic Web handler (*.ashx)	The default HTTP handler for all Web handlers that do not have a UI and that include the @WebHandler directive.
Trace handler (trace.axd)	A handler that displays current page trace information.

Request Flow of ASP.NET:

- The figure on the above slide shows a request flow from IIS to the ASP.NET Runtime and through the request processing pipeline from a high level. When a request comes in, IIS checks for the script map and routes the request to the **aspnet_isapi.dll**. The operation of the DLL and the manner in which it gets to the ASP.NET runtime varies significantly between **IIS 5** and **6**.
  - **IIS 5** hosts **aspnet_isapi.dll** directly in the **inetinfo.exe process** or one of its isolated worker processes if you have isolation set to ***medium*** or ***high*** for the Web or virtual directory.
    - When the first ASP.NET request comes in the DLL, it will spawn a new process in another EXE – **aspnet_wp.exe** – and route processing to this spawned process.
    - This **spawned process** in turn loads and hosts the .NET runtime. Every request that comes into the ISAPI DLL then routes to this **worker process**.
  - **IIS 6** changes the processing model significantly in that IIS no longer hosts any foreign executable code like ISAPI extensions directly. Instead **IIS 6** always creates a separate worker process – an **Application Pool** – and all processing occurs inside of this process, including execution of the **ISAPI dll**.
    - Application Pools can be configured for every virtual directory or the entire website, so you can isolate every Web application easily into its own process that will be completely isolated from any other Web application running on the same machine.
    - In IIS 6, ISAPI extensions run in the **Application Pool worker process**.
    - In IIS 6, the worker process is **w3wp.exe**. The .NET Runtime also runs in this same process, so communication between the ISAPI extension and the .NET runtime happens in-process which is inherently more efficient than the named pipe interface that IIS 5 must use.
  - **Worker Process:** Worker Process (*w3wp.exe*) runs the ASP.Net application in IIS. This process is responsible to manage all the request and response that are coming from client system. All the ASP.Net functionality runs under the scope of worker process. When a request comes to the server from a client worker process is responsible to generate the request and response. In a single word we can say worker process is the heart of ASP.NET Web Application which runs on IIS.
  - **Application Pool:** Application pool is the container of worker process. Application pool is used to separate sets of IIS worker processes that share the same configuration. Application pool enables a better *security, reliability, and availability* for any web application. The worker process serves as the process boundary that separates each application pool so that when one worker process or application is having an issue or recycles, other applications or worker processes are not affected. This makes sure that a particular web application doesn't impact other web application as they are configured into different application pools.

- Although the IIS hosting models are very different, the actual interfaces into managed code are very similar.
- Apart from IIS, Visual Studio 2005 also permits deploying ASP.NET application in the built in server called Cassini.

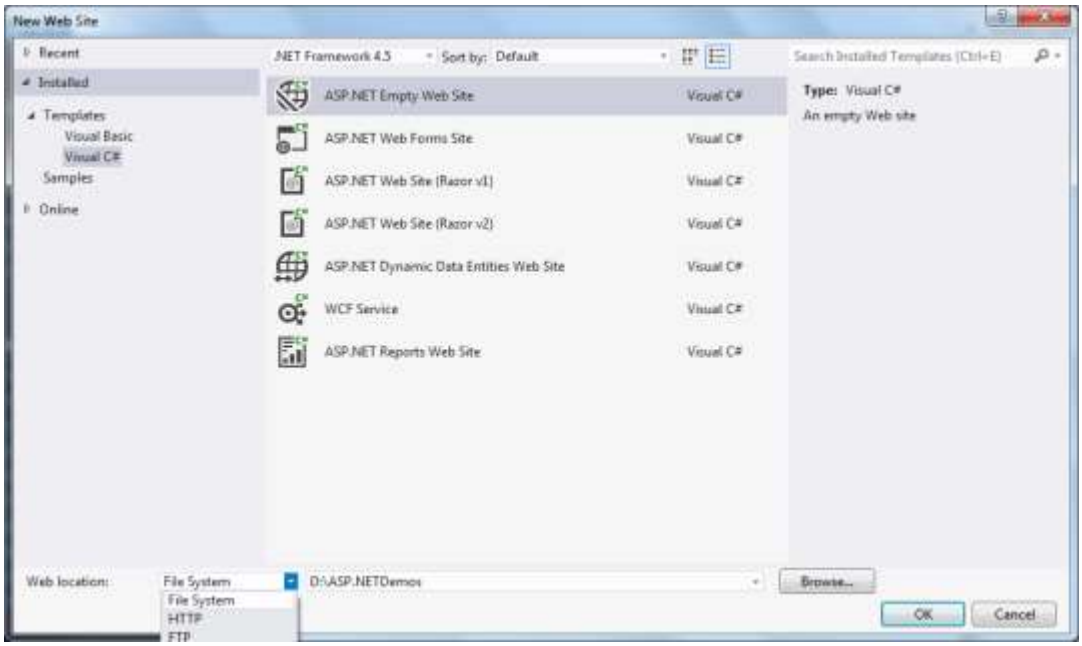
1.8: Demonstration: Hello World Demo

- Demo on ASP.NET application using Hello World

Demonstration: HelloWorld:
Let us take a look at the first ASP.NET application. To create this application follow the steps given below:

Step 1: Open Visual Studio 2012. Select File → New-WebSite. It opens the New WebSite dialog box.
With ASP.NET 4.5, using Visual Studio 2012 you have an option to create an application with virtual directory mapped to IIS or a standalone application outside the confines of IIS.
Built-in Web Server:
In earlier version of ASP.Net 2.0, we used the built-in Web Server called Cassini. And now the ASP.Net 4.5 has IISExpress as its built-in Web Server.
By default, VS2012 builds applications without the use of IIS. The location provided by default for your application is 'C:\Documents and Settings\user\My Documents\Visual Studio 2012\WebSites'.
Optionally you can also create your own folder and allow your WebSites to be created in that location. If you use the Built-in Web Server, then you are not locked into the Websites folder or in the 'C:\Inetpub\wwwroot' folder. You can also run your application completely from this location. Hence with this new way of creating ASP.NET applications you are not dependent on having access to any IIS server and you can go ahead and develop applications on any Windows OS machine.
The figure on the following page shows the creation of website on the FileSystem, that is in the Built-in Web Server.

Demonstration: HelloWorld (contd.):



IIS: If you access IIS, then the application can also be created using IIS.
Since by default the **FileSystem** option is selected. You have to select **HTTP** option from the **Location** drop-down in the **New Website** dialog box.
As shown in the figure, after selecting the **Location** provide the name of your application in the adjacent textbox. For example: http://localhost/MyWebSite. This will create a Virtual Directory for you Web Site.
Now, the ASP.NET application is not dependent on the built-in webserver, however, it will use IIS for execution. When you invoke the application, the URL will be something as shown below: http://localhost/MyWebsite/default.aspx .

Step 2: Once the Website has been created using whichever option, you will have a **default.aspx** page as part of your project. The **default.aspx** page has a **Design View** and **Source View**. The page layout can be designed using the **Design View**. The html source code is seen in the **Source View**. Also if any Script code is required it can be written in the Source View. The **default.aspx.cs** is also created as part of the project and is affixed with the **aspx** page. The server side code has to be written here. For this application we do not have any UI layout but a small code which is written in **default.aspx.cs** file as follows:

```
public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        Response.Write("Hello World");
        Response.Write("Welcome to ASP.NET");
    }
}
```

Demonstration: HelloWorld (contd.):

Step 3 : After the layout is designed and corresponding code is written, you can build the application and execute.

(**Note:** Once the application is executed it goes through the entire process of execution which is specified in the subsequent slides).

1.9: Elements in an ASP.NET Application

List of Elements

- A web application in ASP.NET is a collection of pages, controls, code modules, and services all running under a single web server application directory
- The elements of a typical application would be as follows:
  - The ASP.NET Page Framework (Web Forms)
  - Web and HTML Server Controls
  - Shared Code Modules or Assemblies

Elements in an ASP.NET Application:

- A web application in ASP.NET is a collection of pages, controls, code modules, and services all running under a single web server application directory (usually IIS).
- ASP.NET makes it really easy to build the types of dynamic web applications that exist everywhere on the Internet today. It provides a simple programming model based on the .NET Framework and several built-in controls and services that enable many of the common scenarios found in most applications, with very little effort or code.
- In this section, we demonstrate the basic elements of a web application, including the following:
  - The ASP.NET Page Framework (Web Forms)
  - Web and HTML Server Controls
  - Shared Code Modules or Assemblies

1.10: ASP.NET Page

Web Form

- ASP.NET pages, officially known as "web forms", are the main building blocks for application development
- An ASP.NET page consists of the following elements:
  - Directives
  - Code declaration blocks
  - Code render blocks
  - ASP.NET server controls
  - HTML tags

ASP.NET Page:

- **ASP.NET pages**, known officially as "**web forms**", are the main building blocks for application development. The web forms are contained in files with an **ASPX** extension. Pages consist of **code** and **markup** and are dynamically compiled and executed on the server to produce a rendering to the requesting client browser. When a browser client requests .aspx resources, the ASP.NET runtime parses and compiles the target file into a .NET Framework class. This class is then used to produce the HTML that is sent back to the user.
- (Note that the .aspx file is compiled only the first time it is accessed. The compiled type instance is then reused across multiple requests).
- In the first application that we created, all the code simply executed on the server side and a response was generated to be displayed on the browser. However, as we move along we will be looking at various examples for embedding code within the aspx source code. Before we move on to understand all the contents of a Web form, take a look at the following example.

1.11: Directives

Concept of ASP.NET Directives

- ASP.NET directives are something that is a part of every ASP.NET page
- The behavior of ASP.NET page can be controlled with these directives
- The directive starts with %@. This is followed by directive name and its attribute, and ends with %

```
<%@ [Directive] [Attribute=Value] %>
```

Directives:

The **directives** section is one of the most important part of an ASP.NET page.

- They control the manner in which a page is compiled.
- They specify the manner in which a page is cached by web browsers.
- They aid debugging (error-fixing).
- They allow you to import classes to use within your page's code.

Each directive starts with <%@. This is followed by the directive name, plus any attributes and their corresponding values. The directive then ends with %>.

There are many directives available in ASP.NET as follows:

- **@Page :** The @Page directive enables you to specify attributes and values for an ASP.NET Page to be used when the page is parsed and compiled
- **@Master :** The @Master directive belongs to Master Pages that is .master files.
- **@Control :** The @Control directive is used when we build an ASP.NET user controls.
- **@Register :** The @Register directive associates aliases with namespaces and class names for notation in custom server control syntax.
- **@Reference :** The @Reference directive declares that another ASP.NET page or user control should be complied along with the current page or user control.
- **@PreviousPageType :** The @PreviousPageType is a new directive which makes excellence in ASP.NET pages. The concept of cross-page posting between ASP.NET pages is achieved by this directive
- **@OutputCache :** The @OutputCache directive controls the output caching policies of the ASP.NET page or user control.

Directives (contd.):

- **@Import :** The @Import directive allows you to specify any namespaces to the imported to the ASP.NET pages or user controls
- **@Implements :** The @Implements directive gets the ASP.NET page to implement a specified .NET framework interface.
- **@Assembly :** The @Assembly directive is used to make your ASP.NET page aware of external components
- **@MasterType :** To access members of a specific master page from a content page, you can create a strongly typed reference to the master page by creating a @MasterType directive

We will be discussing the directives at length as we come across them in corresponding topics. As of now, let us understand the **page directive** in detail.

Unlike ASP, ASP.NET directives can appear anywhere on a page. However, they are commonly included at its very beginning.

1.12: Page Directive

Usage of Page Directive

- The page directive enables you to specify attributes and values for an ASP.NET page
- Some of the attributes of the page directive are:

Language	CodeFile
ClassName	ErrorPage
Culture	UICulture
Theme	MasterPageFile
EnableViewState	Inherits
Title	ValidateRequest

Page Directive:
The @Page directive enables you to specify attributes and values for an ASP.NET Page to be used when the page is parsed and compiled. Every .aspx file should include this **@Page** directive to execute. This is the most frequently used directive. Since this is an important directive, many attributes belong to this directive. We will take a look at the few important attributes:

- **Language :** It defines the language used for any inline rendering and script blocks. Values can represent any .NET-supported language, including Visual Basic, C#, or JScript .NET.
- **CodeFile :** It specifies the code-behind file with which the page is associated.
- **ClassName :** It specifies the name of the class that is bound to the page when the page is compiled.
- **ErrorPage :** It specifies a target URL for redirection if an unhandled page exception occurs.
- **Culture :** It specifies the culture setting of the page. If you set to auto, it enables the page to automatically detect the culture required for the page.
- **UICulture :** The value of this attribute specifies the UICulture that has to be used for the ASP.NET page. With ASP.NET, if you set to “auto” it enables the page to automatically detect the culture required for the page.
- **Theme :** It applies the specified theme to the page using ASP.NET 2.0 themes feature.
- **MasterPageFile :** It specifies the location of the MasterPage file to be used with the current ASP.NET page.

Page Directive:

- **EnableViewState:** It indicates whether view state is maintained across page requests. true if view state is maintained; otherwise, false. The default is true.
- **Inherits:** It specifies a code-behind class for the page to inherit. This can be any class derived from the Page class.
- **Title:** It applies a Page Title. This page title is other than that specified in the master page.
- **ValidateRequest:** When this attribute is set to true, the form input values are checked against a list of potentially unsafe values. If a match occurs, then an HttpRequestValidationException Class is thrown. The default is true.

There are also other attributes which are of rarely use such as Buffer, CodePage, ClassName, EnableSessionState, Debug, Description, EnableTheming, EnableViewStateMac, TraceMode, WarningLevel, and so on.
Here is an example of how a @Page directive looks:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="HelloWorld1.aspx.cs"
Inherits="HelloWorld" Title="Title – Hello World" %>
```

1.13: Code Declaration Blocks

Usage of Code Declaration Blocks

- Code declaration blocks mostly contain all the application logic of your ASP.NET page
- It is sometimes also termed as Inline coding

Example:

```
<script runat="server">
 TestMethod()
{
  // Code here
}
</script>
```

Code Declaration Blocks:

- ASP.NET introduces the new code-behind model. However, if you are not working with code-behind pages, you must use code declaration blocks to contain all the application logic of your ASP.NET page. This application logic defines variables, subroutines, functions, and more. In our example on HelloWorld1, observe the code placed <script> tags.

```
<script runat="server">
 void mySub()
{
  // Code here
}
</script>
```

- We can set the language that is used in this code declaration block via the language attribute.

```
<script runat="server" language="C#">
```

- If you do not specify a language within the code declaration block, then the ASP.NET page will use the language provided by the language attribute of the Page directive. Each page's code must be written in a single language. For instance, it is not possible to mix VB and C# in the same page.

1.14: Code Render Blocks

Usage of Code Render Blocks

- ASP.NET provides syntax compatibility with existing ASP pages
- This includes support for <% %> code render blocks that can be intermixed with HTML content within an .aspx file

Example:

```
<% string title = "This is generated by a code render block."; %>
```

Code Render Blocks:

- You can use code render blocks to define **inline code** or **expressions** that will be executed when a page is rendered. The code render blocks provide syntax compatibility with existing ASP pages. Code within a code render block is executed immediately when it is encountered – usually when the page is loaded or rendered. On the other hand, code within a code declaration block (within <script> tags) is executed only when it is called or triggered by user or page interactions.
- There are two types of code render blocks – **inline code** and **inline expressions** – both are typically written within the body of the ASP.NET page. Unlike with ASP, the code used within the above <% %> blocks is actually compiled - not interpreted using a script engine. This results in an improved runtime execution performance.
- Inline code render blocks execute one or more statements, and are placed directly inside a page's HTML between <% and %> delimiters. In our example, we have used the code render block:

```
<% string Title = "This is generated by a code render block."; %>
```

1.15: ASP.NET Server Controls

Usage of ASP.NET Server Controls

- ASP.NET pages can contain server controls in addition to code and markup
- Server controls participate in the execution of the page and produce their own markup rendering to the client
- Three basic types of server control are available, namely:
  - ASP.NET Controls
  - HTML controls
  - Web User Controls

ASP.NET Server Controls:

- ASP.NET pages can contain server controls, which are programmable server-side objects that typically represent a UI element in the page, such as a **textbox** or **image** in addition to **Code** and **markup**. They represent dynamic elements with which your users can interact. Server controls participate in the execution of the page and produce their own markup rendering to the client. The principle advantage of server controls is that they enable developers to get complex rendering and behaviors from simple building-block components, dramatically reducing the amount of code it takes to produce a dynamic Web page.
- Another advantage of server controls is that it is easy to customize their rendering or behavior. Server controls expose properties that can be set either:
  - declaratively within the tag,
  - programmatically in the code
- Server controls also expose events that developers can handle to perform specific actions during the page execution or in response to a client-side action that posts the page back to the server. Server controls also simplify the problem of retaining state across round-trips to the server, automatically retaining their values across successive calls.
- Usually, an ASP.NET control must reside within a <form runat="server"> tag in order to function correctly. Some examples of implementing server controls are as follows:
  <form runat="server">,
  <asp:textbox runat=server>,
  <asp:dropdownlist runat=server> and <asp:button runat=server>.
- At run time, these server controls automatically generate HTML content.

ASP.NET Server Controls:

Controls offer the following advantages to ASP.NET developers:

- They give us the ability to:
  - access HTML elements easily from within our code
  - change these element characteristics
  - check their values
  - even update these values dynamically from server-side programming language of choice
- ASP.NET controls retain their properties. This is the result of a mechanism called **view state**. For now, you need to know that the **view state** prevents users from losing the data that they have entered into a form once that form has been sent to the server for processing. When the response comes back to the client, text box entries, drop-down list selections, and so on, are all retained through view state.
- With the **ASP.NET controls**, developers are able to separate a page's presentational elements from its application logic, so that each can be considered separately.
- Many ASP.NET controls can be "bound" to the data sources from which they will extract data for display with minimal coding effort.

We will get introduced to controls as we go along.

1.16: HTML Tags

Usage of HTML Tags

- ASP.NET page will also contain the traditional HTML tags

Example:

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server"><title>Hello World 1</title></head>
<body>
   <form id="form1" runat="server">
   <div><asp:label ID="messageLabel" runat="server">
</asp:label> </div>
   </form>
</body>
</html>
```

HTML Tags:

- The final elements of an ASP.NET page are the traditional **HTML tags** . A web programmer normally, cannot do without these elements. This is because HTML allows the display of the information in your ASP.NET controls and code in a way that is suitable for users.
- In the example that we saw earlier, that is the HelloWorld1.aspx, the source view of the aspx page shows usage of html tags.

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <title>Hello World 1</title>
   <script runat="server">
      void Page_Load()
      {
         messageLabel.Text = "Hello World";
      }
   </script>
</head>
<body>
   <form id="form1" runat="server">
```

HTML Tags (contd.):

```
<div>
   <asp:label ID="messageLabel" runat="server"> </asp:label>
   <% String title = "This is generated by code render block "; %>
   </div>
   </form>
</body>
</html>
```

Without the HTML element/tags, this page would have no format, and the browser would be unable to understand it.

1.17: Demo on Directives, Code Render Blocks, ASP.NET Server Controls

Demo

- Understanding Directives, Code Render Blocks, and ASP.NET Server Controls

1.18: View State

Usage of View State

- ASP.NET controls automatically retain their data when a page is sent to the server in response to an event
- This persistence of data is called as "view state"
- Maintaining the View State is the default setting for ASP.NET Web Forms

View State:

- Before we move on, let us take a look at other aspects of a typical ASP.NET application. Let us briefly understand the concept of a **view state**.
- When a form is submitted in classic ASP, all form values are cleared. Suppose you have submitted a form with a lot of information and the server comes back with an error. Then you will have to go back to the form and correct the information. You click the **Back** button, and all the form values are cleared, that is the form is as it was refreshed, and you will have to start all over again! The site did not maintain your ViewState.
- When a form is submitted in ASP.NET, the form reappears in the browser window together with all form values. Surprised!! This is because ASP.NET maintains your data. Microsoft calls this persistence of data as "**view state"**. The ViewState indicates the status of the page when submitted to the server.
- Earlier, the developers would have had to use various ways to remember the item a user had selected in a drop-down menu, or store the content entered into a text box. Typically, these ways would have relied on hidden form fields.
- This is no longer the case. Once the form values are submitted to the server for processing, the ASP.NET pages automatically retain all the information contained in text boxes and drop-down lists, as well as radio button and checkbox selections. They even keep track of dynamically generated tags, controls, and text.
- Maintaining the ViewState is the default setting for ASP.NET Web Forms. If you want to NOT maintain the ViewState, include the directive

<%@ Page EnableViewState="false" %> at the top of an .aspx page or add the attribute **EnableViewState="false"** to any control.

1.19: Demo on Understanding View State

Demo

- Understanding View State in ASP.NET

1.20: ASP.NET Page Life Cycle

Life Cycle Stages

- The aspx page goes through a series of procedures while loading. Developers can take advantage of these procedures to initiate actions at specific moments when the page is either getting created or destroyed
- The Life Cycle is as follows:
  - **Pre-Init:** The runtime setting of Themes and Master Pages is done at this stage.
  - **Init:** Page Controls are Initialized to default state.
  - **Load View State:** View State Data is loaded.

ASP.NET Page Life Cycle:

- Having taken a look at various elements in an ASP.NET page and the view state, let us now understand the Life Cycle of ASP.NET page.
- When an initial request for a page (a Web Form) is received by ASP.NET, it locates and loads the requested Web Form (and if necessary compiles the code). It is important to understand the sequence of events that occur when a Web Forms page is processed. The ASP.NET page goes through a series of procedures while loading and developers can take advantage of these procedures to initiate actions at specific moments when the page is either created or destroyed.
- The Life Cycle events of the page are as follows:
  1. **PreInit :** It is called at the beginning of page initialization stage. In this method, the personalization information is loaded and the page theme if applicable is initialized. This is also the preferred stage to dynamically define a **PageTheme** or **MasterPage** for the aspx page.
  2. **Init :** It performs the initialization and setup steps required to create a Page instance. In this stage, the server controls on the page are initialized to default state. However, the view state of the controls is not yet populated.
  3. **LoadViewState :** It restores ViewState information.

1.20: ASP.NET Page Life Cycle

Life Cycle Stages

- **LoadPostData:** The Information that is POSTed by the Browser is given or consumed here
- **InitComplete:** All the controls are completely initialized
- **PreLoad:** View state information and post back data for declared controls is loaded
- **Load:** Initialization Code is loaded at this stage
- **PreRender:** Any prerendering steps are carried out

ASP.NET Page Life Cycle:

- The Life Cycle events of the page are as follows (contd.):
  4. **LoadPostData :** The Information that is POSTed by the Browser is given or consumed here.
  5. **InitComplete :** It is called after the page initialization is complete. All the controls are completely initialized. The user can access server controls. However, they will not contain information returned by the user.
  6. **PreLoad :** It is called after all the **postback data** returned from the user is loaded. At this stage, the view state information and postback data for declared controls and controls created during initialization stage is loaded. Controls created in the preload method will also be loaded with the **view state** and **postback data**. (We will look at what is postback later in this lesson.)
  7. **Load :** It notifies the server control that should perform actions common to HTTP request for the page it is associated with, like setting up a database query. At this stage in the page lifecycle, the server controls in the hierarchy are created and initialized, view state is restored, and form controls will reflect client-side data.
  8. **PreRender:** It is called after the **LoadComplete** method. It notifies the server control to perform any necessary pre-rendering steps prior to saving view state and rendering content.

1.20: ASP.NET Page Life Cycle

Life Cycle Stages

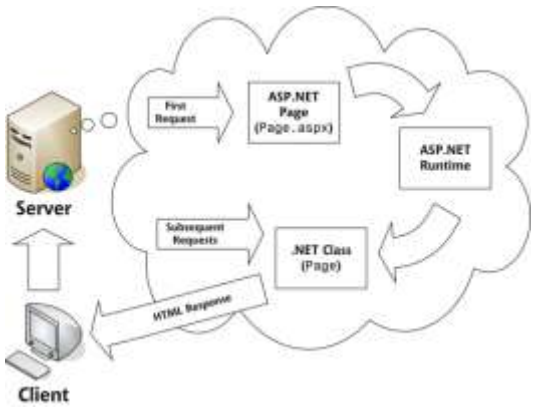
- **PreRenderComplete:** Page is ready to render
- **SaveViewState:** View State of all the controls is serialized
- **Render:** Render method creates the text and markup that is sent to the client browser at this stage
- **Unload:** A code sets in to close the resources and cleanup

ASP.NET Page Life Cycle:

- The Life Cycle events of the page are as follows (contd.):
  9. **PreRenderComplete:** All the controls are created and the page is ready to render the output at this stage. This is the last event called before the page's view state is saved.
  10. **SaveViewState :** It saves any server control view state changes that have occurred since the time the page was posted back to the server. If there is no view state associated with the control this method returns a null reference.
  11. **Render :** It initializes the HTMLTextWriter object and calls on the child controls of the page to render. The Render method creates the text and markup that is sent to the client browser.
  12. **Unload :** It is typically used for cleanup functions that precede disposition of the control.

1.21: Life Cycle Stages

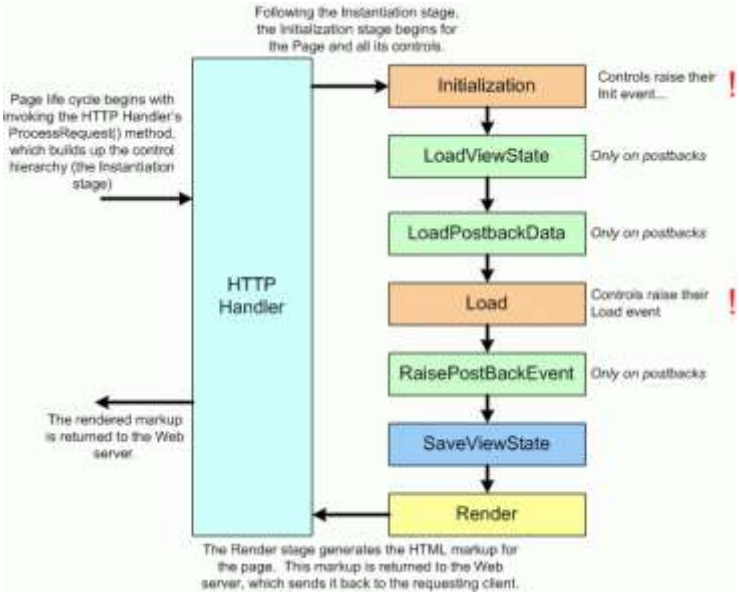
Life Cycle Stages



ASP.NET Page Life Cycle:
The picture on the slide depicts the cycle of how the requested page is provided by the web server on the first request and subsequent requests.

1.21: Events in ASP.NET Page Lifecycle

Diagrammatic Representation



Events in ASP.NET Page Life cycle:
This diagram in the above slide briefly illustrates the events that take place at the first request and which are called only for postback (Not all the events are listed in the above diagram).

1.22: Demo on Understanding Page Life Cycle

Demo

- Understanding ASP.NET Page Life Cycle

1.23: Post Back

Concept of Post Back

- Every ASP.NET page is a separate entity with ability to process its own posted data
- The values of the Form are posted to the same page and the very same page can process the data. This model is called "post back"

```
If (Page.IsPostBack == true)
{
//Do Processing
}
```

PostBack:

- When you are working with ASP.NET pages, be sure you understand the page events just listed. They are important because you place a lot of your page behavior. In classic ASP, developers had to use POST method in form to post the values of a Form to a second page. The second asp page would receive the data and process it for doing any validation or processing any processing on the server side.
- With ASP.NET, each ASP.NET page will be a separate entity with ability to process its own posted data. That is, the values of the Form are posted to the same page and the very same page can process the data. This model is called **post back**.
- This **Postback** is a read only property with each Asp.Net Page (System.Web.UI.Page) class. Each ASP.NET page when loaded goes through a regular creation and destruction process. We have just understood the various life cycle events of the page. This is false when the first time the page is loaded. It is true when the page is submitted and processed. This enables users to write the code depending on if the PostBack is true or false. The Page_Load subroutine runs every time the page is loaded. If you want to execute the code in the Page_Load subroutine only the first time the page is loaded, you can use the Page.IsPostBack property. The following code snippet shows the use of this property:

```
protected void Page_Load(object sender, EventArgs e)
 {
     if (this.IsPostBack == false)
     {
       ListBox1.Items.Add(new ListItem("Language", "C#"));
       ListBox1.Items.Add("VC++");
      }
}
```

1.24: Auto Post Back

Concept of Auto Post Back

- Auto Post Back is built into the form-based server controls and needs to be enabled
- It automatically posts the page back to the server whenever the value of the control in question is changed, when enabled

Note:

- Hence now we go back to the ViewState example that we have seen. We saw the problem of values getting added to **ListBox** every time we clicked the button. We now know the problem is because the State of the control is maintained as a result of the **ViewState** property. However, to resolve this problem for recursive calls we can check for the **IsPostback** property and based on that allow the code to execute.
- So now we know that a Postback is just posting back to the same page. The postback contains all the information collected on the initial page for processing if needed.

AutoPostBack:

- AutoPostback is built into the form based server controls and needs to be enabled. It is a feature which specifies whether the Web Form is automatically submitted when the content of the control is changed. Note that the control will "fire" a postback if **AutoPostback** is set to True, only when the user leaves the control, NOT while editing the content.

1.25: Demo: Understanding Post Back and Auto Post Back

Demo

- Understanding Post Back and Auto Post Back

1.26: Code Behind

Concept

- Code Behind provides separation between the user interface and the programming logic.
- With every ASP.NET (Web Form) page created, there are two distinct files:
  - .aspx
  - .aspx.cs

Code Behind:

- The **Code Behind class model** had been evolved to get rid of the problems of using **inline code** in Visual Studio. We have already seen about inline code – the one written in the source view of the ASP.NET page. The code behind model is based on the idea that each Web form page is bound to a separate class file and this file is the basis of the dynamically generated page class that the ASP.NET runtime creates for each .aspx resource. So the code behind model did exist in earlier versions of ASP.NET. However, the way it is handled in ASP.NET is quite different.
- The thought of using the **code behind model** is for separating the **user interface** and the **programming logic** into different files. The code-behind model entails that a separate class (.aspx.cs) file is used to hold all the code bound to the page. This class inherits from Page and constitutes the effective base class for the aforementioned dynamically generated class that represents the .aspx resource in execution. In other words, if you use **code inline**, then the page is represented by an object derived from Page. Else, it will be represented by an instance of a class derived from the **code-behind class**.
- The .aspx page using this code behind model has some attributes in the Page Directive. The **Codefile** attribute is meant to point to the code-behind page that is used in the presentation page. The next attribute is **Inherits**. It specifies the name of the class that is bound to page when the page is compiled.
- The code-behind page which is more often referred to as **code-beside model** is rather simple in appearance because of the partial class capabilities. The **code-beside model** provides for a modular approach by separating support code and layout.

1.27:Other Code Behind Files

Other Code Behind Files

- Code Behind provides separation between the user interface and the programming logic.
- With every ASP.NET (Web Form) page created, there are two distinct files:
  - .aspx
  - .aspx.cs
- Only the aspx page does not use code behind. The other code components are as follows:
  - Master Page:
    - .master
    - .master.cs
  - Web User Control:
    - .ascx
    - .ascx.cs

Other Code Behind Files:

- You can see that the class created in the code behind file uses partial-classes, employing the new Partial keyword. This enables you to simply place the methods that you need in your page class. In the following code snippet we have only placed the methods required:

```
public partial class Default2 : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
     }
    protected void Button1_Click(object sender, EventArgs e)
    {
        Label1.Text = ListBox1.SelectedValue + "   :   " +
ListBox1.SelectedItem.Text;
    }
    protected void ListBox1_SelectedIndexChanged(object
sender, EventArgs e)
    {
        Label1.Text = ListBox1.SelectedValue + "   :   " +
ListBox1.SelectedItem.Text;
    }
}
```

1.28: Configuration Files in ASP.NET

Types of Files

- ASP.NET configuration is stored in two primary XML-based files
  - Machine.config: Server or machine-wide configuration
  - Web.config: Application Configuration files

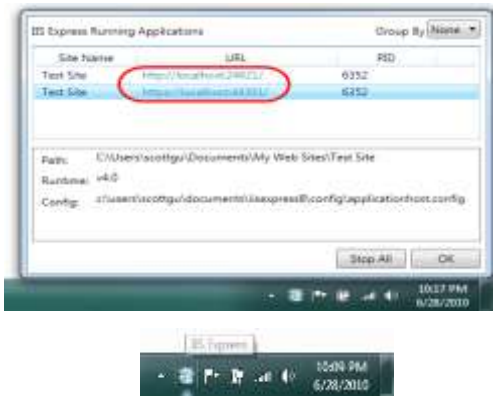
Configuration Files in ASP.NET:

- ASP.NET configuration is stored in two primary XML files. These files are used to describe the properties and behavior of various aspects of ASP.NET applications.
- The ASP.NET configuration system supports two kinds of configuration files:
  - **machine.config:** Server or machine-wide configuration files
  - **web.config:** Application configuration files
- The configuration files are based upon XML and hence the files are readable using any plain text editor. Unlike a binary metabase, the XML configuration files can be easily copied from one server to another. When some settings are changed in the configuration file, the changes are automatically detected by the ASP.NET application and applies it to the executing application. This indicated that configuration changes can be easily applied without having to restart the server. The changes are completely transparent to the end user. Also the ASP.NET configuration files are extensible.

1.29:IISEXPRESS

IISEXPRESS

- When you press F5 to run an ASP.NET project, Visual Studio can automatically launch IIS Express and use it to run/debug the application (no extra configuration required).
- Like the ASP.NET Web Server, IIS Express will show up in your task-bar tray when running
- You can right-click and click "exit" on the icon above to quickly shutdown IIS Express. You can also right-click and pull up a list of all sites running with it, as well as the directory location and .NET versions they are running under.



In Visual Studio 2012, IIS Express replaces the ASP.NET Development Server as the default web server for testing within Visual Studio. (In Visual Studio 2010 SP1, IIS Express was available as an option that you had to explicitly configure as the test web server.) IIS Express is a lightweight, self-contained version of IIS that has been optimized for developers. It has all the core capabilities of IIS as well as additional features designed to ease website development and includes the following:

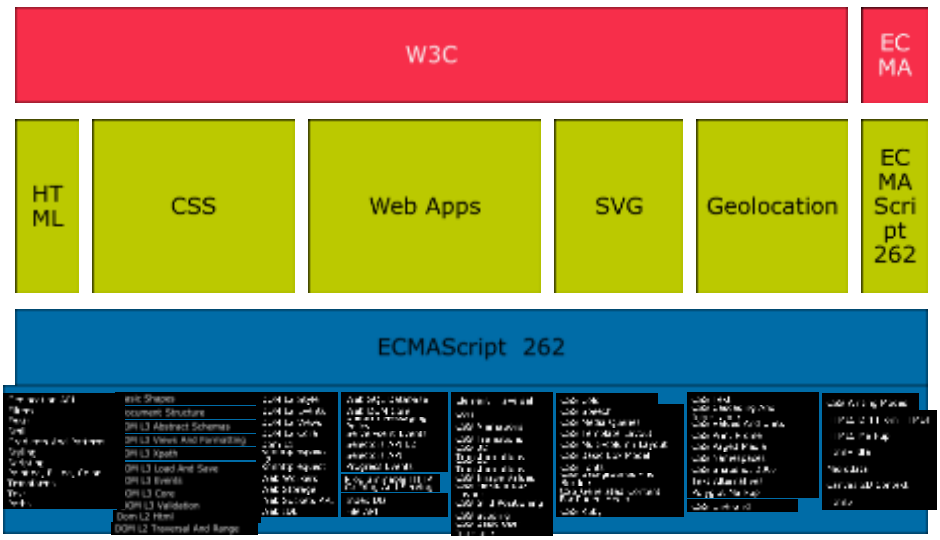
It does not run as a service or require administrator user rights in order to perform most tasks.

IIS Express works well with ASP.NET and PHP applications.

Multiple users of IIS Express can work independently on the same computer.

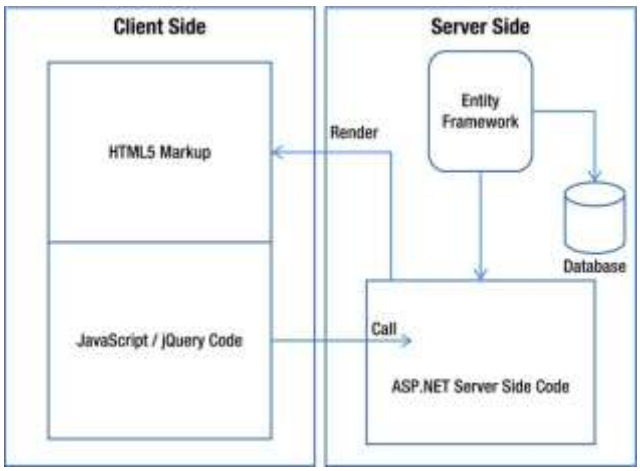
1.30:Map of HTML5

Map of HTML5



1.31:HTML 5- Architecture

HTML 5- Architecture



The ASP.NET server-side infrastructure sends HTML5 markup to the browser when a request is
received. The ASP.NET server-side infrastructure consists of code in a web form code file or MVC
controller. Most real-world web applications use data residing in a data store such as SQL Server. To access
this business data, you can use a combination of ADO.NET and Plain Old CLR Objects (POCOs) or Entity
Framework.
When the ASP.NET server-side infrastructure sends HTML5 markup to the client browser, the browser
renders the user interface and allows the end user to work with the page. Many HTML5 features
mentioned earlier expose programmable APIs that can be consumed using JavaScript code. The JavaScript
code can, in turn, talk with the server to retrieve data or configuration needed for processing. For example,
suppose you let the end user plot a simple bar graph in the browser using the HTML5 canvas API. After the
graph is finished, you may want the user to save its data and related information to a database. This
requires transfer of data from the client browser to the server. To facilitate this data transfer, you can use
jQuery and send an Ajax request to a piece of server-side code. The server-side code then saves the data
into a database.

1.32:New HTML5 Markup Elements

New HTML5 Markup Elements

<header>	used to identify the page or application's introduction or navigational aids
<hgroup>	used to group heading elements
<nav>	represents a section of a document that links to other documents or to parts within the document itself
<aside>	represents content that is tangentially related to the content that forms the main textual flow of a document
<footer>	represents the footer for the nearest ancestor sectioning content or sectioning root element
<article>	represents a section of content that forms an independent part of a document or site
<section>	used to designate generic sections of a document
<figure>	used to enclose a figure like an illustration, diagram, or photo
<figcaption>	used to identify a figure's caption
<video>	represents a video player displayed to allow user control over video playback and a viewport to view a video
<audio>	represents an audio player displayed to allow user control over audio playback

HTML 5 <video> Attributes
src – specifies the location to pull the source file
autoplay – if present starts playing as soon as it's ready
controls – if present displays controls
preload – if present loads source at page load
loop – if present loops back to the beginning of the video

Example:

```
<video poster="video.jpg">
<source src="video.mp4"/>
<source src="video.ogv"/>
<p>This is fallback content</p>
 </video>
```

- **Audio -** HTML5 is likely to put an end to audio plug-in such as Microsoft Windows Media player, Microsoft Silverlight ,Apple QuickTime and the famous Adobe Flash
- **MIME type's -** *audio/mpeg, is optional but its always better to provide*
- Only .mp3, .wav, and .ogg (vorbis) formats are supported till date
- If quick time player is not available, then safari won't support this tag
- Other properties like auto play, loop, preload area also available

1.33:Html5 Updates

Html5 Updates

- Some improvements have been made to Web Forms server controls to take advantage of new features of HTML5:
- The TextMode property of the TextBox control has been updated to support the new HTML5 input types like email, datetime, and so on.
- Validator controls now support validating HTML5 input elements.
- New HTML5 elements that have attributes that represent a URL now support runat="server". As a result, you can use ASP.NET conventions in URL paths, like the ~ operator to represent the application root
- (for example, <video runat="server" src="~/myVideo.wmv" />).

1.34:File Upload Control Supports Multiple File Uploads

File Upload Control Supports Multiple File Uploads

- The file upload control now supports the multiple file upload feature (for browsers that support this).
- To enable, simply set the AllowMultiple property to "true".
- For example,
- <asp:FileUpload ID="uploadFile" runat="server" AllowMultiple="true" />
- You can then iterate through the files uploaded with the PostedFiles property:

```
foreach (var file in uploadFile.PostedFiles)
{
        file.SaveAs("");
}
```

1:34:1:Demo

Demo

- File Upload Control

1:35:Page Inspectior

Page Inspector

- Page Inspector is a tool that renders a web page (HTML, Web Forms, ASP.NET MVC, or Web Pages) in the Visual Studio IDE and lets you examine both the source code and the resulting output.
- For ASP.NET pages, Page Inspector lets you determine which server-side code has produced the HTML markup that is rendered to the browser.

1:35:1: Lab Page Inspector

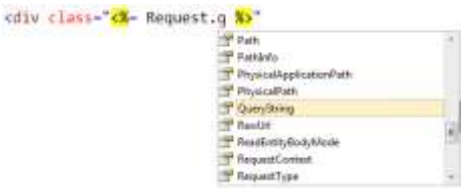
Lab

- PageInspector Walkthrough

1.36: HTML Editor Features

Html Editor Features

- Improved IntelliSense



- Tag completion when renaming tags

- Event handler generation in source view

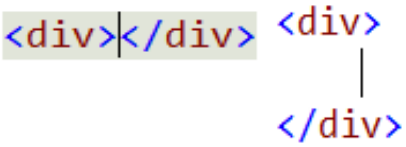
Quotation Tweak
Earlier we had to enter the quotation marks for runat="server". This is no longer necessary and VS2012 will automatically insert the quotation marks for you and move the cursor outside the quote sblock.

1:36:1:Html Editor Features

Html Editor Features

- Extract selected markup to a user control

- Smart indentation for empty elements



1:37:CSS Enhancements

Css Enhancements

- Hierarchical Indentation
- Color-selection
- Auto-reduce statement completion

1:37:1Hierarchical Indentation

Hierarchical Indentation

- The CSS editor uses indentation to display hierarchical rules, which gives you an overview of how the cascading rules are logically organized.
- In the following example, the #list a selector is a cascading child of list and is therefore indented.
- The indentation of a rule is determined by its parent rules. Hierarchical indentation is enabled by default, but you can disable it the Options dialog box (Tools, Options from the menu bar):

```
#list
{
    color: #f00;
}

    #list a
    {
        text-decoration: none;
    }
```

1:37:2: CSS Color selection color appears

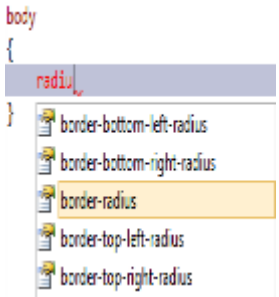
CSS Color Selection colors appears

- Visual Studio 2012 provides an integrated color-picker when a CSS rule related to a color is defined
- When you enter a color value, the color picker is displayed automatically and presents a list of previously used colors followed by a default color palette. You can select a color using the mouse or the keyboard.

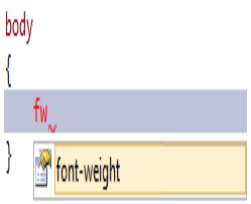
1:37:3Auto reduce Statement completion

Auto-reduce statement completion

- The IntelliSense list for CSS now filters based on the CSS properties and values supported by the selected schema.



- IntelliSense also supports title case searches:



1.38: ASP.NET: Best Practices & Guidelines

Standards

- Naming Conventions for UI Elements:
- Use Hungarian notation while naming UI controls. Use appropriate prefix for the UI elements to identify them from the rest of the variables
- Two different approaches are recommended:
- Use a common prefix (ui) for all UI elements. This will help to group all the UI elements together and easy to access all of them from the intel lisense
- Use appropriate prefix for each of the UI element

1.38: 1:ASP.NET: Best Practices & Guidelines

Standards

Control	Prefix
Label	lbl
TextBox	txt
DataGrid	dtg
Button	btn
ImageButton	imb
Hyperlink	hlk
DropDownList	ddl

ASP.NET Best Practices and Guidelines:

- Use appropriate prefix for each of the UI element. Since .NET has given several controls, a complete list of standard prefixes for each of the controls (including third party controls) need to be derived specific to the application. This approach enhances readability as the type of control is self-explanatory.
- A sample list is given above.

1.38: 1:ASP.NET: Best Practices & Guidelines

Standards

Control	Prefix
ListBox	lst
DataList	dtl
Repeater	rep
Checkbox	chk
CheckBoxList	cbl
RadioButton	rdo
RadioButtonList	rbl

1.38: 1:ASP.NET: Best Practices & Guidelines

Standards

Control	Prefix
Image	img
Panel	pnl
PlaceHolder	phd
Table	tbl
Validators	val

1.38: 4:ASP.NET: Best Practices & Guidelines

Standards

➢Reduce Round Trips:

- Implement Ajax UI (discussed in Lesson 10) whenever possible to avoid full page refresh. Only update the portion of the page that needs to be changed
- Use Page is Post back to avoid unnecessary processing on a round trip. Use this to ensure that the page initialization logic is performed when a page is loaded the first time and not in response to client post backs

1.38:5 :ASP.NET: Best Practices & Guidelines

Standards

- Page Redirection:
  - Consider using Server. Transfer instead of Response. Redirect
  - Use Server. Transfer for redirecting between pages in the same application which avoids unnecessary client-side redirection.
  - Response. Redirect sends a response header to client that causes client to send a new request to the redirected server by using the new URL
  - Server. Transfer avoids this level of indirection by simply making a server-side call

Summary

- In this lesson, you have learnt:
  - Basic ASP.NET concepts, ASP.NET Framework
  - Web Forms and their features
  - Life Cycle of a Web Form
  - Advantages of ASP.NET over ASP
  - Concept of Code Behind, Web Configuration File
  - Page Life Cycle

Review Question

- Question 1: IIS stands for:
  - Intermediate Internet Service
  - International Information System
  - Internet Information Services
  - None
- Question 2: runat="server" says
  - Script to be executed at server
  - Data must read from server
  - Open pages only in server
  - None
- Question 3: Which property of ASP.NET control can be set so that whenever control is operated, the web form is immediately submitted?
  - AutoSubmit
  - IsPostBack
  - AutoPostBack
  - AutoForm