ASP(Active Server Pages).NET

ASP.NET is a unified Web development model that includes the services necessary for you to build enterprise-class Web applications with a minimum of coding. ASP.NET is part of the .NET Framework, and when coding ASP.NET applications you have access to classes in the .NET Framework. You can code your applications in any language compatible with the common language runtime (CLR), including Microsoft Visual Basic and C#. These languages enable you to develop ASP.NET applications that benefit from the common language runtime, type safety, inheritance, and so on.

Microsoft® ASP.NET is a set of technologies in the Microsoft .NET Framework for building Web applications and XML Web services. ASP.NET pages execute on the server and generate markup such as HTML, WML, or XML that is sent to a desktop or mobile browser. ASP.NET pages use a compiled, event-driven programming model that improves performance and enables the separation of application logic and user interface. ASP.NET pages and ASP.NET XML Web services files contain server-side logic (as opposed to client-side logic) written in Microsoft® Visual Basic® .NET, Microsoft® Visual C#® .NET, or any Microsoft®.NET Framework-compatible language.

Microsoft® ASP.NET is the next generation technology for Web application development. It takes the best from Active Server Pages (ASP) as well as the rich services and features provided by the Common Language Runtime (CLR) and add many new features. The result is a robust, scalable, and fast Web development experience that will give you great flexibility with little coding.

The Three Flavors of ASP.NET: Web Forms, MVC, and Web Pages

ASP.NET offers three frameworks for creating web applications: ASP.NET Web Forms, ASP.NET MVC, and ASP.NET Web Pages. All three frameworks are stable and mature, and you can create great web applications with any of them.

Each framework targets a different audience or type of application. Which one you choose depends on a combination of your web development experience, what framework you're most comfortable with, and which is the best fit for the type of application you're creating. All three frameworks will be supported, updated, and improved in future releases of ASP.NET.
Here's an overview of each of the frameworks and some ideas for how to choose between them.

ASP.NET Web Forms (.aspx pages)

The Web Forms framework targets developers who prefer declarative and control-based programming, such as Microsoft Windows Forms (WinForms) and WPF/XAML/Silverlight. It offers a WYSIWYG designer-driven (drag-and-drop) development model, so it's popular with developers looking for a rapid application development (RAD) environment for web development. If you're new to web programming and are familiar with the traditional Microsoft RAD client development tools (for example, for Visual Basic and Visual C#), you can quickly build a web application without having expertise in HTML and JavaScript.
In particular, the Web Forms model provides the following features:

- An event model that exposes events which you can program like you would program a client application like WinForms or WPF.
- Server controls that render HTML for you and that you can customize by setting properties and styles.
- A rich assortment of controls for data access and data display.
- Automatic preservation of state (data) between HTTP requests, which makes it easy for a programmer who is accustomed to client applications to learn how to create applications for the stateless web.

Web Forms works well for small teams of Web developers and designers who want to take advantage of the large number of components available for rapid application development. In general, creating a Web Forms application requires less programming effort than creating the same application by using the ASP.NET MVC framework. The components (the Page class, controls, and so on) are tightly integrated and usually require less code than ASP.NET MVC applications. However, Web Forms is not just for rapid application development. There are many complex commercial apps and app frameworks built on top of Web Forms.

Because a Web Forms page and the controls on the page automatically generate much of the markup that's sent to the browser, you don't have the kind of fine-grained control over the HTML that the other ASP.NET models offer. An event-driven, control-focused model hides some of the behavior of HTML and HTTP. For example, it's not always possible to specify exactly what markup might be generated by a control.

The Web Forms model doesn't lend itself as readily as ASP.NET MVC to patterns-based development, separation of concerns, and automated unit testing. If you want to write code factored that way, you can; it's just not as automatic as it is in the ASP.NET MVC framework. The ASP.NET Web Forms MVP project shows an approach that facilitates separation of concerns and testability while maintaining the rapid development that Web Forms was built to deliver. **As an example of this in action, Microsoft SharePoint is built using Web Forms MVP.**

ASP.NET MVC

ASP.NET MVC targets developers who are interested in patterns and principles like test-driven development, separation of concerns, inversion of control (IoC), anddependency injection (DI). This framework encourages separating the business logic layer of a web application from its presentation layer.
By dividing the application into the model (M), views (V), and controllers (C), ASP.NET MVC can make it easier to manage complexity in larger applications. With ASP.NET MVC, you can have multiple teams working on a web site because the code for the business logic is separate from the code and markup for the presentation layer — developers can work on the business logic while designers work on the markup and JavaScript that is sent to the browser.
With ASP.NET MVC, you work more directly with HTML and HTTP than in Web Forms. Web Forms tends to hide some of that by mimicking the way you would program a WinForms or WPF application. For example, Web Forms can automatically preserve state between HTTP requests, but you have to code that explicitly in MVC. The MVC model enables you to take complete control over exactly what your application is doing and how it behaves in the web environment.
MVC was designed to be extensible, providing power developers the ability to customize the framework for their application needs. In addition, the ASP.NET MVC source code is available under an OSI license.

ASP.NET Web Pages (.cshtml and .vbhtml files)

ASP.NET Web Pages targets developers who want a simple web development story, along the lines of PHP. In the Web Pages model, you create HTML pages and then add server-based code to the page in order to dynamically control how that markup is rendered. Web Pages is specifically designed to be a lightweight framework, and it's the easiest entry point into ASP.NET for people who know HTML but might not have broad programming experience — for example, students or hobbyists. It's also a good way for web developers who know PHP or similar frameworks to start using ASP.NET.
Like Web Forms, Web Pages is oriented toward rapid development. Web Pages provides components called *helpers* that you can add to pages and that let you use just a few lines of code to perform tasks that would either be tedious or complex. For example, there are helpers to display database data, add a Twitter feed, log in using Facebook, add maps to a page, and so on.
Web Pages provides a simpler approach than Web Forms. If you look at a .cshtml or .vbhtml file, you can generally think of the logic as executing top-to-bottom in the file, as you would with PHP, SHTML, etc. And because .cshtml

and .vbhtml files are essentially HTML files that have additional ASP.NET code in them, they lend themselves easily to adding client-side functionality via JavaScript and jQuery.

Page and Controls Framework

The ASP.NET Web Forms page and controls framework is a programming framework that runs on a Web server to dynamically produce and render ASP.NET Web pages. ASP.NET Web pages can be requested from any browser or client device, and ASP.NET renders markup (such as HTML) to the requesting browser. As a rule, you can use the same page for multiple browsers, because ASP.NET renders the appropriate markup for the browser making the request. However, you can design your ASP.NET Web page to target a specific browser and take advantage of the features of that browser.

ASP.NET Web Forms pages are completely object-oriented. Within ASP.NET Web forms pages you can work with HTML elements using properties, methods, and events. The ASP.NET page framework removes the implementation details of the separation of client and server inherent in Web-based applications by presenting a unified model for responding to client events in code that runs at the server. The framework also automatically maintains the state of a page and the controls on that page during the page processing life cycle. For more information see ASP.NET Web Forms Pages Overview.

The ASP.NET page and controls framework also enables you to encapsulate common UI functionality in easy-to-use, reusable controls. Controls are written once, can be used in many pages, and are integrated into the ASP.NET Web page that they are placed in during rendering.

The ASP.NET page and controls framework also provides features to control the overall look and feel of your Web site via themes and skins. You can define themes and skins and then apply them at a page level or at a control level. For more information, see ASP.NET Themes and Skins.

In addition to themes, you can define master pages that you use to create a consistent layout for the pages in your application. A single master page defines the layout and standard behavior that you want for all the pages (or a group of pages) in your application. You can then create individual content pages that contain the page-specific content you want to display. When users request the content pages, they merge with the master page to produce output that combines the layout of the master page with the content from the content page. For more information see ASP.NET Master Pages.

The ASP.NET page framework also enables you to define the pattern for URLs that will be used in your site. This helps with search engine optimization (SEO) and makes URLs more user-friendly. For more information, see ASP.NET Routing.

The ASP.NET page and control framework is designed to generate HTML that conforms to accessibility guidelines. For more information, see Accessibility in Visual Studio and ASP.NET.

ASP.NET Compiler

All ASP.NET code is compiled, which enables strong typing, performance optimizations, and early binding, among other benefits. Once the code has been compiled, the common language runtime further compiles ASP.NET code to native code, providing improved performance.

ASP.NET includes a compiler that will compile all your application components including pages and controls into an assembly that the ASP.NET hosting environment can then use to service user requests. For more information, see ASP.NET Compilation Overview.

ASP.NET Compilation Overview

In order for application code to service requests by users, ASP.NET must first compile the code into one or

more assemblies. Assemblies are files that have the file name extension .dll. You can write ASP.NET code in many different languages, such as Visual Basic and C#. When the code is compiled, it is translated into a language-independent and CPU-independent representation called Microsoft Intermediate Language (MSIL). At run time, MSIL runs in the context of the .NET Framework, which translates MSIL into CPU-specific instructions for the processor on the computer running the application.

There are many benefits to compiling application code including:

- **Performance** Compiled code is much faster than scripting languages such as ECMAScript or VBScript because it is a closer representation to machine code and does not require additional parsing.
- **Security** Compiled code is more difficult to reverse engineer than non-compiled source code because it lacks the readability and abstraction of a high-level language. Additionally, there are obfuscation tools that make compiled code even more resistant to reverse engineering.
- **Stability** Code is checked at compile time for syntax errors, type safety, and other problems. By catching these errors at build-time you can eliminate many errors in your code.
- **Interoperability** Because MSIL code supports any .NET language, you can use assemblies that were originally written in other languages in your code. For example, if you are writing an ASP.NET Web page in C#, you can add a reference to a .dll file that was written in Visual Basic.

The ASP.NET compilation architecture includes a number of features including:

- Multiple language support.
- Automatic compilation.
- Flexible deployment.
- Extensible build system.

The following sections describe each of these features.

Multiple Language Support

In ASP.NET 2.0 you can use different languages such as Visual Basic and C# in the same application because ASP.NET will create multiple assemblies, one for each language. For code stored in the App_Code folder, you can specify a subfolder for each language. For more information on the App_Code folder, see Shared Code Folders in ASP.NET Web Site Projects.

Automatic Compilation

ASP.NET automatically compiles your application code and any dependent resources the first time a user requests a resource from the Web site. In general, ASP.NET creates an assembly for each application directory (such as App_Code) and one for the main directory. (If files in a directory are in different programming languages, then separate assemblies will be created for each language.) You can specify which directories are compiled into single assemblies in the Compilation section of the Web.config file.

Flexible Deployment

Because ASP.NET compiles your Web site on first user request, you can simply copy your application's source code to the production Web server. However, ASP.NET also provides precompilation options that allow you to compile your Web site before it has been deployed, or to compile it after it has been deployed but before a user requests it. Precompilation has several advantages. It can improve the performance of your Web site on first request because there will be no lag time while ASP.NET compiles the site. Precompiling can also help you find errors that might otherwise be found only when a user requests a page. Finally, if you precompile the Web site before you deploy it, you can deploy the assemblies instead of the source code.

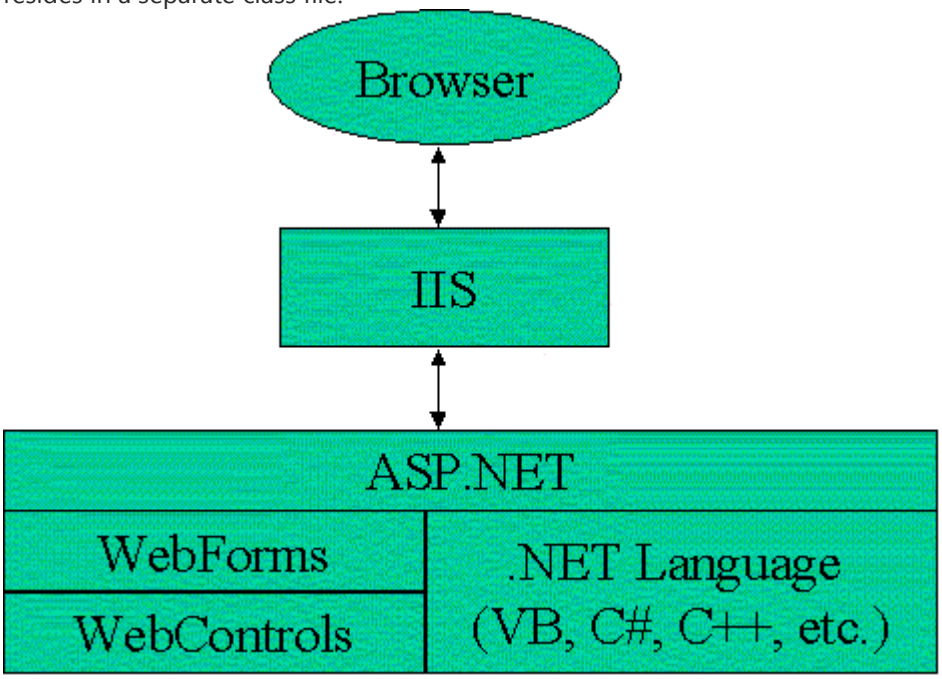
You can precompile a Web site using the ASP.NET compiler tool (ASPNET_Compiler.exe). The tool that provides the following precompilation options:

- **In-place compilation** This option performs the same compilation that occurs during dynamic compilation. Use this option to compile a Web site that has already been deployed to a production server.

- **Non-updateable full precompilation** Use this to compile an application and then copy the compiled output to the production server. All application code, markup, and UI code is compiled into assemblies. Placeholder files such as .aspx pages still exist so that you can perform file-specific tasks such as configure permissions, but the files contain no updateable code. In order to update any page or any code you must precompile the Web site again and deploy it again.
- **Updateable precompilation** This is similar to non-updateable full precompilation, except that UI elements such as .aspx pages and .ascx controls retain all their markup, UI code, and inline code, if any. You can update code in the file after it has been deployed; ASP.NET will detect changes to the file and recompile it. Note that code in a code-behind file (.vb or .cs file) built into assemblies during precompilation, and you therefore cannot change it without going through the precompilation and deployment steps again.

ASP.NET Web Forms Pages Overview

Web Forms are the heart and soul of ASP.NET. Web Forms are the User Interface (UI) elements that give your Web applications their look and feel. Web Forms are similar to Windows Forms in that they provide properties, methods, and events for the controls that are placed onto them. However, these UI elements render themselves in the appropriate markup language required by the request, e.g. HTML. If you use Microsoft Visual Studio® .NET, you will also get the familiar drag-and-drop interface used to create your UI for your Web application.
Web Forms are made up of two components: the visual portion (the ASPX file), and the code behind the form, which resides in a separate class file.



You use ASP.NET Web Forms pages as the programmable user interface for your Web Forms application. An ASP.NET Web Forms page presents information to the user in any browser or client device and implements application logic using server-side code. ASP.NET Web Forms pages are:

•Based on Microsoft ASP.NET technology, in which code that runs on the server dynamically generates Web page output to the browser or client device.

•Compatible with any browser or mobile device. An ASP.NET Web page automatically renders the correct browser-compliant HTML for features such as styles, layout, and so on.
•Compatible with any language supported by the .NET common language runtime, such as Microsoft Visual Basic and Microsoft Visual C#.
•Built on the Microsoft .NET Framework. This provides all the benefits of the framework, including a managed environment, type safety, and inheritance.
•Flexible because you can add user-created and third party controls to them.

Note

ASP.NET offers several technologies for creating Web applications. Web Forms pages (described here) offer a familiar drag-and-drop, event-driven model that includes hundreds of controls for rapid development. ASP.NET Web Pages provide a simple way to connect to data and add dynamic code into HTML using a lightweight syntax. ASP.NET MVC (Model View Controller) is a patterns-based development model that enables a separation of concerns and that emphasizes agile, test-driven development. We encourage you to review the features of each of the technologies and to select that on that best fits with your preferences and goals.

The Purpose of Web Forms

Web Forms and ASP.NET were created to overcome some of the limitations of ASP. These new strengths include:

- Separation of HTML interface from application logic
- A rich set of server-side controls that can detect the browser and send out appropriate markup language such as HTML
- Less code to write due to the data binding capabilities of the new server-side .NET controls
- Event-based programming model that is familiar to Microsoft Visual Basic® programmers
- Compiled code and support for multiple languages, as opposed to ASP which was interpreted as Microsoft Visual Basic Scripting (VBScript) or Microsoft Jscript®
- Allows third parties to create controls that provide additional functionality

On the surface, Web Forms seem just like a workspace where you draw controls. In reality, they can do a whole lot more. But normally you will just place any of the various controls onto the Web Form to create your UI. The controls you use determine which properties, events, and methods you will get for each control. There are two types of controls that you can use to create your user interface: HTML controls and Web Form controls.
Let's look at the different types of controls that you can use in Web Forms and the ASP.NET Framework.

Components of ASP.NET Web Forms Pages

In ASP.NET Web Forms pages, user interface programming is divided into two pieces: the visual component and the logic. If you have worked with tools like Visual Basic and Visual C++ in the past, you will recognize this division between the visible portion of a page and the code that interacts with it.

The visual element consists of a file containing static markup such as HTML or ASP.NET server controls or both. The ASP.NET Web Forms page works as a container for the static text and controls you want to display.

The logic for the Web Forms page consists of code that you create to interact with the page. The code can reside either in a **script** block in the page or in a separate class. If the code is in a separate class file, this file is referred to as the *code-behind* file. The code in the code-behind file can be written in Visual Basic, C#, or any other .NET Framework language. For more information about how ASP.NET Web Forms pages are constructed, see ASP.NET Web Forms Page Code Model.

For ASP.NET Web Forms site projects, you deploy page source code to a Web server and the pages are compiled automatically the first time a user browses to any page in the site. (Optionally, you can also precompile the site so that there is no compilation delay the first time a user browses a page.) For ASP.NET Web application projects, you must compile the Web Forms pages before deployment and deploy one or more assemblies.

What ASP.NET Web Forms Pages Help You Accomplish

Web application programming presents challenges that do not typically arise when programming traditional client-based applications. Among the challenges are:

- **Implementing a rich Web user interface** It can be difficult and tedious to design and implement a user interface using basic HTML facilities, especially if the page has a complex layout, a large amount of dynamic content, and full-featured user-interactive objects.
- **Separation of client and server** In a Web application, the client (browser) and server are different programs often running on different computers (and even on different operating systems). Consequently, the two halves of the application share very little information; they can communicate, but typically exchange only small chunks of simple information.
- **Stateless execution** When a Web server receives a request for a page, it finds the page, processes it, sends it to the browser, and then discards all page information. If the user requests the same page again, the server repeats the entire sequence, reprocessing the page from scratch. Put another way, a server has no memory of pages that it has processed—page are stateless. Therefore, if an application needs to maintain information about a page, its stateless nature can become a problem.
- **Unknown client capabilities** In many cases, Web applications are accessible to many users using different browsers. Browsers have different capabilities, making it difficult to create an application that will run equally well on all of them.
- **Complications with data access** Reading from and writing to a data source in traditional Web applications can be complicated and resource-intensive.
- **Complications with scalability** In many cases Web applications designed with existing methods fail to meet scalability goals due to the lack of compatibility between the various components of the application. This is often a common failure point for applications under a heavy growth cycle.

Meeting these challenges for Web applications can require substantial time and effort. ASP.NET Web Forms pages and the ASP.NET page framework address these challenges in the following ways:

- **Intuitive, consistent object model** The ASP.NET Web Forms page framework presents an object model that enables you to think of your forms as a unit, not as separate client and server pieces. In this model, you can program the page in a more intuitive way than in other models, including the ability to set properties for page elements and respond to events. In addition, ASP.NET Web Forms server controls are an abstraction from the physical contents of an HTML page and from the direct interaction between browser and server. In general, you can use server controls the way you might work with controls in a client application and not have to think about how to create the HTML to present and process the controls and their contents.
- **Event-driven programming model** ASP.NET Web Forms pages bring to Web applications the familiar model of writing event handlers for events that occur on either the client or server. The ASP.NET Web Forms page framework abstracts this model in such a way that the underlying mechanism of capturing an event on the client, transmitting it to the server, and calling the appropriate method is all automatic and invisible to you. The result is a clear, easily written code structure that supports event-driven development.
- **Intuitive state management** The ASP.NET Web Forms page framework automatically handles the task of maintaining the state of your page and its controls, and it provides you with explicit ways to maintain the state of application-specific information. This is accomplished without heavy use of server resources and can be implemented with or without sending cookies to the browser.
- **Browser-independent applications** The ASP.NET Web Forms page framework enables you to create all application logic on the server, eliminating the need to explicitly code for differences in browsers. However, it still enables you to take advantage of browser-specific features by writing client-side code to provide improved performance and a richer client experience.
- **.NET Framework common language runtime support** The ASP.NET Web Forms page framework is built on the .NET Framework, so the entire framework is available to any ASP.NET application. Your applications can be written in any language that is compatible that is with the runtime. In addition, data access is simplified using the data access infrastructure provided by the .NET Framework, including ADO.NET.
- **.NET Framework scalable server performance** The ASP.NET page framework enables you to scale your Web application from one computer with a single processor to a multi-computer Web farm cleanly and without complicated changes to the application's logic.

ASP.NET Web Site Layout

You can keep your Web site's files in any folder structure that is convenient for your application. To make it easier to work with your application, ASP.NET reserves certain file and folder names that you can use for specific types of content.

Default Pages

You can establish default pages for your application, which can make it simpler for users to navigate to your site. The default page is the page that is served when users navigate to your site without specifying a particular page. For example, you can create a page named Default.aspx and keep it in your site's root folder. When users navigate to

your site without specifying a particular page (for example, http://www.contoso.com/) you can configure your application so that the Default.aspx page is requested automatically. You can use a default page as the home page for your site, or you can write code in the page to redirect users to other pages.

Note:

In Internet Information Services (IIS), default pages are established as properties of your Web site.

Application Folders

ASP.NET recognizes certain folder names that you can use for specific types of content. The table below lists the reserved folder names and the type of files that the folders typically contain.

Note:

The content of application folders, except for the App_Themes folder, is not served in response to Web requests, but it can be accessed from application code.

Folder	Description
App_Browsers	Contains browser definitions (.browser files) that ASP.NET uses to identify individual browsers and determine their capabilities. For more information, see Browser Definition File Schema (browsers Element) and How to: Detect Browser Types in ASP.NET Web Pages.
App_Code	Contains source code for utility classes and business objects (for example, .cs, .vb, and .jsl files) that you want to compile as part of your application. In a dynamically compiled application, ASP.NET compiles the code in the App_Code folder on the initial request to your application. Items in this folder are then recompiled when any changes are detected. **Note:** Arbitrary file types can be placed in the App_Code folder to create strongly typed objects. For example, placing Web service files (.wsdl and .xsd files) in the App_Code folder creates strongly typed proxies. Code in the App_Code folder is referenced automatically in your application. In addition, the App_Code folder can contain subdirectories of files that need to be compiled at run time. For more information, see Shared Code Folders in ASP.NET Web Sites and codeSubDirectories Element for compilation (ASP.NET Settings Schema).
App_Data	Contains application data files including MDF files, XML files, as well as other data store files. The App_Data folder is used by ASP.NET 2.0 to store an application's local database, which can be used for maintaining membership and

	role information. For more information, see Introduction to Membership and Understanding Role Management.
App_GlobalResources	Contains resources (.resx and .resources files) that are compiled into assemblies with global scope. Resources in the App_GlobalResources folder are strongly typed and can be accessed programmatically. For more information, see ASP.NET Web Page Resources Overview.
App_LocalResources	Contains resources (.resx and .resources files) that are associated with a specific page, user control, or master page in an application For more information, see ASP.NET Web Page Resources Overview.
App_Themes	Contains a collection of files (.skin and .css files, as well as image files and generic resources) that define the appearance of ASP.NET Web pages and controls. For more information, see ASP.NET Themes and Skins Overview.
App_WebReferences	Contains reference contract files (.wsdl files), schemas (.xsd files), and discovery document files (.disco and .discomap files) defining a Web reference for use in an application. For more information about generating code for XML Web services, see Web Services Description Language Tool (Wsdl.exe).
Bin	Contains compiled assemblies (.dll files) for controls, components, or other code that you want to reference in your application. Any classes represented by code in the Bin folder are automatically referenced in your application. For more information, see Shared Code Folders in ASP.NET Web Sites.

ASP.NET Page Life Cycle Overview

When an ASP.NET page runs, the page goes through a life cycle in which it performs a series of processing steps. These include initialization, instantiating controls, restoring and maintaining state, running event handler code, and rendering. It is important for you to understand the page life cycle so that you can write code at the appropriate life-cycle stage for the effect you intend.

If you develop custom controls, you must be familiar with the page life cycle in order to correctly initialize controls, populate control properties with view-state data, and run control behavior code. The life cycle of a control is based on the page life cycle, and the page raises many of the events that you need to handle in a custom control.

In general terms, the page goes through the stages outlined in the following table. In addition to the page life-cycle stages, there are application stages that occur before and after a request but are not specific to a page. For more information, see Introduction to the ASP.NET Application Life Cycle and ASP.NET Application Life Cycle Overview for IIS 7.0.

Some parts of the life cycle occur only when a page is processed as a postback. For postbacks, the page life cycle is the same during a partial-page postback (as when you use an UpdatePanel control) as it is during a full-page postback.

Stage	Description
Page request	The page request occurs before the page life cycle begins. When the page is requested

	by a user, ASP.NET determines whether the page needs to be parsed and compiled (therefore beginning the life of a page), or whether a cached version of the page can be sent in response without running the page.
Start	In the start stage, page properties such as Request and Response are set. At this stage, the page also determines whether the request is a postback or a new request and sets the IsPostBack property. The page also sets the UICulture property.
Initialization	During page initialization, controls on the page are available and each control's UniqueID property is set. A master page and themes are also applied to the page if applicable. If the current request is a postback, the postback data has not yet been loaded and control property values have not been restored to the values from view state.
Load	During load, if the current request is a postback, control properties are loaded with information recovered from view state and control state.
Postback event handling	If the request is a postback, control event handlers are called. After that, the Validate method of all validator controls is called, which sets theIsValid property of individual validator controls and of the page. (There is an exception to this sequence: the handler for the event that caused validation is called after validation.)
Rendering	Before rendering, view state is saved for the page and all controls. During the rendering stage, the page calls the Render method for each control, providing a text writer that writes its output to the OutputStream object of the page's Response property.
Unload	The Unload event is raised after the page has been fully rendered, sent to the client, and is ready to be discarded. At this point, page properties such as Response and Request are unloaded and cleanup is performed.

Life-Cycle Events

Within each stage of the life cycle of a page, the page raises events that you can handle to run your own code. For control events, you bind the event handler to the event, either declaratively using attributes such as **onclick**, or in code.

Pages also support automatic event wire-up, meaning that ASP.NET looks for methods with particular names and automatically runs those methods when certain events are raised. If the **AutoEventWireup** attribute of the @ Page directive is set to **true**, page events are automatically bound to methods that use the naming convention of **Page**_*event*, such as **Page_Load** and **Page_Init**. For more information on automatic event wire-up, see ASP.NET Web Server Control Event Model.

The following table lists the page life-cycle events that you will use most frequently. There are more events than those listed; however, they are not used for most page-processing scenarios. Instead, they are primarily used by server controls on the ASP.NET Web page to initialize and render themselves. If you want to write custom ASP.NET server controls, you need to understand more about these events. For information about creating custom controls, see Developing Custom ASP.NET Server Controls.

Page Event	Typical Use
PreInit	Raised after the start stage is complete and before the initialization stage begins. Use this event for the following:

	• Check the IsPostBack property to determine whether this is the first time the page is being processed. The IsCallback andIsCrossPagePostBack properties have also been set at this time. • Create or re-create dynamic controls. • Set a master page dynamically. • Set the Theme property dynamically. • Read or set profile property values. **Note:** If the request is a postback, the values of the controls have not yet been restored from view state. If you set a control property at this stage, its value might be overwritten in the next event.
Init	Raised after all controls have been initialized and any skin settings have been applied. The Init event of individual controls occurs before the Init event of the page. Use this event to read or initialize control properties.
InitComplete	Raised at the end of the page's initialization stage. Only one operation takes place between the Init and InitComplete events: tracking of view state changes is turned on. View state tracking enables controls to persist any values that are programmatically added to theViewState collection. Until view state tracking is turned on, any values added to view state are lost across postbacks. Controls typically turn on view state tracking immediately after they raise their Init event. Use this event to make changes to view state that you want to make sure are persisted after the next postback.
PreLoad	Raised after the page loads view state for itself and all controls, and after it processes postback data that is included with the Requestinstance.
Load	The Page object calls the OnLoad method on the Page object, and then recursively does the same for each child control until the page and all controls are loaded. The Load event of individual controls occurs after the Load event of the page. Use the OnLoad event method to set properties in controls and to establish database connections.
Control events	Use these events to handle specific control events, such as a Button control's Click event or a TextBox control's TextChanged event. **Note:** In a postback request, if the page contains validator controls, check the IsValid property of the Page and of individual validation controls before performing any processing.
LoadComplete	Raised at the end of the event-handling stage. Use this event for tasks that require that all other controls on the page be loaded.
PreRender	Raised after the Page object has created all controls that are required in order to render the page, including child controls of composite controls. (To do this, the Page object calls EnsureChildControls for each control and for the page.) The Page object raises the PreRender event on the Page object, and then recursively does the same for each child control. The PreRenderevent of individual controls occurs after the PreRender event of the page.

	Use the event to make final changes to the contents of the page or its controls before the rendering stage begins.
PreRenderComplete	Raised after each data bound control whose DataSourceID property is set calls its DataBind method. For more information, see Data Binding Events for Data-Bound Controls later in this topic.
SaveStateComplete	Raised after view state and control state have been saved for the page and for all controls. Any changes to the page or controls at this point affect rendering, but the changes will not be retrieved on the next postback.
Render	This is not an event; instead, at this stage of processing, the Page object calls this method on each control. All ASP.NET Web server controls have a Render method that writes out the control's markup to send to the browser. If you create a custom control, you typically override this method to output the control's markup. However, if your custom control incorporates only standard ASP.NET Web server controls and no custom markup, you do not need to override the Render method. For more information, see Developing Custom ASP.NET Server Controls. A user control (an .ascx file) automatically incorporates rendering, so you do not need to explicitly render the control in code.
Unload	Raised for each control and then for the page. In controls, use this event to do final cleanup for specific controls, such as closing control-specific database connections. For the page itself, use this event to do final cleanup work, such as closing open files and database connections, or finishing up logging or other request-specific tasks. **Note:** During the unload stage, the page and its controls have been rendered, so you cannot make further changes to the response stream. If you attempt to call a method such as the **Response.Write** method, the page will throw an exception.

Login Control Events

The Login control can use settings in the Web.config file to manage membership authentication automatically. However, if your application requires you to customize how the control works, or if you want to understand how Login control events relate to the page life cycle, you can use the events listed in the following table.

Control Event	Typical Use
LoggingIn	Raised during a postback, after the page's LoadComplete event has occurred. This event marks the beginning of the login process. Use this event for tasks that must occur prior to beginning the authentication process.
Authenticate	Raised after the LoggingIn event. Use this event to override or enhance the default authentication behavior of a Login control.
LoggedIn	Raised after the user name and password have been authenticated. Use this event to redirect to another page or to dynamically set the text in the control.

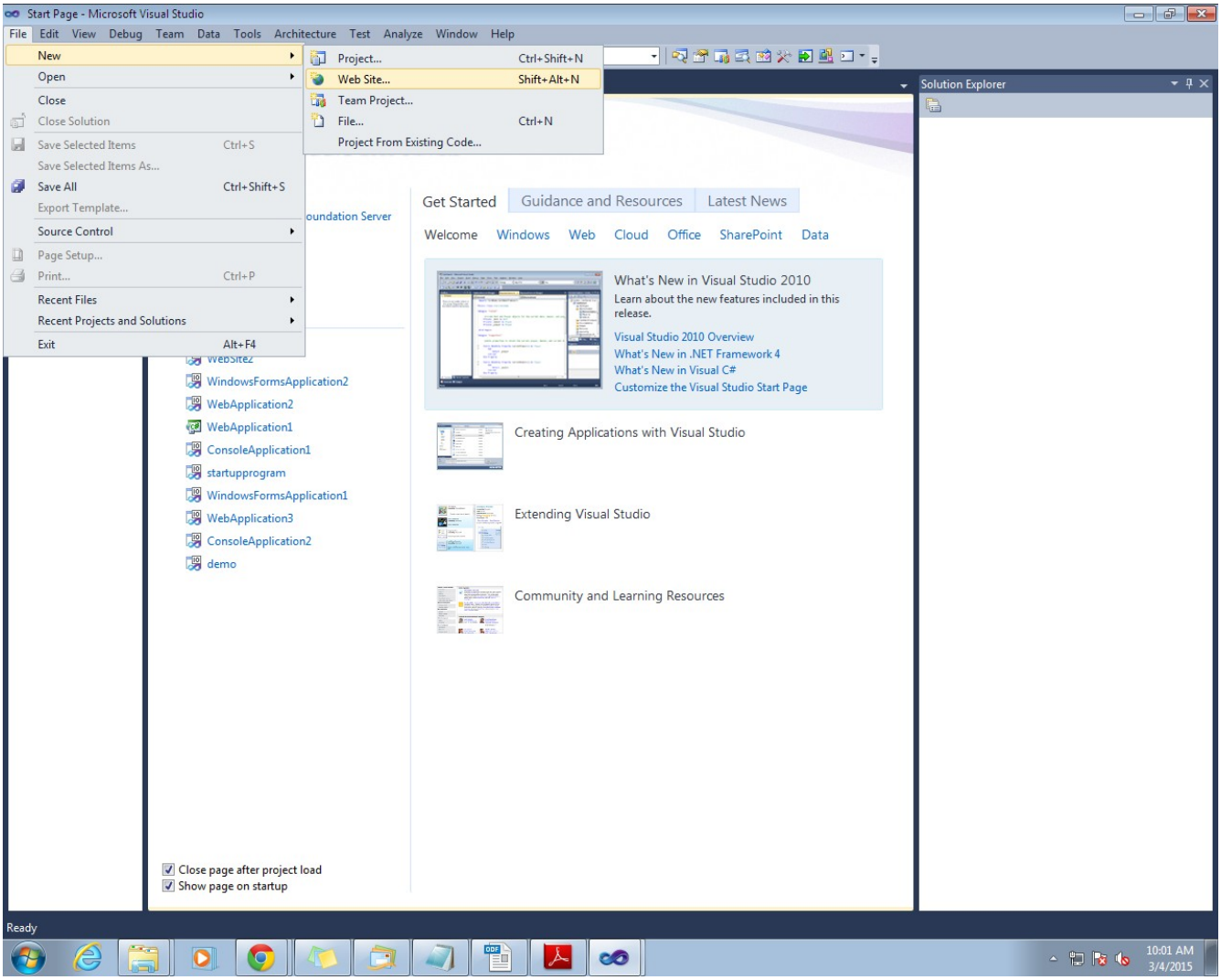
	This event does not occur if there is an error or if authentication fails.
LoginError	Raised if authentication was not successful. Use this event to set text in the control that explains the problem or to direct the user to a different page.

Creating Empty Web Site

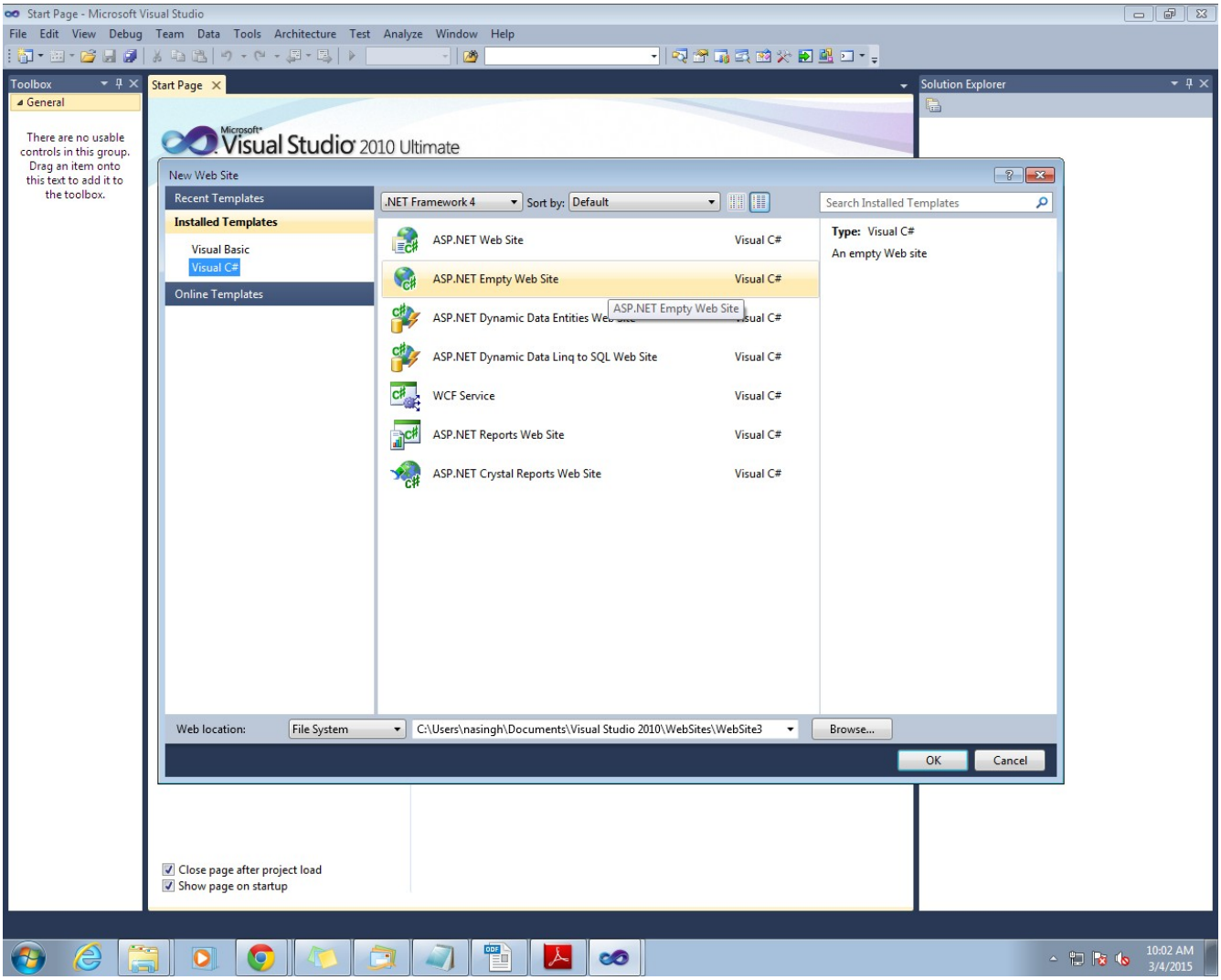
Now we will create empty website which will include only web.config file only .

To open empty website follow the below steps:

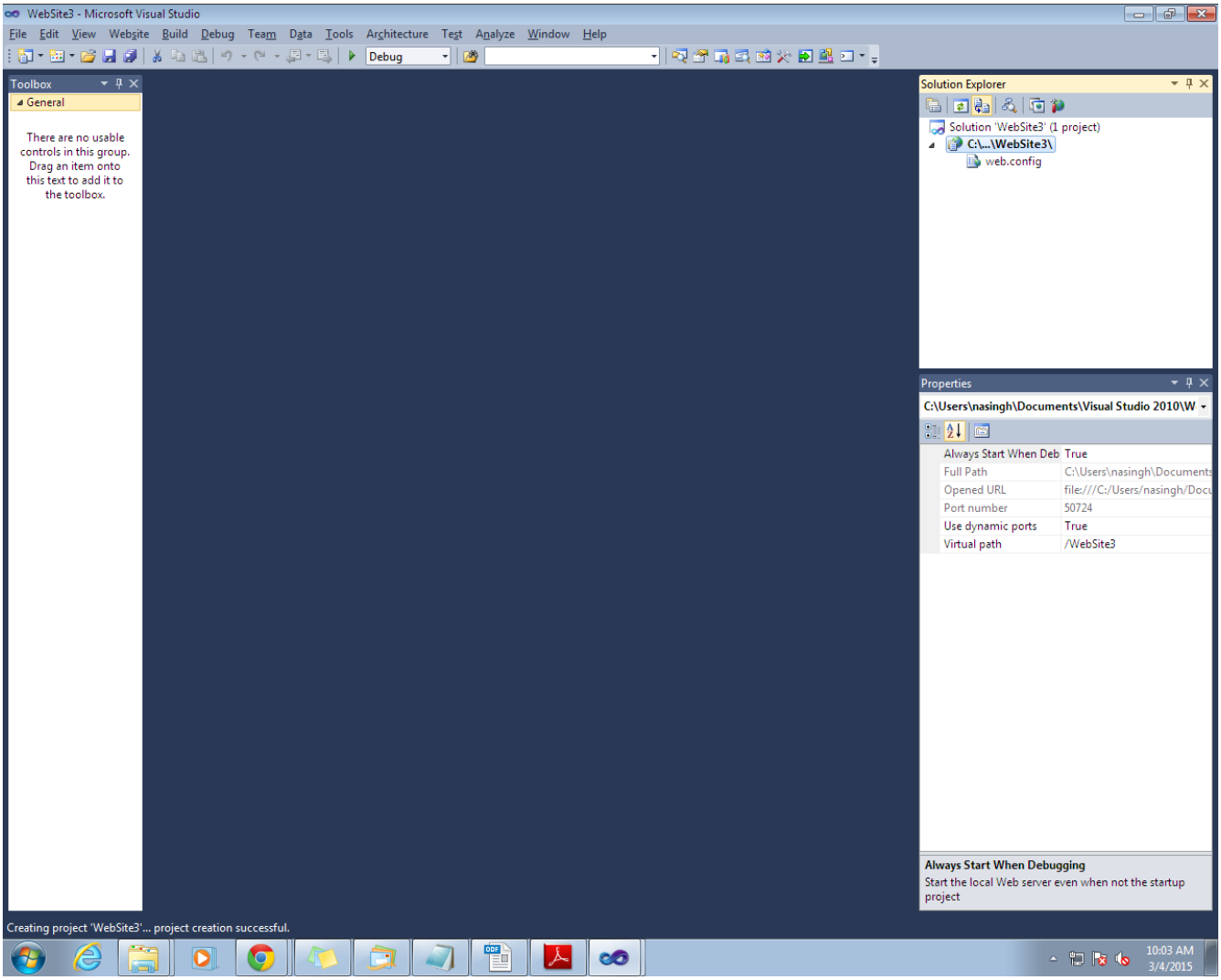
Open the visual studio and click on File tab as mentioned below:



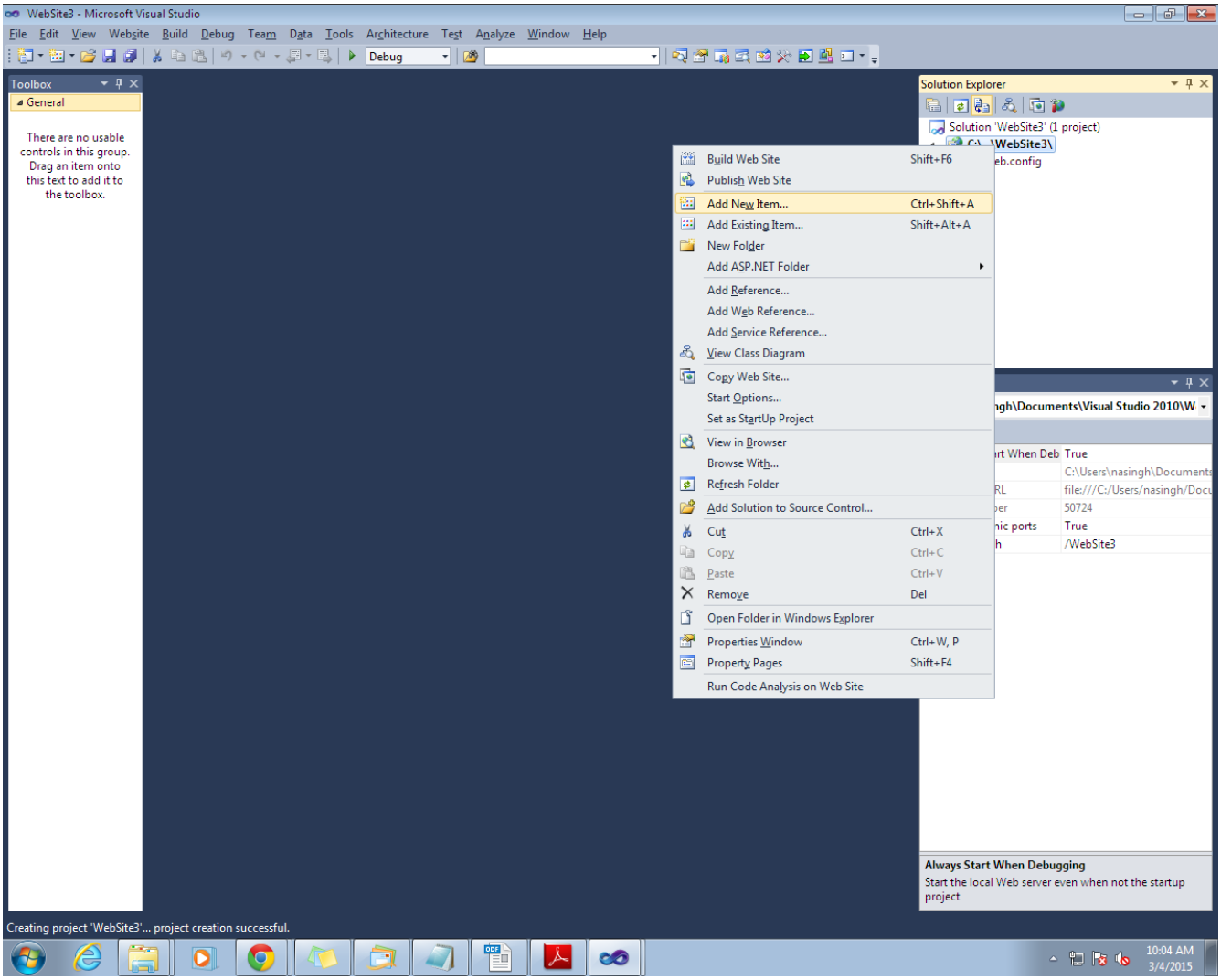
then select empty web site as shown below :



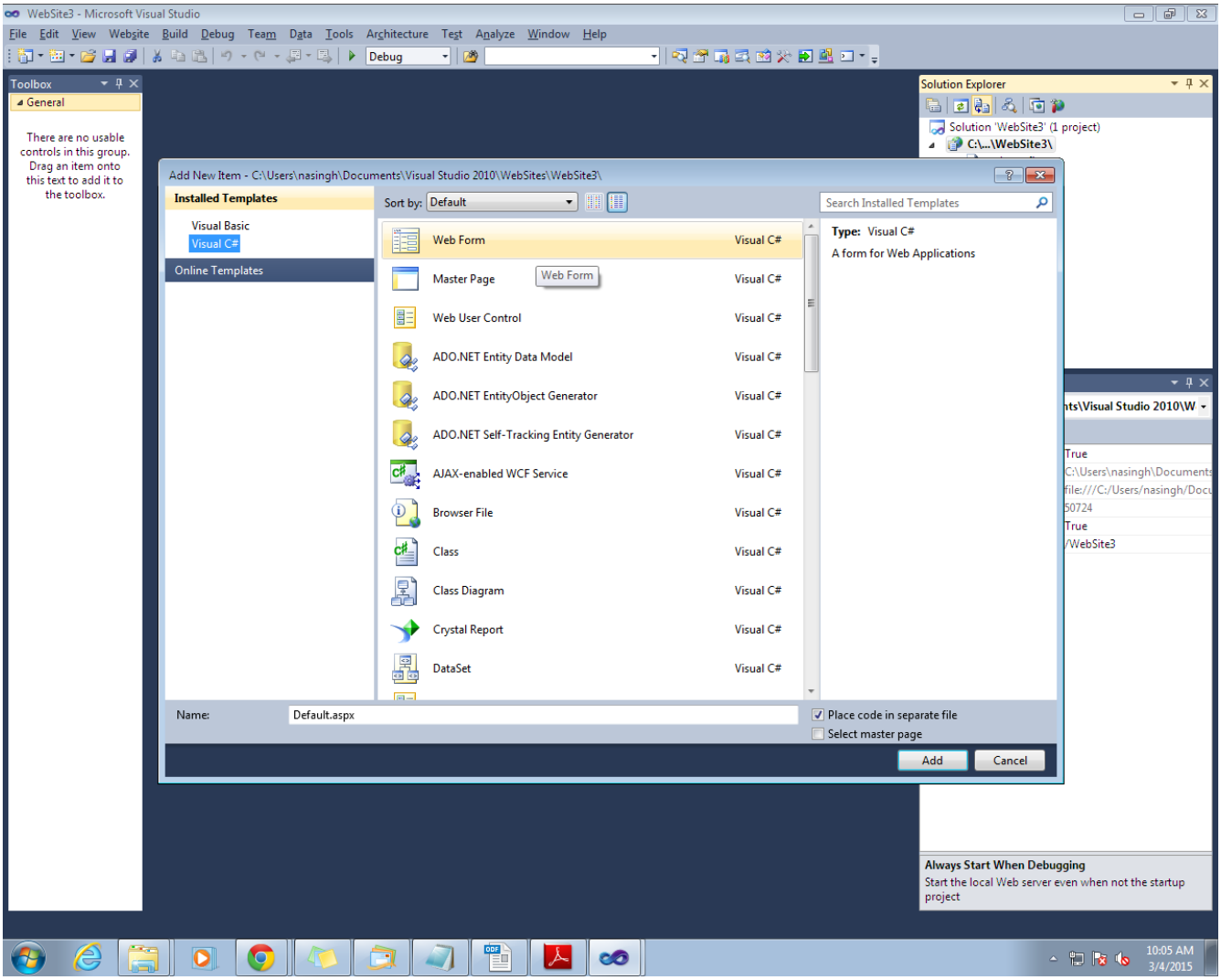
After selecting the empty web site ,. you will get the empty web site with only web.config file as shown at right in solution explorer.



Now add any webform into the website by right clicking the root directory of website like :



select the web-form which will be given name default.aspx .



After selecting webform , you will get the Default.aspx page on UI. Kinldy assure that you have checked the checkbox at right bottom with title **"Place code in separate file "** to create a separate code behind file for webform like Default.aspx.cs other wise you will single code file including html and c# script in same file ..

Activity 1.1 :

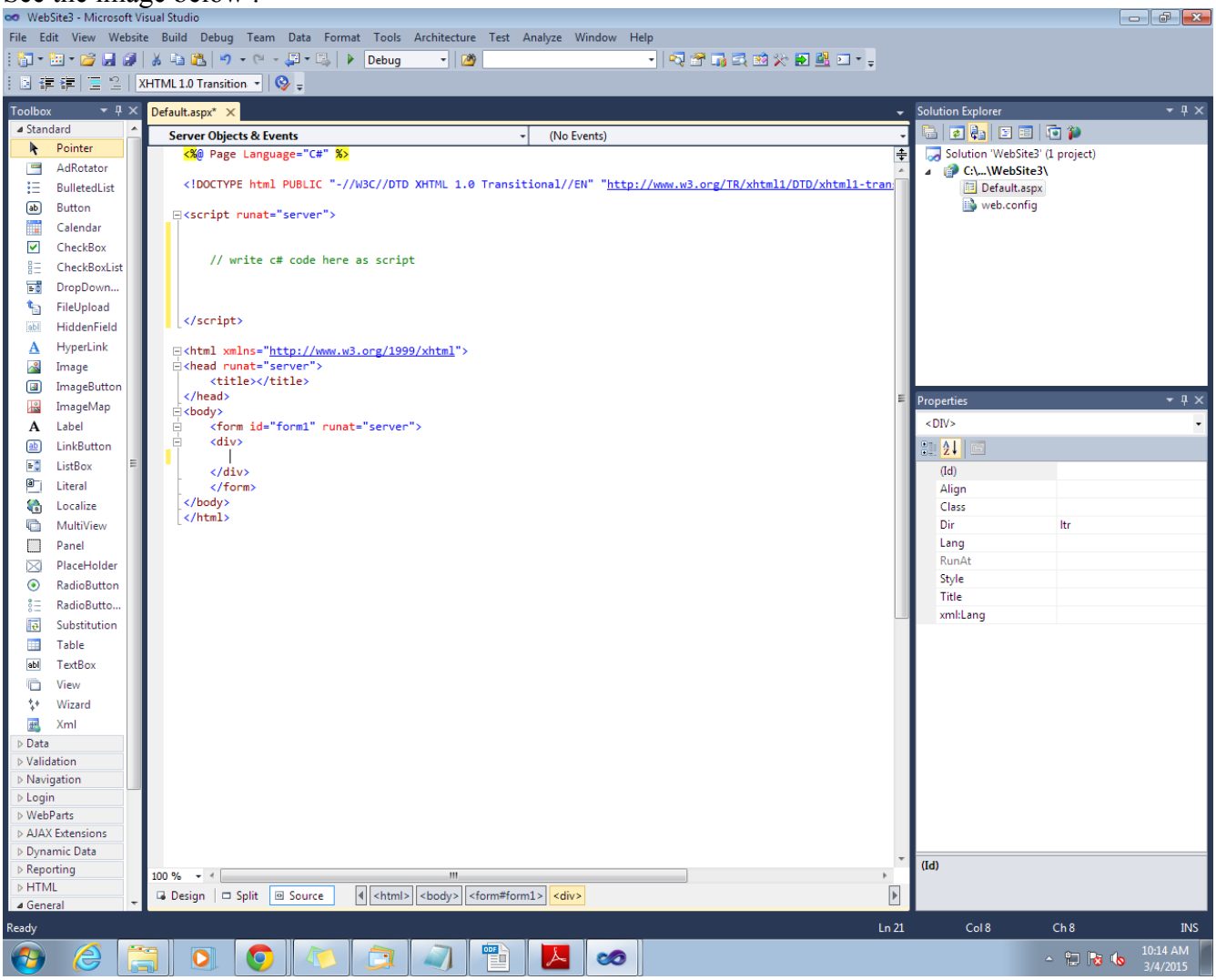
Creating a single file for aspx including html and script in same file ..

1.Right click the Root directory of website and select the Add New Item and then select the Web Form as shown above ..

While choosing the web form , do not check the "Place Code in Separate File" checkbox

after Clicking the ADD button you will get single file called Default.aspx page only including code and UI in both file .

See the image below :



You will get one <script> tag at top which will have all c# code required for asp page .

If we see the code of default.aspx page , we have Page declaration at the top E.G.

```
<%@ Page Language="C#" %>
```

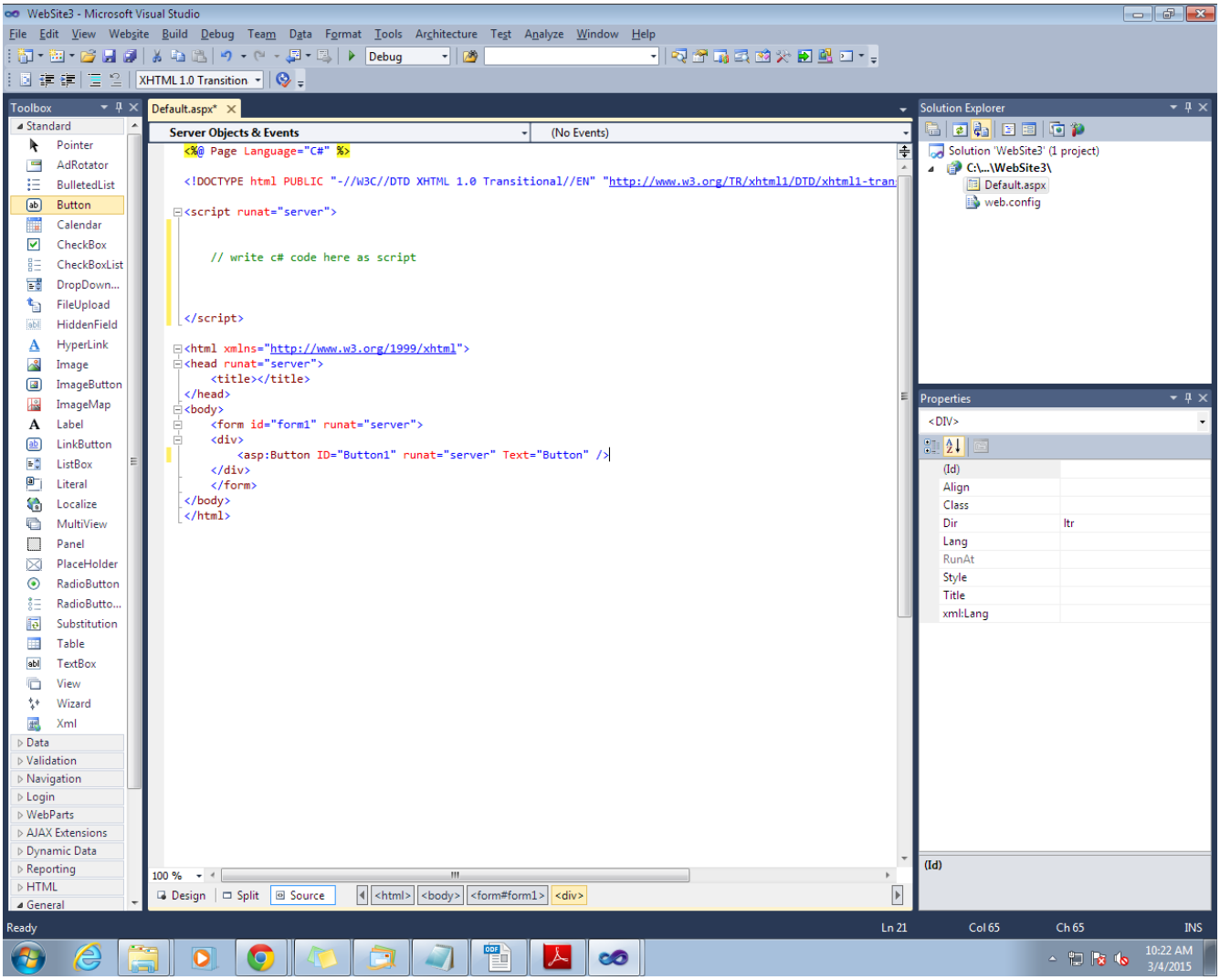
Which tells compiler about the Source code language of web page . Always remember one thing that we include any source code C# either in <script> tag or in <% %> tag which provides the functionality to add on source code in between HTML code also..
as done for Page declaration .

Just below it we are having certain other line also..

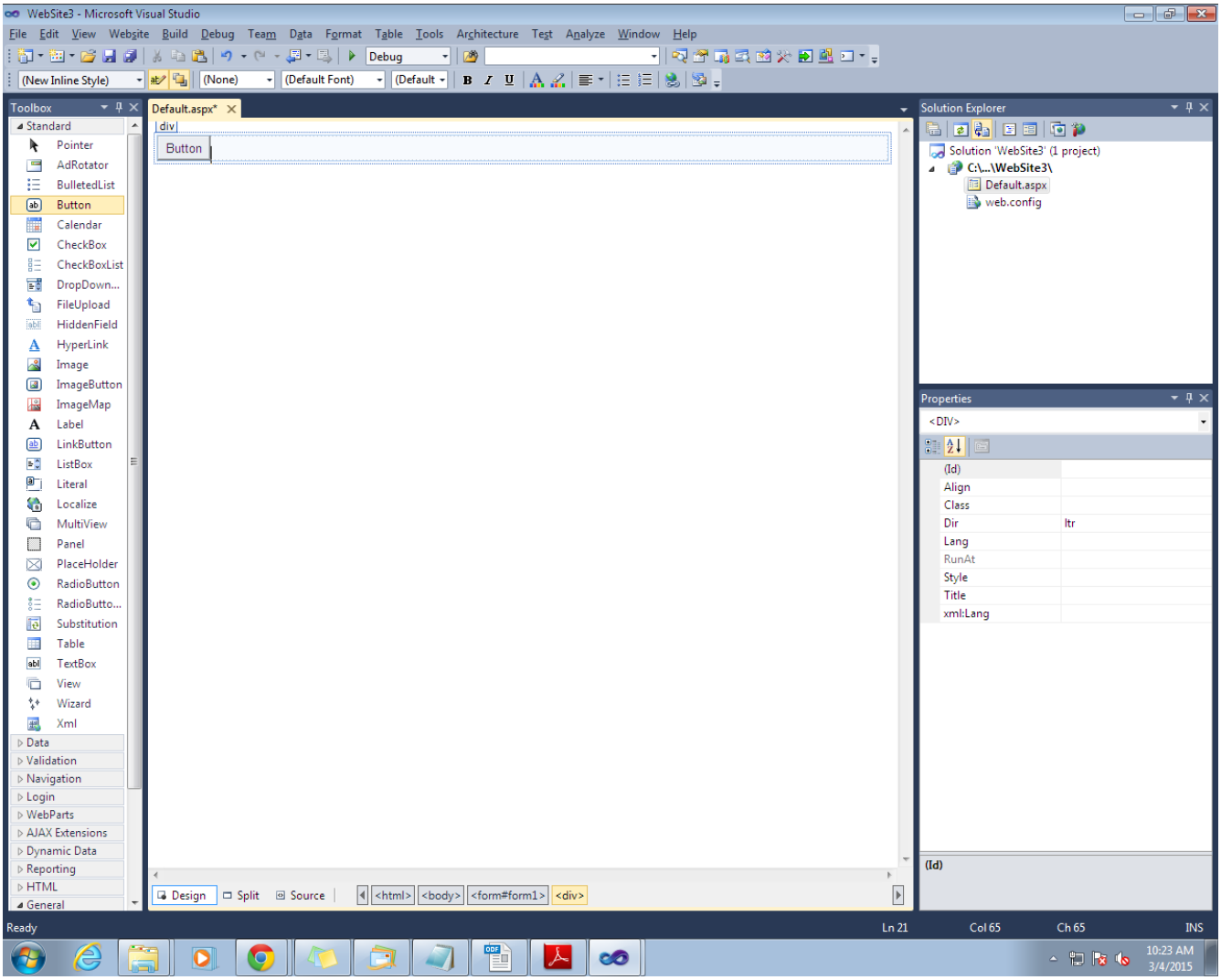
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

this indicates the doctype for the rules to be followed by w3C.

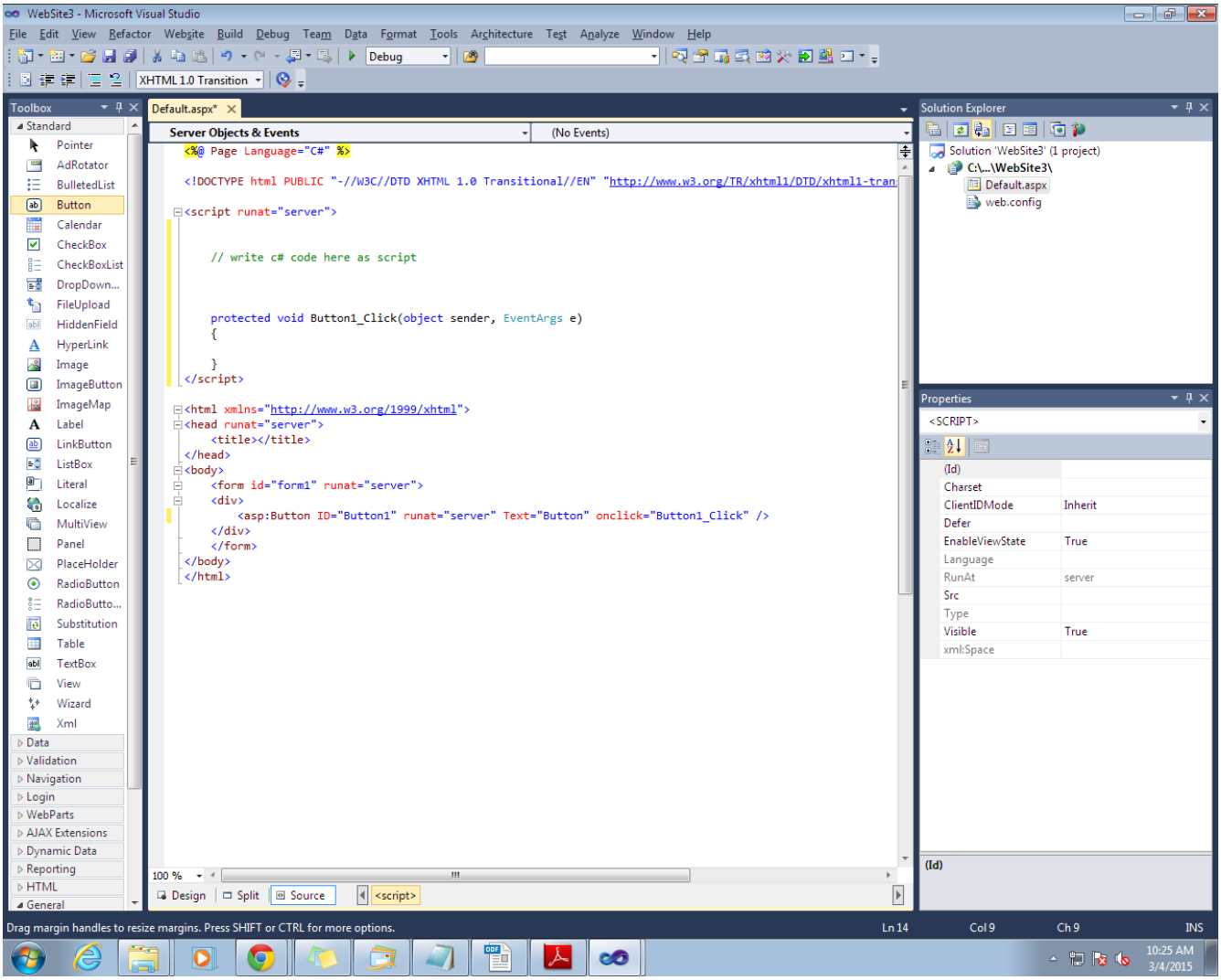
2.Add the button from the toolbox at left into the web form .



3.Click on the Design Tab at left bottom of web form



4. Double click on the button control , due to which the default event of button which click event will be added on it and it's respective event handler will be added into the script tag ..
see the image below ..



Let us understand the asp control tag below :

```
<asp:Button ID="Button1" runat="server" Text="Button" onclick="Button1_Click" />
```

Every asp control Is started with prefix asp and should have runat attribute set to server .. Due to double clicking the button ,we have onclick event is addedd with event handler name Button1_Click whose functionality is done in script tag above.
Like

```
<script runat="server">

    protected void Button1_Click(object sender, EventArgs e)
    {

    }

</script>
```

5.
Add the code in event handler to display the Current time .

```
<script runat="server">
 protected void Button1_Click(object sender, EventArgs e)
    {
        Response.Write(DateTime.Now.ToString());
    }
</script>
```

6. Importing more namespaces for code .. Add the name space for coloring like System.Drawing and add the code in buttton click event handler so that when user click on button then it's back color should be changed with time coming at top ..

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Drawing"  %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">
 protected void Button1_Click(object sender, EventArgs e)
    {

        Button1.BackColor = Color.Red;
        Response.Write(DateTime.Now.ToString());

    }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Button ID="Button1" runat="server" Text="Button" onclick="Button1_Click" />
    </div>
    </form>
</body>
</html>
```

7. Adding Page level events like Page_Load event ,

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Drawing"  %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">
 protected void Button1_Click(object sender, EventArgs e)
    {

        Button1.BackColor = Color.Red;


    }
```

```
 protected void Page_Load(object sender, EventArgs e)
 {
     Response.Write(DateTime.Now.ToString());
 }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Button ID="Button1" runat="server" Text="Button" onclick="Button1_Click" />
    </div>
    </form>
</body>
</html>
```

If you see , we have added the Page_Load event .. By default the page level events are bound to the page and can work with explicitly adding it to page as we do for other controls like button and dropdownlist . If you run the web page , the load event will work properly.

One of the property of Page called AutoEventWireUp property controls event binding to the page . By default it is true but can be made false if not required automatically binding .change the Page declaration like :

```
<%@ Page Language="C#" AutoEventWireup="false" %>
```

By making this property false , no any page level events will be bound for the handlers .

Now page load event will not work .

Next step is adding the web form with separate code file by checking at bottom check box while adding web form from new item dialog box.

If we make separate code file for webform , then we maintain two files one file called Default.aspx and other called Default.aspx.cs .

Default.aspx maintains the UI for web page which includes the html code with html and asp controls. Default.aspx.cs includes the business logic including all event handlers and user defined functions.

Let us see the Default.aspx code page below and understand it ..

```
<%@ Page Language="C#" AutoEventWireup="true"  CodeFile="Default11.aspx.cs"
Inherits="Default11" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
```

```
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>



    </div>
    </form>
</body>
</html>
```

if we see the code , we are having page declaration quite different from the page with single model .

Here page declaration includes by default certain properties like:
A. **Language** which is set to C#.
B. **AutoEventWireUp** which is set to true , in case you remove it then internally it is implemented with value true.
C.**CodeFile** property defines the code-behind file where we will maintain the source code which is of the same name but with extension .cs at last like Default.aspx.cs.
D.**Inherits** property tell us that this page is inherited from Default11 class which is mandatory.

Let us see the code-behind file called Default.aspx.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class Default11 : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        Response.Write(DateTime.Now.ToString());
    }
}
```

Code Explanation:

At the top we are having certain namespaces .

Class Default11 is made partial which indicates that certain part of this class is going to be used some where also like in Default.aspx page where we have used Inherits property.

Let us first understand the meaning of partial classes.

Partial Classes

Partial classes give you the ability to split a single class into more than one C# source code (.cs) file. For example, if the Product class became particularly long and intricate, you might decide to break it into two pieces

For e.g you want to divide the single class into multiple class , than you can declare the classes partial

For E.G. we are having one file called prodruct.cs with code given below :

```
using System;
partial class products
{
    int a, b ,sum;

    void accept()
    {
        a = 9;
        b = 10;
    }
    void add()
    {
        sum = a + b;
    }
}
```

Now you want to split the same class into one more class than the code will be :

```
using System;
partial class products
{
    void display()
    {
        Console.WriteLine("Sum is : " + sum);
    }

    public static void Main()
    {
        products ob = new products();
        ob.accept();
        ob.add();
        ob.display();
    }
}
```

```
if we see both classes are having the same name products means both classes are same internally
but split-ted into two classes .
```

Therefore in webforms also we are having two file called Default.aspx and second called Default.aspx.cs both files are same but are split-ted into two ,where one provides the UI and second provides the code file called .cs

Let us understand some terms in deep :

The Page Directive

The Default.aspx page, like all ASP.NET web forms, consists of three sections. The first section is the *page directive*:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Default.aspx.cs" Inherits="_Default" %>
```

The page directive gives ASP.NET basic information about how to compile the page. It indicates the language you're using for your code and the way you connect your event handlers. If you're using the code-behind approach (which is recommended), the page directive also indicates where the code file is located and the name of your custom page class. You won't need to modify the page directive by hand, because Visual Studio maintains it for you.

Note The page directive is for ASP.NET's eyes only. The page directive doesn't appear in the HTML that's sent to the browser—instead, ASP.NET strips it out.

The Doctype

In an ordinary, non-ASP.NET web page, the *doctype* occupies the very first line. In an ASP.NET web form, the doctype gets second place and appears just underneath the page directive.
The doctype indicates the type of markup (for example, HTML or XHTML) that you're using to create your web page. Technically, the doctype is optional, but Visual Studio adds it automatically. This is important, because depending on the type of markup you're using, there may be certain tricks that aren't allowed. For example, strict XHTML doesn't let you use HTML formatting features that are considered obsolete and have been replaced by CSS.

The doctype is also important because it influences how a browser interprets your web page. For example, if you don't include a doctype on your web page, Internet Explorer (IE) switches itself into a legacy mode known as *quirks mode*. While IE is in quirks mode, certain formatting details are processed in inconsistent, nonstandard ways, simply because this is historically the way IE behaved. Later versions of IE don't attempt to change this behavior, even though it's faulty, because some websites may depend on it. However, you can specify a more standardized rendering that more closely matches the behavior of other browsers (like Firefox) by adding a doctype.

Below is the example of doctype used in webform of asp.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

All the latest and updated website is using XHTML not HTML due to certain issues mentioned below in deep.

HTML and XHTML

What Is XHTML?

- XHTML stands for E**x**tensible Hyper Text **M**arkup **L**anguage
- XHTML is almost identical to HTML
- XHTML is stricter than HTML
- XHTML is HTML defined as an XML application
- XHTML is supported by all major browsers

Why XHTML?

Many pages on the internet contain "bad" HTML.

This HTML code works fine in most browsers (even if it does not follow the HTML rules):

```
<html>
<head>
  <title>This is bad HTML</title>

<body>
  <h1>Bad HTML
  <p>This is a paragraph
</body>
```

Today's market consists of different browser technologies. Some browsers run on computers, and some browsers run on mobile phones or other small devices. Smaller devices often lack the resources or power to interpret "bad" markup.

XML is a markup language where documents must be marked up correctly (be "well-formed").

If you want to study XML, please read our XML tutorial.

By combining the strengths of HTML and XML, XHTML was developed.

XHTML is HTML redesigned as XML.

The Most Important Differences from HTML:

Document Structure

- XHTML DOCTYPE is **mandatory**
- The xmlns attribute in <html> is **mandatory**
- <html>, <head>, <title>, and <body> are **mandatory**

XHTML Elements

- XHTML elements must be **properly nested**
- XHTML elements must always be **closed**
- XHTML elements must be in **lowercase**
- XHTML documents must have **one root element**

XHTML Attributes

- Attribute names must be in **lower case**
- Attribute values must be **quoted**
- Attribute minimization is **forbidden**

<!DOCTYPE ....> Is Mandatory

An XHTML document must have an XHTML DOCTYPE declaration.

A complete list of all the XHTML Doctypes is found in our HTML Tags Reference.

The <html>, <head>, <title>, and <body> elements must also be present, and the xmlns attribute in <html> must specify the xml namespace for the document.

This example shows an XHTML document with a minimum of required tags:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
  <title>Title of document</title>
</head>

<body>
  some content
</body>

</html>
```

XHTML Elements Must Be Properly Nested

In HTML, some elements can be improperly nested within each other, like this:

```
<b><i>This text is bold and italic</b></i>
```

In XHTML, all elements must be properly nested within each other, like this:

```
<b><i>This text is bold and italic</i></b>
```

XHTML Elements Must Always Be Closed

This is wrong:

```
<p>This is a paragraph
<p>This is another paragraph
```

This is correct:

```
<p>This is a paragraph</p>
<p>This is another paragraph</p>
```

Empty Elements Must Also Be Closed

This is wrong:

```
A break: <br>
A horizontal rule: <hr>
```

```
An image: <img src="happy.gif" alt="Happy face">
```

This is correct:

```
A break: <br />
A horizontal rule: <hr />
An image: <img src="happy.gif" alt="Happy face" />
```

XHTML Elements Must Be In Lower Case

This is wrong:

```
<BODY>
<P>This is a paragraph</P>
</BODY>
```

This is correct:

```
<body>
<p>This is a paragraph</p>
</body>
```

XHTML Attribute Names Must Be In Lower Case

This is wrong:

```
<table WIDTH="100%">
```

This is correct:

```
<table width="100%">
```

Attribute Values Must Be Quoted

This is wrong:

```
<table width=100%>
```

This is correct:

```
<table width="100%">
```

Attribute Minimization Is Forbidden

Wrong:

```
<input type="checkbox" name="vehicle" value="car" checked />
```

Correct:

```
<input type="checkbox" name="vehicle" value="car" checked="checked" />
```

Wrong:

```
<input type="text" name="lastname" disabled />
```

Correct:

```
<input type="text" name="lastname" disabled="disabled" />
```

How to Convert from HTML to XHTML

1. Add an XHTML <!DOCTYPE> to the first line of every page
2. Add an xmlns attribute to the html element of every page
3. Change all element names to lowercase
4. Close all empty elements
5. Change all attribute names to lowercase
6. Quote all attribute values

This is good practice of writing the XHTML instead of HTML for avoiding browser issues .

Elements

The most important concept in the XHTML (and HTML) standard is the idea of elements. Elements are containers that contain bits of your web page content. For example, if you want to add a paragraph of text to a web page, you stuff it inside a paragraph element.

Elements are the good concepts for developing the web page . A single web page consists of hundreds of elements for making it working.

The XHTML language defines a small set of elements that you can use—in fact, there are fewer than you probably expect. XHTML also defines the syntax for using these elements. A typical element consists of three pieces: a start tag, some content, and an end tag. Here's an example:

This is a sentence in a paragraph.
This example uses the paragraph element. The element starts with the start tag, ends with the end tag, and contains some text inside

Tags are easy to recognize, because they're always enclosed
in angled brackets. And here's a combination that adds a heading to a web page followed
by a paragraph:

<h1>A Heading</h1>
This is a sentence in a paragraph.

Browsers have built-in rules about how to process and display different elements. When a browser digests this markup, it always places the heading in a large, bold font and adds a line break and some extra space underneath it, before starting the paragraph.

Of course, there are ways to modify these formatting rules using the CSS standard,

Many XHTML elements can contain other elements. For example, you can use the element inside the element to apply bold formatting to a portion of a paragraph:

This is a sentence in a paragraph.

The <h1> and elements usually hold content inside. As a result, they're split into a start tag and an end tag. For example, a heading begins with <h1> and ends with </h1>. However, some elements don't need any content and can be declared using a special empty tag syntax that fuses the start and end
tag together. For example, the
 element represents a line break. Rather than writing
</br>, you can simply use
, as shown here:
This is line one.
This is line two.
This is line three.
Other elements that can be used in this fashion include <img> (for showing an image), <hr> (for creating a horizontal rule, or line), and most ASP.NET controls.

Below is the list of XHTML elements and their functionality .

Tag	Name	Type	Description
<b>, <i>, <u>	Bold, Italic, Underline	Container	These elements are used to apply basic formatting and make text bold, italic, or underlined. Some web designers prefer to use instead of <b> and <emphasis> instead of <i>. Although these elements have the same standard rendering (bold and italic, respectively), they make more sense if you plan to use styles to change the formatting sometime in the future.
<p>	Paragraph	Container	The paragraph groups a block of free-flowing text together. The browser automatically adds a bit of space between paragraphs and other elements (such as headings) or between subsequent paragraphs.
<h1>, <h2>, <h3>, <h4>, <h5>, <h6>	Heading	Container	These elements are headings, which give text bold formatting and a large font size. The lower the number, the larger the text, so <h1> is for the largest heading. The <h5> heading is normal text size, and <h6> is actually a bit smaller than ordinary text.
<img>	Image	Stand-alone	The image element shows an external image file (specified by the src attribute) in a web page.
 	Line Break	Stand-alone	This element adds a single line break, with no extra space.
<hr>	Horizontal Line	Stand-alone	This element adds a horizontal line (which gets the full width of the containing element). You can use the horizontal line to separate different content regions.
<a>	Anchor	Container	The anchor element wraps a piece of text and turns it into a link. You set the link target using the href attribute.
,	Unordered List, List Item	Container	These elements allow you to build bulleted lists. The element defines the list, while the element defines an item in the list (you nest the actual content for that item inside).
,	Ordered List, List Item	Container	These elements allow you to build numbered lists. The element defines the list, while the element defines an item in the list (you nest the actual content for that item inside).
<table>, <tr>, <td>, <th>	Table	Container	The <table> element allows you to create a multicolumn, multirow table. Each row is represented by a <tr> element inside the <table>. Each cell in a row is represented by a <td> element inside a <tr>. You place the actual content for the cell in the individual <td> elements (or, in the case of the header cells that sit at the top of the table, you can use <th> elements instead).
	Division	Container	This element is an all-purpose container for other elements. It's used to separate different regions on the page, so you can format them or position them separately. For example, you can use a to create a shaded box around a group of elements.
	Span	Container	This element is an all-purpose container for bits of text content inside other elements (such as headings or paragraphs). It's most commonly used to format those bits of text. For example, you can use a to change the color of a few words in a sentence.
<form>	Form	Container	This element is used to hold all the controls on a web page. Controls are HTML elements that can send information back to the web server when the page is submitted. For example, text boxes submit their text, list boxes submit the currently selected item in the list, and so on.

Attributes

Every XHTML document fuses together two types of information: the document content and information about how that content should be presented. You control the presentation of your content in just three ways: by using the right elements, by arranging these elements to get the right structure, and by adding attributes to your elements.

The <img> tag requires two pieces of information—the image URL (the source) and the alternate text that describes the picture (which is used for accessibility purposes, as with screen-reading software).

These two pieces of information are specified using two attributes, named src and alt:

The <a> anchor element is an example of an element that uses attributes and takes content. The content inside the <a> element is the blue, underlined text of the hyperlink. The href attribute defines the destination that the browser will navigate to when the link is clicked.

Click here to visit my website.

You'll use attributes extensively with ASP.NET control tags. With ASP.NET controls, every attribute maps to a property of the control class.

Formatting

Along with the <b> (or) element for bold, XHTML also supports <i> (or <emphasis>) element for italics. However, this is about as far its formatting goes.
XHTML elements are intended to indicate the structure of a document, not its formatting. Although you can adjust colors, fonts, and some formatting characteristics using XHTML elements, a better approach is to define formatting using a CSS style sheet.

For example, a style sheet can tell the browser to use specific formatting for every <h1> element in a page. You can even apply the styles in a style sheet to all the pages in your website.

Complete Web Page

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default4.aspx.cs" Inherits="Default4"
%>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
```

```
    <div>

    </div>
    </form>
</body>
</html>
```

In an ASP.NET web page , Inside the <body> element is a <form> element. The <form> element is required because it defines a portion of the page that can send information back to the web server. This becomes important when you start adding text boxes, lists, and other controls. As long as they're in a form, information like the current text in the text box and the current selection in the list will be sent to the web server using a process known as a postback.

the element is optional—it's just a container. You can think of it as an invisible box that has no built-in appearance or formatting. However, it's useful to use a tag to group portions of your page that you want to format in a similar way (for example, with the same font, background color, or border). That way, you can apply style settings to the tag, and they'll cascade down into every tag it contains. You can also create a real box on your page by giving the a border.

Writing Code

To start coding, you need to switch to the code-behind view. To switch back and forth, you can use two View Code or View Designer buttons, which appear just above the Solution Explorer window. Another approach that works just as well is to double-click either the .aspx page in the Solution Explorer (to get to the designer) or the .aspx.cs page (to get to the code view).

As we have already discussed about code-behind view.

Coding is a specifically practice of implementation business logic with the help of events and its based event handlers.

What are Events ?

Events are the good methodology in .Net which provides the functionality based on actions . Events allows you to do certain task when user do any action which may be clicking , moving mouse , pressing key etc. We have provided various events to be implemented like click events , mouse events ,press events and many more . Every control or web page is associated with some events that we can add on it . For example let us talk about click event , for example I am using the control button which allows user to click from mouse . But what actually should happen after user clicks on it is controlled by events . So to implement the functionality of clicking we need to add click event to the button . Every event requires the reference of the special type of methods called event handlers.

Button --------->Added Event on it(E.G.Click) ------> having reference of ------>Event handler(method)

Event handlers contains the code to be executed when user will click on button ..

Adding Event Handlers

Most of the code in an ASP.NET web page is placed inside event handlers that react to web control events. Using Visual Studio, you have three easy ways to add an event handler to your code:

A. Type it in manually: In this case, you add the subroutine directly to the page class in your C# code file. You must specify the appropriate parameters

B. Double-click a control in design view: In this case, Visual Studio will create an event handler for that control's default event, if it doesn't already exist. For example, if you double-click a Button control, it will create an event handler for the Button. Click event. If you double-click a Text Box control, you'll get an event handler for the TextBox.TextChanged event. If the event handler already exists, Visual Studio simply takes you to the relevant place in your code.

C. Choose the event from the Properties window: Just select the control, and click the lightning bolt in the Properties window. You'll see a list of all the events provided by that control. Double-click next to the event you want to handle, and Visual Studio will automatically generate the event handler in your page class. Alternatively, if you've already created the event handler method, just select the event in the Properties window, and click the drop-down arrow at the right. You'll see a list that includes all the methods in your class that match the signature this event requires. You can then choose a method from the list to connect it.

No matter which approach you use, the event handler looks (and functions) the same.
For example, when you double-click a Button control, Visual Studio creates an event handler like this:

```
protected void Button1_Click(object sender, EventArgs e)
 {

 }
```

```
<asp:Button ID="Button1" runat="server" Text="Button" onclick="Button1_Click" />
```

By clicking double times on button , visual studio automatically add the default event of button E.G. Click event by adding one more attribute E.G. "onclick" and automatically we get the event handler on code-behind page as displayed above .

Web Page Control Categories

A. Standard
B. Data
C. Validation
D. Navigation
E. Logan
F.WebParts
G. Ajax Extensions
H. Dynamic Data
I. Reporting
J.HTML

When you create ASP.NET Web pages, you can use these types of controls:

A.HTML server controls :HTML elements exposed to the server so you can program them. HTML server controls expose an object model that maps very closely to the HTML elements that they render.

B. Web server controls :Controls with more built-in features than HTML server controls. Web server controls include not only form controls such as buttons and text boxes, but also special-purpose controls such as a calendar, menus, and a tree view control. Web server controls are more abstract than HTML server controls in that their object model does not necessarily reflect HTML syntax.

C. Validation controls:Controls that incorporate logic to enable you to what users enter for input controls such as the TextBox control. Validation controls enable you to check for a required field, to test against a specific value or pattern of characters, to verify that a value lies within a range, and so on.

D. User controls Controls that you create as ASP.NET Web pages. You can embed ASP.NET user controls in other ASP.NET Web pages, which is an easy way to create toolbar and other reusable elements. For more information, see ASP.NET User Controls.

ASP.NET web page provides different types of controls , one of it is HTML controls ..

HTML Server Controls :

HTML server controls are HTML elements (or elements in other supported markup, such as XHTML) containing attributes that make them programmable in server code. By default, HTML elements on an ASP.NET Web page are not available to the server. Instead, they are treated as opaque text and passed through to the browser. However, by converting HTML elements to HTML server controls, you expose them as elements you can program on the server.

The object model for HTML server controls maps closely to that of the corresponding elements. For example, HTML attributes are exposed in HTML server controls as properties.

Any HTML element on a page can be converted to an HTML server control by adding the attribute runat="server". During parsing, the ASP.NET page framework creates instances of all elements containing the runat="server" attribute. If you want to reference the control as a member within your code, you should also assign an id attribute to the control.

The page framework provides predefined HTML server controls for the HTML elements most commonly used dynamically on a page: the form element, the input elements (text box, check box, Submit button), the select element, and so on. These predefined HTML server controls share the basic properties of the generic control, and in addition, each control typically provides its own set of properties and its own event.

Let us talk about the HTML controls

HTML controls are basically divided into
A. Input controls
B. Textarea
C. Table
D. Image.
E. Select.
F. Horizontal Line.
G. Div

HTML Input Controls

The following controls, which are based on the HTML INPUT element, are available on the HTML tab of the Toolbox:

Input (Button) control: INPUT type="button" element
Input (Checkbox) control: INPUT type="checkbox" element
Input (File) control: INPUT type="file" element
Input (Hidden) control: INPUT type="hidden" element
Input (Password) control: INPUT type="password" element
Input (Radio) control: INPUT type="radio" element
Input (Reset) control: INPUT type="reset" element
Input (Submit) control: INPUT type="submit" element
Input (Text) control: INPUT type="text" element

HTMLInputButton , HTMLInputSubmit , HTMLInputReset Controls:

HTML provides three type of Button controls
A. Button
B. Submit
C. Reset

We can use all above mentioned controls either at client-based or server-based .

```
<input id="Button1" type="button" value="button" /> // client-based

<input id="Button2" type="button" value="button" runat="server"/> //server-based
```