❖ Framework History:

| Generation | Release Date | Development Tool |
|---|---|---|
| 1.0 | 13-02-2002 | Visual Studio.NET |
| 1.1 | 24-04-2003 | Visual Studio.NET 2003 |
| 2.0 | 07-11-2005 | Visual Studio.NET 2005 |
| 3.0 | 06-11-2006 | Microsoft Blend |
| 3.5 | 19-11-2007 | Visual Studio 2008 |
| 4.0 | 12-04-2010 | Visual Studio 2010 |
| 4.5 | 15-08-2012 | Visual Studio 2012 |
| 4.5.1 | 07-10-2013 | Visual Studio 2013 |
| 4.5.2 | 05-05-2014 | Visual Studio 2013 |

❖ Introduction of Visual Web Developer:

Visual Web Developer 2005 Express Edition is part of the Microsoft Visual Studio 2005 family, and is the best development tool for building data driven web applications with ASP.NET 2.0. As part of the Express family, Visual Web Developer provides a seamless upgrade path to Visual Studio Standard, Professional, and Team System.

Visual Web Developer is tuned to the specific needs of the Web developer through a new Web profile that exposes a menu and window layout optimized for Web development. The environment includes a best-of-breed HTML source editor, an improved visual page designer, a new project system, better support for working with data, and full XHTML standards support. Collectively, these features enable you to develop data-driven Web applications faster and easier than ever before. Below we'll dive in and explore a few of the many Web development improvements coming with Visual Web Developer.

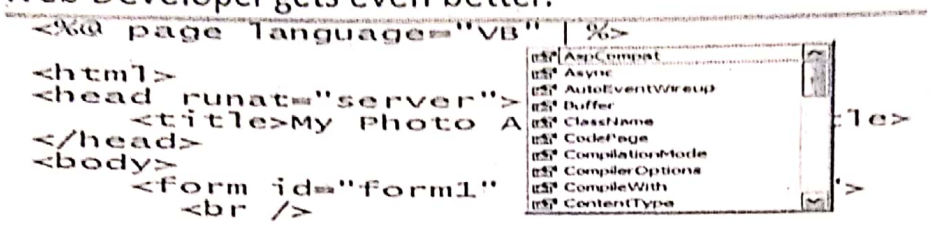
**Better Source Code Editing**

Visual Web Developer has an improved HTML source editor which enables you to write and modify your pages faster.

Scanned by CamScanner

## Intellisense Everywhere

**Intellisense** -- the popup code hints which appear while you type -- has a dramatic impact on your productivity as a developer. While support for Intellisense in Visual Studio is excellent today, support for Intellisense in Visual Web Developer gets even better.




## HTML Source Preservation

Visual Web Developer respects your HTML. The formatting of your HTML markup -- including all white space, casing, indention, carriage returns, and word wrapping -- is now preserved exactly as originally written, even when switching back and forth between the design view and source view of the page.

## HTML Formatting Options

Visual Web Developer enables you to precisely control the format of all HTML and ASP.NET Server Control markup generated using the WYSIWYG designer. You can now configure the tag casing, attribute quotation, indention style and word wrap characteristics of every html or server control tag in a page.

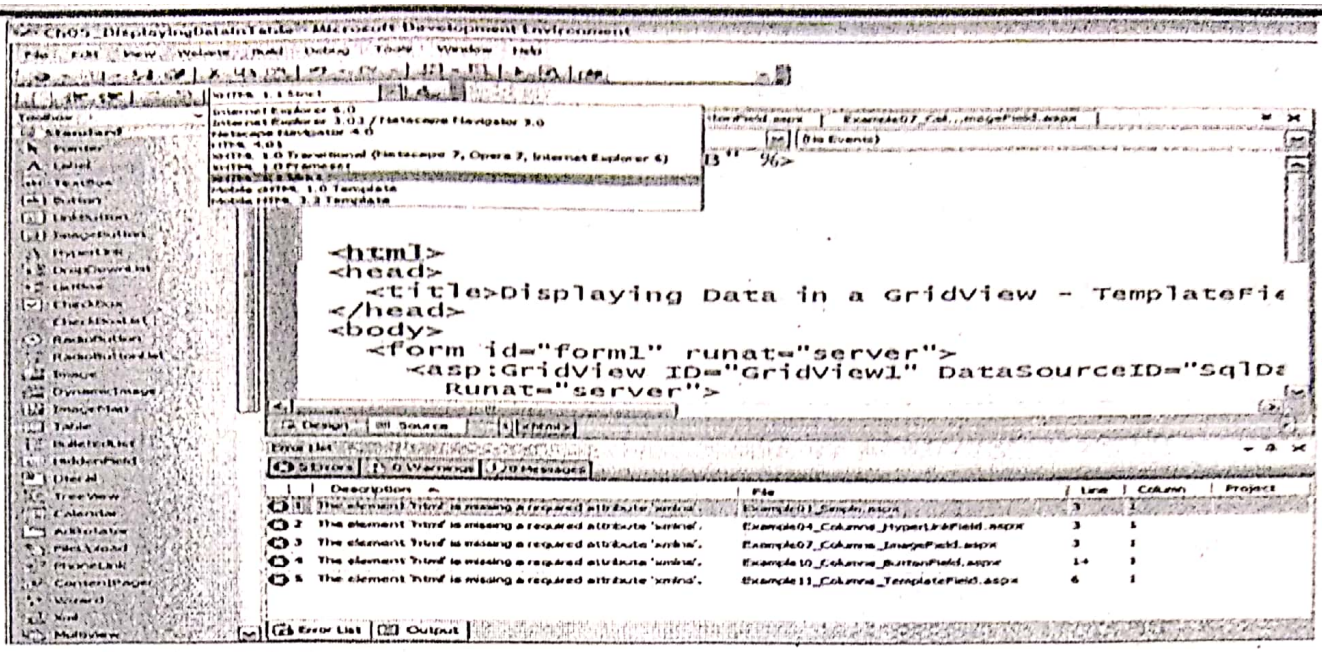
## Flexible Browser Targeting and Validation

Visual Web Developer enables you to easily target a specific HTML standard or browser when writing your HTML pages. For example, you can target your HTML pages to work with a particular browser such as Netscape Navigator 4.0 or Internet Explorer 6.0.

Your HTML will then be validated in real-time as you type in the source editor. Invalid HTML will automatically be underlined with a red squiggly (with a tooltip displaying an explanation of precisely how you violated the target).

All browser/standard validation rules are pluggable within Visual Web Developer, and can be easily extended and customized by developers. Visual Web Developer will include a number of automatic validation targets out of the box including built-in validation support for XHTML, XHTML Transitional, and all major desktop and mobile browsers.

Scanned by CamScanner




*Selecting a validation target*

## Code Refactoring

Code Refactoring enables you to easily and systematically make changes to your code. Code Refactoring is supported everywhere that you can write code including both code-behind and single-file ASP.NET pages. For example, you can use Code Refactoring to automatically promote a public field to a full property.

## Richer Visual Designer

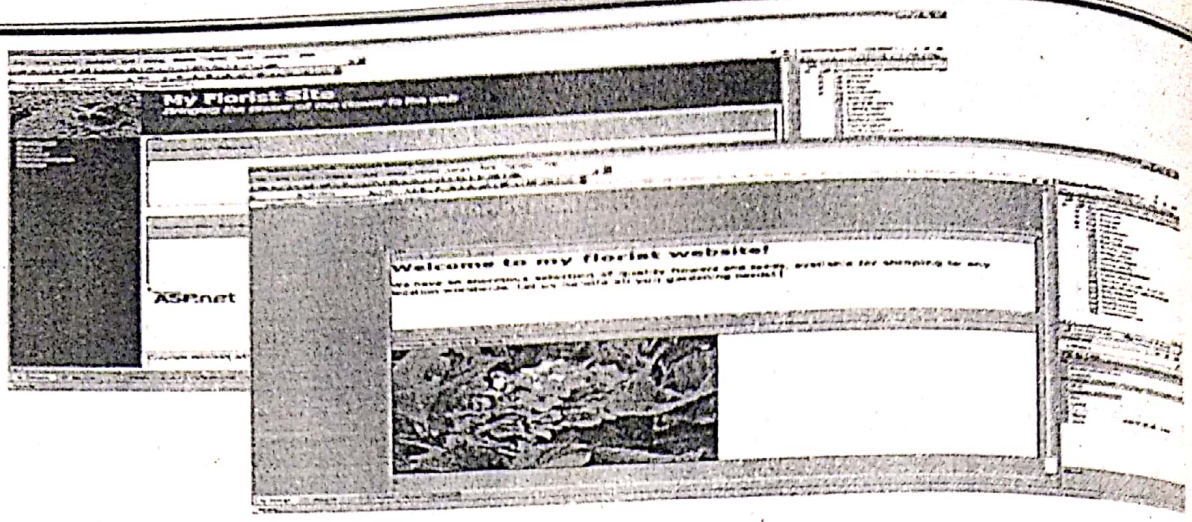
Visual Web Developer has an improved designer which makes it easier to visually build ASP.NET Web applications. By taking advantage of the new designer tools, you can build a feature rich, database-driven Web application without writing a single line of code.

## ASP.NET Master Page Designer

Master Pages enable you to create a common look and feel for the pages in an ASP.NET Web application. You can take advantage of Master Pages to create a single page layout and apply the page layout to multiple Content Pages. For example, you can use a Master Page to ensure that every page in an application contains the same standard header, footer, and navigation bar.

After you create a Master Page, you can apply the Master to new ASP.NET Pages within your Web site. Visual Web Developer provides great editing support when authoring an ASP.NET page based on a Master enabling the designer to see what the combined page will look like.

Scanned by CamScanner

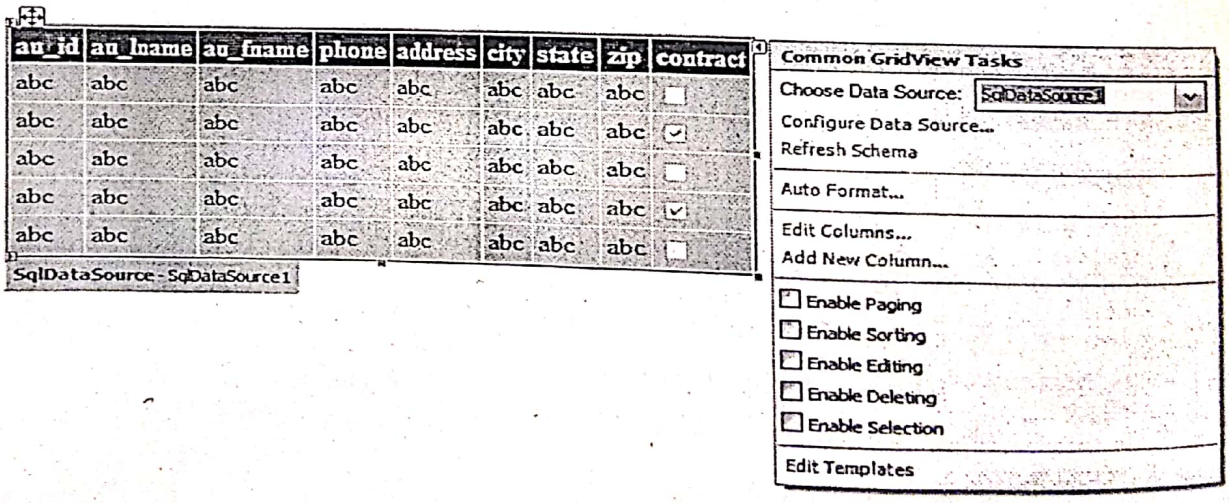



*Master Pages in the designer*

## Smart Tasks

Visual Web Developer enables you to perform many of the most commor programming tasks directly from the designer surface. By taking advantage of Smart Tasks, You can create an entire, feature rich, database-driven Web application without writing a single line of code.

When you drag new controls onto the designer surface, a popup list of common tasks automatically appears. You can use the common tasks list to quickly configure a control's properties, as well as walk through common operations you might perform with it. For example, when you add a GridView control to a page, a common task list appears which enables you to quickly enable sorting, paging, or editing for the GridView. When you add a TextBox control to a page, a common task list appears which enables you to quickly associate a validation control with the control.




*Enabling GridView options with Smart Tasks*

Scanned by CamScanner

❖ Basics of Web Application Development:

## State Management

**Application State**: Application state is used to store data corresponding to all the variables of an ASP.NET Web application. The data in application state is stored once and read several times. Application state uses the HttpApplicationstate class to store and share the data throughout the application. You can access the information stored in an applicationstate by using the HttpApplication class property. Data stored in application state is accessible to all the pages of the application and is the same for all users accessing the application. The HttpApplicationstate class provides a lock, method, which you can use to ensure that only one user is able to access and modify the data of an application at any instant of time.

**Session State:** Each client accessing a Web application maintains a distinct session with theWeb server, and there is also specific information associated with each of these sessions. Session state is defined in the <sessionState> section of the web.config file. It also stores the data specific to a user session in session variables. Different session variables are created for each user session. In addition, session variables can be accessed from any page of the application. When a user accesses a page, a session ID for the user is created. The session ID is transferred between the server and the client over the HTTP protocol using cookies.

**View State**: View state stores page specific information, when a page is posted back to the server. When a page is processed, the current state of the page and its controls is hashed (transforming a sequence of character into a fixed-length value) into a string and saved as a hidden field on the page. Such a state of the page is called view state.

**Query String:**Query string is used to transfer small amount of data from one web page to another web page. It does not provide security because the data can be shown in the address bar.

Scanned by CamScanner

## Introduction of Web Forms:

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.

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.

### 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.

Scanned by CamScanner

❖ **Advantages:**

12 important advantages ASP.NET offers over other Web development models:

1. ASP.NET drastically reduces the amount of code required to build large applications.
2. With built-in Windows authentication and per-application configuration, your applications are safe and secured.
3. It provides better performance by taking advantage of early binding, just-in-time compilation, native optimization, and caching services right out of the box.
4. The ASP.NET framework is complemented by a rich toolbox and designer in the Visual Studio integrated development environment. WYSIWYG editing, drag-and-drop server controls, and automatic deployment are just a few of the features this powerful tool provides.
5. Provides simplicity as ASP.NET makes it easy to perform common tasks, from simple form submission and client authentication to deployment and site configuration.
6. The source code and HTML are together therefore ASP.NET pages are easy to maintain and write. Also the source code is executed on the server. This provides a lot of power and flexibility to the web pages.
7. All the processes are closely monitored and managed by the ASP.NET runtime, so that if process is dead, a new process can be created in its place, which helps keep your application constantly available to handle requests.
8. It is purely server-side technology so, ASP.NET code executes on the server before it is sent to the browser.
9. Being language-independent, it allows you to choose the language that best applies to your application or partition your application across many languages.
10. ASP.NET makes for easy deployment. There is no need to register components because the configuration information is built-in.
11. The Web server continuously monitors the pages, components and applications running on it. If it notices any memory leaks, infinite loops, other illegal activities, it immediately destroys those activities and restarts itself.

Scanned by CamScanner

12. Easily works with ADO.NET using data-binding and page formatting features. It is an application which runs faster and counters large volumes of users without having performance problems

❖ **New in ASP.NET 2.0:**

- Better language support
- Programmable controls
- Event-driven programming
- XML-based components
- User authentication, with accounts and roles
- Higher scalability
- Increased performance - Compiled code
- Easier configuration and deployment
- Not fully ASP compatible

❖ **New Features in ASP.NET 2.0**

**Master Pages**

Master pages are a new feature introduced in ASP.NET 2.0 to help you reduce development time for Web applications by defining a single location to maintain a consistent look and feel in a site. Master pages allow you to design a template that can be used to generate a common layout for many pages in the application. The primary goal of master pages is to avoid creating each page from scratch and having to repeat the layout code. Another benefit of using master pages is that, if you want to change the layout of the pages in the application, you only have to update the master page rather than each individual page. This feature is somewhat similar to the Windows Form technique of *Visual Inheritance*, which was available with the original version of the .NET Framework and is used for desktop application development.

**Content Pages**

Content pages are attached to a master-page and define content for any ContentPlaceHolder controls in the master page. The content page contains <asp:content> controls which reference the <asp:contentPlaceHolder> controls in the master page through the ContentPlaceHolder id. The content pages and the master page combine to form a single response.

```
<%@ page language="VB" MasterPageFile="~/Mysite.master" %>
```

Scanned by CamScanner

```
<asp:content id="Content1"  contentplaceholderid="LeftSideContent">

<H2>Navigation </H2>

<asp:treeview id="Navigation tree" runat="server" datasourceid="NavSource"/>

</asp:content>

<asp:content id="Content1"  contentplaceholderid="RightSideContent">

<asp:label runat="server">Support section</asp:label>

</asp:content>
```

**Overriding Master Pages**

Despite the best planning and requirements review, situations will arise where a new feature has to be added to one particular web page. If you are considering abandoning your master page to add or remove functionality from a specific content page, you can override your master page.

Consider, for example, a case where you don't want a specific content page to display the standard navigation bar that you have built into your master page. If you add a configurable property to your master page, you can decide to turn the navigation bar on or off for each of your content pages.

**Themes and Skins**

Currently ASP developers must use Cascading Style Sheets (CSS) and inline styles to enforce a look and feel on a Web site. Although these technologies help, consistency is still primarily a function of developer discipline in using the correct styles. ASP.NET 2.0 rectifies this issue through the use of themes and skins, which are applied uniformly across every page and control in a website.

A theme is a set of skin files grouped together to make a specific look for a set of controls.

**Themes**

Themes are similar to CSS style sheets in that both themes and style sheets define a set of common attributes that apply to any page where the theme or style sheet is applied. However, themes differ from style sheets in the following ways

Scanned by CamScanner

Themes can define many properties of a control or page, not just a specific set of style properties.

Themes can include auxiliary files (e.g. graphics) that can't be included in a CSS style sheet.

Themes do not cascade the way style sheets do (e.g. theme property values always override local property values).

Themes can include style sheet references.

Each application has a themes directory. Each specific theme has its own subdirectory that contains skin files and any other files that apply to the style.

**Web Parts**

One of the major differences between a Web application and a desktop application has been the ease with which a desktop application can contain multiple configurable components. For example, consider the Visual Studio IDE itself. A user can decide which screens to display and how they are arranged. Developing similar functionality in a Web site is a daunting prospect.

ASP.NET 2.0 introduces a solution to this problem in the form of Web Parts. Web Parts are modular components that can be included and arranged by the user to create a productive interface that is not cluttered with unnecessary details. The user can:

Choose which parts to display

Configure the parts in any order or arrangement

Save the view from one Web session to the next

Customize the look of certain Web Parts.

All of these features are practically impossible to implement with ordinary Web applications.

You can think of Web Parts as modular Web page blocks. Each block can be added or removed from the Web page dynamically, at runtime. Code for organizing and manipulating Web Parts is built in to ASP.NET 2.0. All of the functionality for adding, removing and configuring layout is automatically handled by the Web Parts system. The programmer simply builds Web Parts and assigns them to Web

Scanned by CamScanner

Part Zones. A user can mix and match Web Parts, display them in any order, and expect the configuration to be saved between site visits.

❖ **Page Event Life Cycle:**

1. **PreInit** The entry point of the page life cycle is the pre-initialization phase called "PreInit". This is the only event where programmatic access to master pages and themes is allowed. You can dynamically set the values of master pages and themes in this event. You can also dynamically create controls in this event.
2. **Init** This event fires after each control has been initialized, each control's Unique ID is set and any skin settings have been applied. You can use this event to change initialization values for controls.
3. **InitComplete:** this event raise when once all initializations of the page and its controls have been completed. Till now the viewstate values are not yet loaded, hence you can use this event to make changes to view state that you want to make sure are persisted after the next postback.
4. **PreLoad:** this event raise after the page loads view state for itself and all controls, and after it processes postback data that is included with the Request instance.
5. **PreRender** Allows final changes to the page or its control. This event takes place after all regular PostBack events have taken place. This event takes place before saving ViewState, so any changes made here are saved.For example : After this event, you cannot change any property of a button or change any viewstate value. Because, after this event, SaveStateComplete and Render events are called.
6. **SaveStateComplete** Prior to this event the view state for the page and its controls is set. Any changes to the page's controls at this point or beyond are ignored.
7. **Render** This is a method of the page object and its controls (and not an event). At this point, ASP.NET calls this method on each of the page's controls to get its output. The Render method generates the client-side HTML, Dynamic Hypertext Markup Language (DHTML), and script that are necessary to properly display a control at the browser.
8. **Load** The important thing to note about this event is the fact that by now, the page has been restored to its previous state in case of postbacks. Code

Scanned by CamScanner

inside the page load event typically checks for PostBack and then sets control properties appropriately. This method is typically used for most code, since this is the first place in the page lifecycle that all values are restored. Most code checks the value of IsPostBack to avoid unnecessarily resetting state. You may also wish to call Validate and check the value of IsValid in this method. You can also create dynamic controls in this method.

9. **LoadComplete** This event signals the end of Load.
10. **Control (PostBack) event(s)**ASP.NET now calls any events on the page or its controls that caused the PostBack to occur. This might be a button's click event or a dropdown's selectedindexchange event.
11. **UnLoad** This event is used for cleanup code. After the page's HTML is rendered, the objects are disposed of. During this event, you should destroy any objects or references you have created in building the page. At this point, all processing has occurred and it is safe to dispose of any remaining objects, including the Page object. Cleanup can be performed on-

    a. Instances of classes i.e. objects
    b. Closing opened files
    c. Closing database connections.

❖ **Global Application Class:**

The Global.asax file, also known as the ASP.NET application file, is an optional file that contains code for responding to application-level events raised by ASP.NET or by HttpModules. The Global.asax file resides in the root directory of an ASP.NET-based application. At run time, Global.asax is parsed and compiled into a dynamically generated .NET Framework class derived from the HttpApplicationbase class. The Global.asax file itself is configured so that any direct URL request for it is automatically rejected; external users cannot download or view the code written within it.

The ASP.NET Global.asax file can coexist with the ASP Global.asax file. You can create a Global.asax file either in a WYSIWYG designer, in Notepad, or as a compiled class that you deploy in your application's \Bin directory as an assembly. However, in the latter case, you still need a Global.asax file that refers to the assembly.

Scanned by CamScanner

The Global.asax file is optional. If you do not define the file, the ASP.NET page framework assumes that you have not defined any application or session event handlers.

When you save changes to an active Global.asax file, the ASP.NET page framework detects that the file has been changed. It completes all current requests for the application, sends the Application_OnEnd event to any listeners, and restarts the application domain. In effect, this reboots the application, closing all browser sessions and flushing all state information. When the next incoming request from a browser arrives, the ASP.NET page framework reparses and recompiles the Global.asax file and raises the Application_OnStart event.

**Events which are not fired for every request:**

- **Application_Start()** – This event raised when the application starts up and application domain is created.
- **Session_Start()** – This event raised for each time a new session begins, This is a good place to put code that is session-specific.
- **Application_Error()** – This event raised whenever an unhandled exception occurs in the application. This provides an opportunity to implement generic application-wide error handling.
- **Session_End()** – This event called when session of user ends.
- **Application_End()** – This event raised just before when web application ends.
- **Application_Disposed()** – This event fired after the web application is destroyed and this event is used to reclaim the memory it occupies.

### ❖ Web Configuration file:

Web configuration file is used to configure all settings of the web application. This is very important file of the asp.net web application. The main purpose of the web configuration file is to provide all necessary information and security to the particular web application. The information and security to the application provides by different tags.

Scanned by CamScanner

For e.g. **connectionstring** tag provides the connection string of the database. There can be more than one connection string of the web application when application uses more than one database.

Web.config is the main settings and configuration file for an ASP.NET web application. It is an XML document that resides in the root directory of the site or application and contains data about how the web application will act. This information controls module loading, security configuration, session state configuration, and application language and compilation settings. Web.config files can also contain application specific items such as database connection strings.

The behavior of an ASP.NET application is affected by different settings in the configuration files:

- machine.config
- web.config

The machine.config file contains default and the machine-specific value for all supported settings. The machine settings are controlled by the system administrator and applications are generally not given access to this file.

An application however, can override the default values by creating web.config files in its roots folder. The web.config file is a subset of the machine.config file.

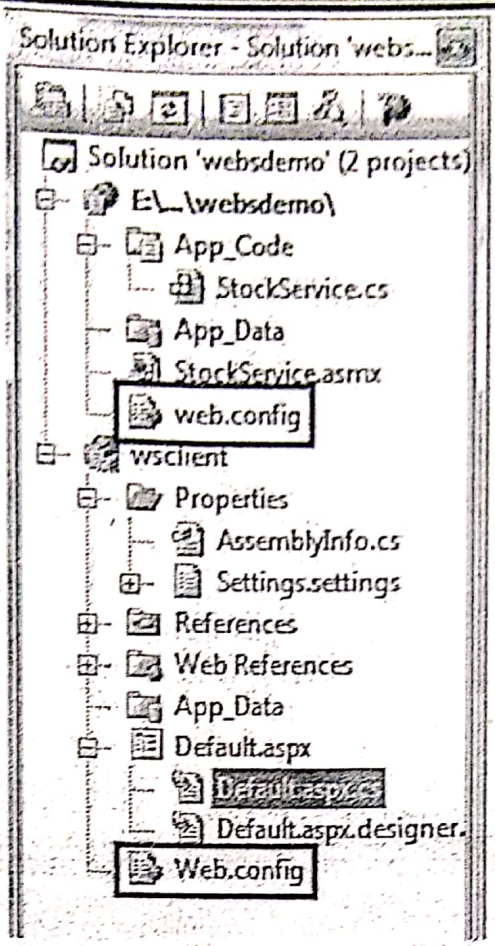
If the application contains child directories, it can define a web.config file for each folder. Scope of each configuration file is determined in a hierarchical top-down manner.

Any web.config file can locally extend, restrict, or override any settings defined on the upper level.

Visual Studio generates a default web.config file for each project. An application can execute without a web.config file, however, you cannot debug an application without a web.config file.

The following figure shows the Solution Explorer for the sample example used in the web services tutorial:

Scanned by CamScanner




In this application, there are two web.config files for two projects i.e., the web service and the web site calling the web service.

The web.config file has the configuration element as the root node. Information inside this element is grouped into **two main areas:** the configuration section-handler **declarationarea**, and the configuration section **settings area.**

The following code snippet shows the basic syntax of a configuration file:

```
<configuration>

<!-- Configuration section-handler declaration area. -->
<configSections>
<section name="section1" type="section1Handler" />
<section name="section2" type="section2Handler" />
</configSections>
<!-- Configuration section settings area. -->

<section1>
<s1Setting1 attribute1="attr1" />
</section1>
```

Scanned by CamScanner

```
<section2>
<s2Setting1 attribute1="attr1" />
</section2>

<system.web>
<authentication mode="Windows" />
</system.web>

</configuration>
```

**Configuration Section Handler declarations**

The configuration section handlers are contained within the <configSections> tags. Each configuration handler specifies name of a configuration section, contained within the file, which provides some configuration data. It has the following basic syntax:

```
<configSections>
<section />
<sectionGroup />
<remove />
<clear/>
</configSections>
```

It has the following elements:

- **Clear** - It removes all references to inherited sections and section groups.
- **Remove** - It removes a reference to an inherited section and section group.
- **Section** - It defines an association between a configuration section handler and a configuration element.
- **Section group** - It defines an association between a configuration section handler and a configuration section.

Scanned by CamScanner

**Application Settings:**

The application settings allow storing application-wide name-value pairs for read-only access. For example, you can define a custom application setting as:

```
<configuration>
<appSettings>
<add key="Application Name" value="MyApplication" />
</appSettings>
</configuration>
```

For example, you can also store the name of a book and its ISBN number:

```
<configuration>
<appSettings>
<add key="appISBN" value="0-273-68726-3" />
<add key="appBook" value="Corporate Finance" />
</appSettings>
</configuration>
```

**Connection Strings:**

The connection strings show which database connection strings are available to the website. For example:

```
<connectionStrings>
<add name="ASPDotNetStepByStepConnectionString"
    connectionString="Provider=Microsoft.Jet.OLEDB.4.0;
    Data Source=E:\\projects\datacaching\ /
    datacaching\App_Data\ASPDotNetStepByStep.mdb"
    providerName="System.Data.OleDb" />

<add name="booksConnectionString"
    connectionString="Provider=Microsoft.Jet.OLEDB.4.0;
    Data Source=C:\ \databinding\App_Data\books.mdb"
    providerName="System.Data.OleDb" />
</connectionStrings>
```

Scanned by CamScanner

## System.Web Element:

The system.web element specifies the root element for the ASP.NET configuration section and contains configuration elements that configure ASP.NET Web applications and control how the applications behave.

It holds most of the configuration elements needed to be adjusted in common applications. The basic syntax for the element is as given:

```
<system.web>
    <authentication>
    <authorization>
    <caching>
    <customErrors>
    <httpCookies>
    <httpHandlers>
    <httpModules>
    <roleManager>
    <securityPolicy>
    <webControls>
</system.web>
```

The following table provides brief description of some of common sub elements of the system.web element:

### Authentication

It configures the authentication support. The basic syntax is as given:

```
<authentication mode="[Windows|Forms|Passport|None]">
<forms>...</forms>
<passport/>
</authentication>
```

### Authorization

It configures the authorization support. The basic syntax is as given:

```
<authorization>
<allow .../>
<deny .../>
</authorization>
```

Scanned by CamScanner

**Caching**

It Configures the cache settings. The basic syntax is as given:

```
<caching>
<cache>...</cache>
<outputCache>...</outputCache>
<outputCacheSettings>...</outputCacheSettings>
<sqlCacheDependency>...</sqlCacheDependency>
</caching>
```

**CustomErrors**

It defines custom error messages. The basic syntax is as given:

```
<customErrors defaultRedirect="url" mode="On|Off|RemoteOnly">
<error. . ./>
</customErrors>
```

**RoleManager**

It configures settings for user roles. The basic syntax is:

```
<roleManager cacheRolesInCookie="true|false" cookieName="name"
  cookiePath="/" cookieProtection="All|Encryption|Validation|None"
  cookieRequireSSL="true|false " cookieSlidingExpiration="true|false "
  cookieTimeout="number of minutes" createPersistentCookie="true|false"
  defaultProvider="provider name" domain="cookie domain">
  enabled="true|false"
  maxCachedResults="maximum number of role names cached"
<providers>...</providers>
</roleManager>
```

**SecurityPolicy**

It configures the security policy. The basic syntax is:

```
<securityPolicy>
<trustLevel />
</securityPolicy>
```

**WebControls**

It provides the name of shared location for client scripts. The basic syntax is:

```
<webControls clientScriptsLocation="String" />
```

Scanned by CamScanner

❖ Basic Controls:

1. Label:
   - **Purpose:**Displays text on the HTML page
   - **Common Properties:**
   - **Common Events:** None
2. TextBox:
   - **Purpose:**Gives the user an input area on an HTML form
   - **Common Properties:**
   - **Common Events:**TextChanged
3. Button:
   - **Purpose:**A normal button control used to respond to click events on the server. You are allowed to pass additional information by setting the CommandName and CommandArguments properties.
   - **Common Properties:**
   - **Common Events:**Click, Command
4. LinkButton:
   - **Purpose:**Like a button in that it posts back to a server, but the button looks like a hyperlink.
   - **Common Properties:**
   - **Common Events:**Click, Command
5. ImageButton:
   - **Purpose:** Can display a graphical image, and when clicked, posts back to the server command information such as the mouse coordinates within the image, when clicked
   - **Common Properties:**
   - **Common Events:**Click
6. Hyperlink:
   - **Purpose:** A normal hyperlink control that responds to a click event
   - **Common Properties:**
   - **Common Events:**None
7. ListBox:
   - **Purpose:** A normal ListBox control like the HTML control, but can be data bound to a data source
   - **Common Properties:**
   - **Common Events:** SelectedIndexChanged

Scanned by CamScanner