

ASP.NET in C#

MD. MAHEDEE HASAN

MAHEDEE.NET

ASP.NET in C#

Md. Mahedee Hasan

Software Architect

Leadsoft Bangladesh Limited

Blog: <http://mahedee.net/>

<http://mahedee.blogspot.com>

Email: mahedee.hasan@gmail.com

Linkedin: <http://www.linkedin.com/in/mahedee>

Facebook: <http://www.facebook.com/mahedee19>

Introduction to .NET Framework	3
Narrow view of .NET Application	4
.NET Technical Architecture	4
Compilation in .NET	5
CLR Execution Model	6
Inspection: Compilation in .NET	6
Introduction to ASP.NET	6
Logical Evolution of ASP	7
ASP.net Web form	7
Web Architecture in ASP.net	7
First Application Using ASP.NET Web form	8
Create a first html page - HtmlPage.html	8
Dynamic Page in ASP .NET	8
ASP.NET - Server Controls	9
ASP.NET - HTML Server Controls	9
HTML Control	10
ASP.NET - Web Server Controls	10
Web Server Control	11
ASP.NET - Validation Server Controls	12
Events in ASP.NET	14
The Page_Load Event	15
The Page.IsPostBack Property	15
HTML Forms	17
ASP.NET Web Forms	17
Submitting a Form	17
ASP.NET Web Forms - Data Binding	19
Data Binding	19
Software Architecture Basics	21
Layer vs. Tier	22
Master Page/ Content Page	22
Master Page	22

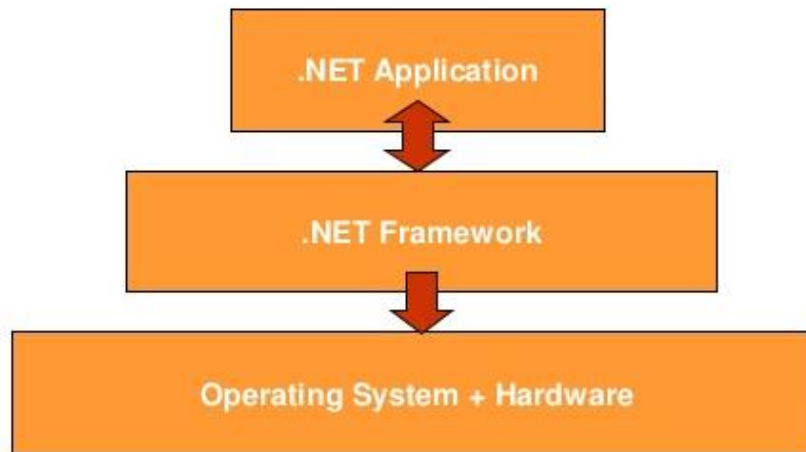
Content Page.....	23
ASP.net Themes	23
ASP.net Provider Model.....	25
References:	26
ASP.net Compilation	27
Website configuration	29
State Management	32
View state	33
Caching.....	38
Global.asax File	40
Culture (Localization and globalization)	41
ASP.net Grid view.....	42
IIS.....	43
ASP.net Page Lifecycle	45
ASP.net Website deployment in IIS	46

Introduction to .NET Framework

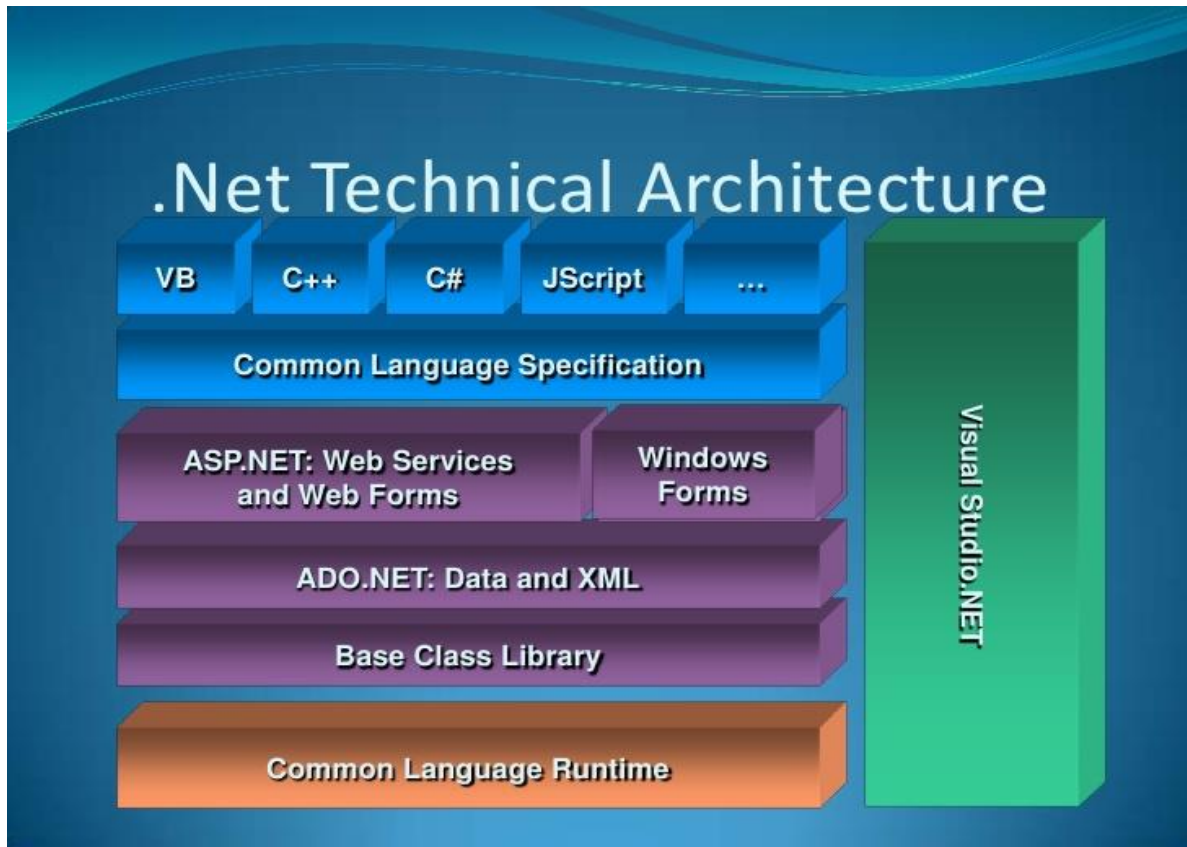
- What is Microsoft .NET?
 - Another Programming Language?
 - No, then what it is?
 - A framework that supports
 - Programming Languages:
 - E.g. C#, Visual Basic, C++, J# etc.
 - Data, Markup Languages:
 - e.g. HTML, XML, CS
 - A **Software Platform**.
 - Language **neutral**.
 - In general
 - Framework is a basic or essential **supporting structure** of a system, concept or text.
- NET framework is for developing **web-based** and **windows-based applications** within the Microsoft environment.

- **ASP.NET** is the **part of Microsoft .NET** framework.
- **Framework Class Library** was designed to make it easier to perform the most common programming tasks.
 - Example: File class, Graphics class, Random class, SmtplibClient class e.t.c

Narrow view of .NET Application

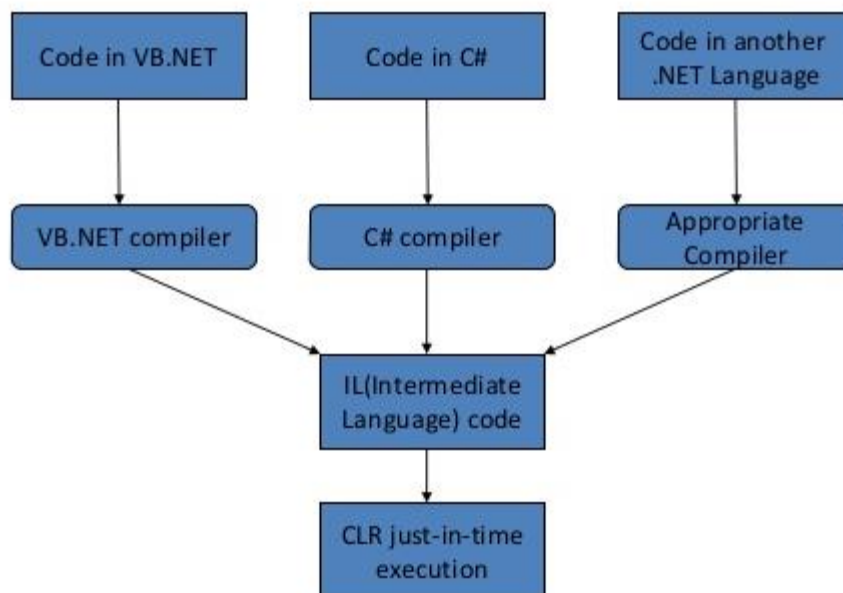


.NET Technical Architecture

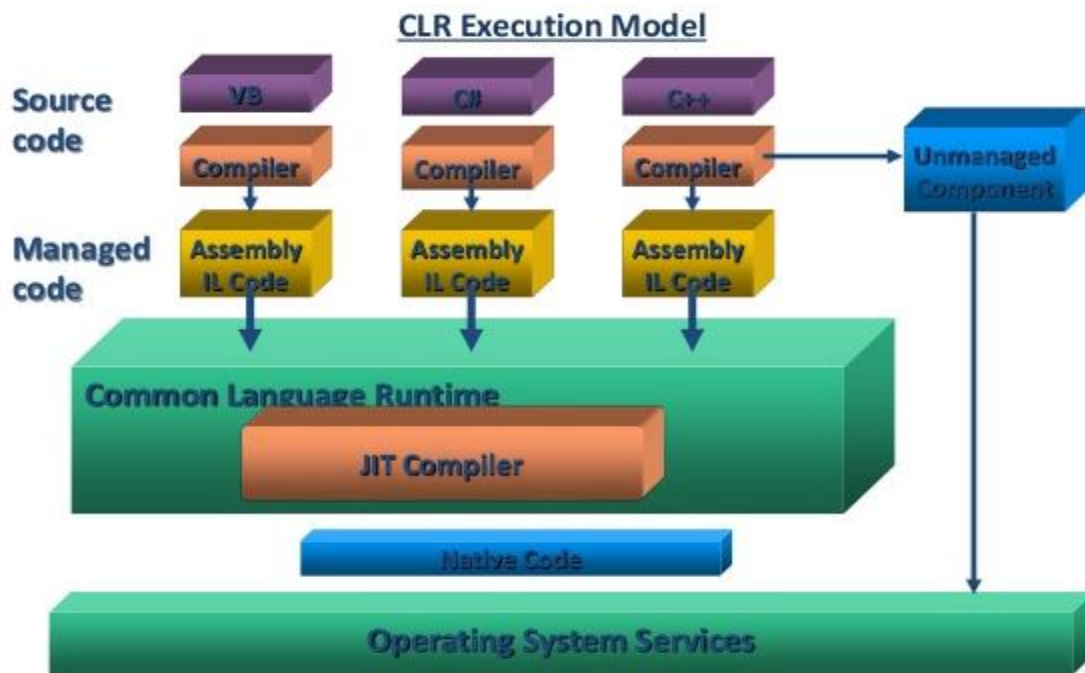


Compilation in .NET

Compilation in .NET



CLR Execution Model



Inspection: Compilation in .NET

- The source code gets compiled by its own Compiler. As C# gets compiled by the c# compiler.
- After the compilation the Intermediate Language Code is generated which is also known as MSIL (Microsoft Intermediate Language) . This code is understandable by the CLR.
- No matter in which language the code is written if the language is .net supported then CLR will start processing the Intermediate Language Code.
- JIT is a part of CLR which executes the assembly. **Just in Time** compiles the code to generate the machine Native Code, which will be processed by the OS.

Introduction to ASP.NET

- ASP - **Active Server Pages**
- Introduced in 1998 as Microsoft's first server side scripting engine.
- ASP.NET is a new ASP generation.
- It is **not compatible** with Classic ASP, but may include Classic ASP.
- ASP.NET pages are compiled, which makes them **faster than Classic ASP**.
- ASP.NET pages have the extension .aspx, and are normally written in VB (Visual Basic) or C# (C sharp).

- ASP.NET, the platform services that allow to program Web Applications and Web Services in any .NET language
- ASP.NET Uses .NET languages to generate HTML pages. HTML page is targeted to the capabilities of the requesting Browser
- ASP.NET “Program” is compiled into a .NET class and cached the first time it is called. All subsequent calls use the cached version.

Logical Evolution of ASP

- Supports **multiple languages**
- Improved **performance**
- **Control-based**, event-driven execution model
- More **productive**
- **Cleanly encapsulated** functionality

ASP.net Web form

- **Allows clean cut code**
 - Code-behind Web Forms
- Easier and **rich** tool
- **Code within is compiled** then executed
- Improved handling of **state information**
- Support for **ASP.NET server controls**
 - Data validation
 - Data bound grids

Web Architecture in ASP.net

- The web application gets **hosted in the server**. The **web client** accepts the request and transmits it to the IIS using the HTTP Protocol.
- HTTP is a transfer protocol for hypertext which is responsible for the transmission of **request and reply**. It uses actions like Get/Put and different Status Codes while transmission. It carries the request to the target server and also traverses back with the reply. Depending on the action and status code the client and server communicates.
- In the server side **IIS accepts the request and process the request**. IIS checks for the resource and if it is a .net resource then IIS asks the .net framework to process the request.

- Once the resource is ready then it is transferred to IIS again and using the HTTP the reply is sent to the client as it uses the HTTP channel so the reply has the content in hypertext.

First Application Using ASP.NET Web form

- ASP.NET Web form
- Empty website of ASP.NET Web form

Create a first html page - HtmlPage.html

Code of HtmlPage.html

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title></title>
</head>
<body>
  <h1>Welcome to mahedee.net</h1>
</body>
</html>
```

The simplest way to convert an HTML page into an ASP.NET page is to copy the HTML file to a new file with an .aspx extension.

Code of HtmlPage.aspx

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title></title>
</head>
<body>
  <h1>Welcome to mahedee.net</h1>
</body>
</html>
```

Dynamic Page in ASP .NET

DynaPage.aspx

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

```
<head>
  <title></title>
</head>
<body>
  <h1><%Response.Write(Now())%></h1>
</body>
</html>
```

ASP.NET - Server Controls

ASP.NET has solved the "spaghetti-code" problem with server controls. Server controls are tags that are understood by the server.

There are **three** kinds of server controls:

- HTML Server Controls - Traditional HTML tags
- Web Server Controls - New ASP.NET tags
- Validation Server Controls - For input validation

ASP.NET - HTML Server Controls

- HTML server controls are HTML tags understood by the server.
- To make HTML elements programmable, add a `runat="server"` attribute to the HTML element. This attribute indicates that the element should be treated as a server control.
- The `id` attribute is added to identify the server control. The `id` reference can be used to manipulate the server control at run time.

Note:

- All HTML server controls must be within a `<form>` tag with the `runat="server"` attribute.
- The `runat="server"` attribute indicates that the form should be processed on the server. It also indicates that the enclosed controls can be accessed by server scripts.

HTMLControl.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="HTMLControl.aspx.cs"
Inherits="App01.HTMLControl1" %>

<!DOCTYPE html>

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

```
<a href="http://mahedee.net" id="link1" runat="server">Welcome to
mahedee.net</a>
<br />
<asp:Label ID="lblMsg" runat="server"></asp:Label>
</div>
</form>
</body>
</html>
```

HTMLControl.aspx.cs

```
public partial class HTMLControl : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        this.lblMsg.Text = "This is a server control";
    }
}
```

HTML Control

- **Html Button** Controls a <button> HTML element
- **Html Form** Controls a <form> HTML element
- **Html Hidden Input** Controls an <input type="hidden"> HTML element
- **Html Table** Controls a <table> HTML element
- **Html Table Cell** Controls <td>and <th> HTML elements
- **Html Table Row** Controls a <tr> HTML element
- **Html Text Area** Controls a <textarea> HTML element

ASP.NET - Web Server Controls

- Web server controls are special ASP.NET tags understood by the server.
- Like HTML server controls, Web server controls are also created on the server and they require a runat="server" attribute to work.
- The syntax for creating a Web server control is:

```
<asp:control_name id="some_id" runat="server" />
```

Example of TextBoxControl.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="TextBoxControl.aspx.cs"
Inherits="App01.TextBoxControl" %>

<!DOCTYPE html>

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

```
</head>
<body>
  <form id="form1" runat="server">
    <div>
      <table>
        <tr>
          <td>
            Name:
          </td>
          <td>
            <asp:TextBox ID="txtFullName" runat="server"></asp:TextBox>
          </td>
        </tr>
        <tr>
          <td></td>
          <td><asp:Button ID="btnSubmit" runat="server" Text="Submit"
OnClick="btnSubmit_Click" /></td>
        </tr>
      </table>
      <asp:Label ID="lblMsg" runat="server"></asp:Label>
    </div>
  </form>
</body>
</html>
```

Example of TextBoxControl.aspx.cs

```
public partial class TextBoxControl : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }

    protected void btnSubmit_Click(object sender, EventArgs e)
    {
        this.lblMsg.Text = this.txtFullName.Text;
    }
}
```

Web Server Control

- AdRotator Displays a sequence of images
- Button Displays a push button
- Calendar Displays a calendar
- CalendarDay A day in a calendar control
- CheckBox Displays a check box
- DropDownList Creates a drop-down list
- HyperLink Creates a hyperlink
- Label Displays static content which is programmable

- LinkButton Creates a hyperlink button
- ListBox Creates a single- or multi-selection drop-down list
- Panel Provides a container for other controls
- Placeholder Reserves space for controls added by code
- RadioButton Creates a radio button
- RadioButtonList Creates a group of radio buttons
- Table Creates a table
- TableCell Creates a table cell
- TableRow Creates a table row
- TextBox Creates a text box

ASP.NET - Validation Server Controls

- Validation server controls are used to validate user-input.
- If the user-input does not pass validation, it will display an error message to the user.
- Each validation control performs a specific type of validation (like validating against a specific value or a range of values).
- By default, page validation is performed when a Button, ImageButton, or LinkButton control is clicked.
- You can prevent validation when a button control is clicked by setting the CausesValidation property to false.

The syntax for creating a Validation server control is:

```
<asp:control_name id="some_id" runat="server" />
```

In the following example we declare one TextBox control, one Button control, and one RangeValidator control in an .aspx file. If validation fails, the text "The value must be from 1 to 100!" will be displayed in the RangeValidator control:

ValidationControl.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="ValidationControl.aspx.cs"
Inherits="App01.ValidationControl" %>

<!DOCTYPE html>

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

```
<form id="form1" runat="server">
  <div>
    <p>
      Enter a number from 1 to 100:
      <asp:TextBox ID="tbox1" runat="server" />
      <br />
      <br />
      <asp:Button Text="Submit" runat="server" />
    </p>

    <p>
      <asp:RangeValidator
        ControlToValidate="tbox1"
        MinimumValue="1"
        MaximumValue="100"
        Type="Integer"
        Text="The value must be from 1 to 100!"
        runat="server" />
    </p>
  </div>
</form>
</body>
</html>
```

Add the following code to web.config

```
<appSettings>
  <add key="ValidationSettings:UnobtrusiveValidationMode" value="None" />
</appSettings>
```

Validation Server Control	Description
CompareValidator	Compares the value of one input control to the value of another input control or to a fixed value
CustomValidator	Allows you to write a method to handle the validation of the value entered
RangeValidator	Checks that the user enters a value that falls between two values
RegularExpressionValidator	Ensures that the value of an input control matches a specified pattern
RequiredFieldValidator	Makes an input control a required field

ValidationSummary

Displays a report of all validation errors occurred in a Web page

Events in ASP.NET

- An **Event Handler** is a **subroutine** that executes code for a given event.

Source Code of TextBoxControl.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="TextBoxControl.aspx.cs"
Inherits="App01.TextBoxControl" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <table>
                <tr>
                    <td>
                        Name:
                    </td>
                    <td>
                        <asp:TextBox ID="txtFullName" runat="server"></asp:TextBox>
                    </td>
                </tr>
                <tr>
                    <td></td>
                    <td><asp:Button ID="btnSubmit" runat="server" Text="Submit"
OnClick="btnSubmit_Click" /></td>
                </tr>
            </table>
            <asp:Label ID="lblMsg" runat="server"></asp:Label>
        </div>
    </form>
</body>
</html>
```

C Sharp code of TextBoxControl.aspx

```
public partial class TextBoxControl : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {

    }

    protected void btnSubmit_Click(object sender, EventArgs e)
```

```
{
    this.lblMsg.Text = this.txtFullName.Text;
}
```

The Page_Load Event

- The **Page_Load** event is one of many events that ASP.NET understands.
- The Page_Load event is triggered when a page loads, and ASP.NET will automatically call the subroutine Page_Load

EventTest.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="EventTest.aspx.cs"
Inherits="App01.EventTest" %>

<!DOCTYPE html>

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

EventTest.aspx.cs

```
public partial class EventTest : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        //this.lblMsg.BackColor = System.Drawing.Color.Red;
        this.lblMsg.ForeColor = System.Drawing.Color.Red;
        this.lblMsg.Text = "Welcome to mahedee.net";
    }
}
```

The Page.IsPostBack Property

- The **Page_Load** subroutine runs every time the page is loaded.

- If you want to execute the code in the **Page_Load** subroutine only the **first** time the page is loaded, you can use the Page.IsPostBack property.
- If the **Page.IsPostBack** property is **false**, the page is loaded for the first time, if it is true, the page is posted back to the server (i.e. from a button click on a form):

Source code for EventTest.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="EventTest.aspx.cs"
Inherits="App01.EventTest" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Label ID="lblMsg" runat="server"></asp:Label>
        </div>
        <asp:Button ID="btnSubmit" Text="Submit" runat="server" OnClick="btnSubmit_Click"
    />
    </form>
</body>
</html>
```

Source code for EventTest.aspx.cs

```
public partial class EventTest : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (Page.IsPostBack)
        {
            //this.lblMsg.BackColor = System.Drawing.Color.Red;
            this.lblMsg.ForeColor = System.Drawing.Color.Green;
        }
        else
        {
            this.lblMsg.ForeColor = System.Drawing.Color.Red;
        }
        this.lblMsg.Text = "Welcome to mahedee.net";
    }

    protected void btnSubmit_Click(object sender, EventArgs e)
    {
        this.lblMsg.Text = "This for button event!";
    }
}
```

HTML Forms

- All server controls must appear within a <form> tag, and the <form> tag must contain the runat="server" attribute.

ASP.NET Web Forms

- All server controls must appear within a <form> tag, and the <form> tag must contain the runat="server" attribute.
- The runat="server" attribute indicates that the form should be processed on the server.
- It also indicates that the enclosed controls can be accessed by server scripts:

```
<form runat="server">
...HTML + server controls
</form>
```

- The form is always submitted to the page itself.
- If you specify an action attribute, it is ignored. If you omit the method attribute, it will be set to method="post" by default. Also, if you do not specify the name and id attributes, they are automatically assigned by ASP.NET.
- An .aspx page can only contain ONE <form runat="server"> control!

If you select view source in an .aspx page containing a form with no name, method, action, or id attribute specified, you will see that ASP.NET has added these attributes to the form. It looks something like this:

```
<form name="_ctl0" method="post" action="page.aspx" id="_ctl0">

...some code

</form>
```

Submitting a Form

- A form is most often submitted by clicking on a button. The Button server control in ASP.NET has the following format:

```
<asp:Button id="id" text="label" OnClick="sub" runat="server" />
```

- The id attribute defines a unique name for the button and the text attribute assigns a label to the button.

- The onClick event handler specifies a named subroutine to execute.

In the following example we declare a Button control in an .aspx file. A button click runs a subroutine which changes the text on the button:

Example

TextBoxControl.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="TextBoxControl.aspx.cs"
Inherits="App01.TextBoxControl" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <table>
                <tr>
                    <td>Name:
                    </td>
                    <td>
                        <asp:TextBox ID="txtFullName" runat="server"></asp:TextBox>
                    </td>
                </tr>
                <tr>
                    <td></td>
                    <td>
                        <asp:Button ID="btnSubmit" runat="server" Text="Submit"
OnClick="btnSubmit_Click" /></td>
                </tr>
            </table>
            <asp:Label ID="lblMsg" runat="server"></asp:Label>
        </div>
    </form>
</body>
</html>
```

TextBoxControl.aspx.cs

```
public partial class TextBoxControl : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {

    }

    protected void btnSubmit_Click(object sender, EventArgs e)
    {
        this.lblMsg.Text = this.txtFullName.Text;
    }
}
```

```
}
```

ASP.NET Web Forms - Data Binding

We may use data binding to fill lists with selectable items from an imported data source, like a database, an XML file, or a script.

Data Binding

The following controls are list controls which support data binding:

- `asp:RadioButtonList`
- `asp:CheckBoxList`
- `asp:DropDownList`
- `asp:ListBox`

The selectable items in each of the above controls are usually defined by one or more `asp:ListItem` controls, like this:

However, with data binding we may use a separate source, like a database, an XML file, or a script to fill the list with selectable items.

By using an imported source, the data is separated from the HTML, and any changes to the items are made in the separate data source.

In the next three chapters, we will describe how to bind data from a scripted data source.

Example: RadioButtonPage.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="RadioButtonPage.aspx.cs"
Inherits="App01.RadioButtonPage" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title></title>
</head>
<body>
  <form id="form1" runat="server">
    <div>
      <asp:RadioButtonList ID="rbtnCity" runat="server">
        <asp:ListItem Value="1" Text="Dhaka" />
      </asp:RadioButtonList>
    </div>
  </form>
</body>
</html>
```

```
        <asp:ListItem Value="2" Text="Chittagong" />
        <asp:ListItem Value="3" Text="Barisal" />
        <asp:ListItem Value="4" Text="Rajshahi" />
    </asp:RadioButtonList>
    <hr />
    <asp:RadioButtonList ID="rbtnDynamic" runat="server">
    </asp:RadioButtonList>
    <hr />
    <asp:RadioButtonList ID="rbtnArrayList" runat="server">
    </asp:RadioButtonList>
    <hr />
    <asp:RadioButtonList ID="rbtnHashTable" runat="server">
    </asp:RadioButtonList>
    <hr />
    <asp:RadioButtonList ID="rbtnXml" runat="server">
    </asp:RadioButtonList>
</div>
</form>
</body>
</html>
```

Example: RadioButtonPage.aspx.cs

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        LoadRadioButton();
        LoadRbtnArray();
        LoadRbtnHashTable();
        LoadrbtnXML();
    }
}

private void LoadRbtnArray()
{
    ArrayList lst = new ArrayList();
    lst.Add("Barisal");
    lst.Add("Noakhali");
    lst.Add("Comilla");
    lst.Add("Chandpur");
    rbtnArrayList.DataSource = lst;
    rbtnArrayList.DataBind();
}

private void LoadRbtnHashTable()
{
    Hashtable htbl = new Hashtable();
    htbl.Add("1", "CSE");
    htbl.Add("2", "EEE");
    htbl.Add("3", "ETE");
    htbl.Add("4", "ICT");
    rbtnHashTable.DataSource = htbl;
    rbtnHashTable.DataValueField = "Key";
}
```

```
        rbtnHashTable.DataTextField = "Value";
        rbtnHashTable.DataBind();
    }

    private void LoadRadioButton()
    {
        this.rbtnDynamic.Items.Add(new ListItem("One", "1"));
        this.rbtnDynamic.Items.Add(new ListItem("Two", "2"));
        this.rbtnDynamic.Items.Add(new ListItem("Three", "3"));
        this.rbtnDynamic.Items.Add(new ListItem("Four", "4"));
        this.rbtnDynamic.Items.Add(new ListItem("Five", "5"));
    }

    private void LoadrbtnXML()
    {
        var countries = new DataSet();
        countries.ReadXml(MapPath("countries.xml"));
        this.rbtnXml.DataSource = countries;
        rbtnXml.DataValueField = "value";
        rbtnXml.DataTextField = "text";
        rbtnXml.DataBind();
    }
}
```

Country.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<countries>

    <country>
        <text>Norway</text>
        <value>N</value>
    </country>

    <country>
        <text>Sweden</text>
        <value>S</value>
    </country>

    <country>
        <text>France</text>
        <value>F</value>
    </country>

    <country>
        <text>Italy</text>
        <value>I</value>
    </country>

</countries>
```

Software Architecture Basics

- Software Architecture is the **structure of the solution** that meets

- All of the **technical and operational requirements**
- Optimizing quality attributes
 - **Performance**
 - **Security**
- Increase **manageability**
- Software architecture is
 - A **blue print** of system and project
 - The **primary carrier** of the **system quality**
 - An **artifact** for early analysis to make sure that the design approach will yield an acceptable system.
 - In short, architecture is the **conceptual glue** that holds every phase of the project together for all its many stakeholders.
 - Manages a collection of discrete objects
 - Implements a set of specific design elements

Layer vs. Tier

- Layer
 - Logical structure of the application
 - Typically layer is the logical organization of the code
 - It cannot deploy as different process or different server
 - Ex. Presentation, business and data layer
- Tier
 - Physical structure of the application
 - It is said that tier is the physical deployment of the layer
 - It can deploy as different process or different server
 - Ex. Presentation, business and data tier

Master Page/ Content Page

Master Page

- A Master page defines the **site-wise layout** and the **regions editable on a content page-by-content page basis**.
- Firstly introduce in ASP.NET 2.0 on VS 2005. And in VS 2008 offers design-time support for nested master pages.
- Special **@Master directive**

```
<%@ Master Language="C#" CodeFile="MasterPage.master.cs" Inherits="MasterPage" %>
```

Content Page

- With the master page created, we are ready to start creating ASP.NET pages that are bound to the master page. Such pages are referred to as content pages.

```
<asp:contentplaceholder id="Main" runat="server" />
```

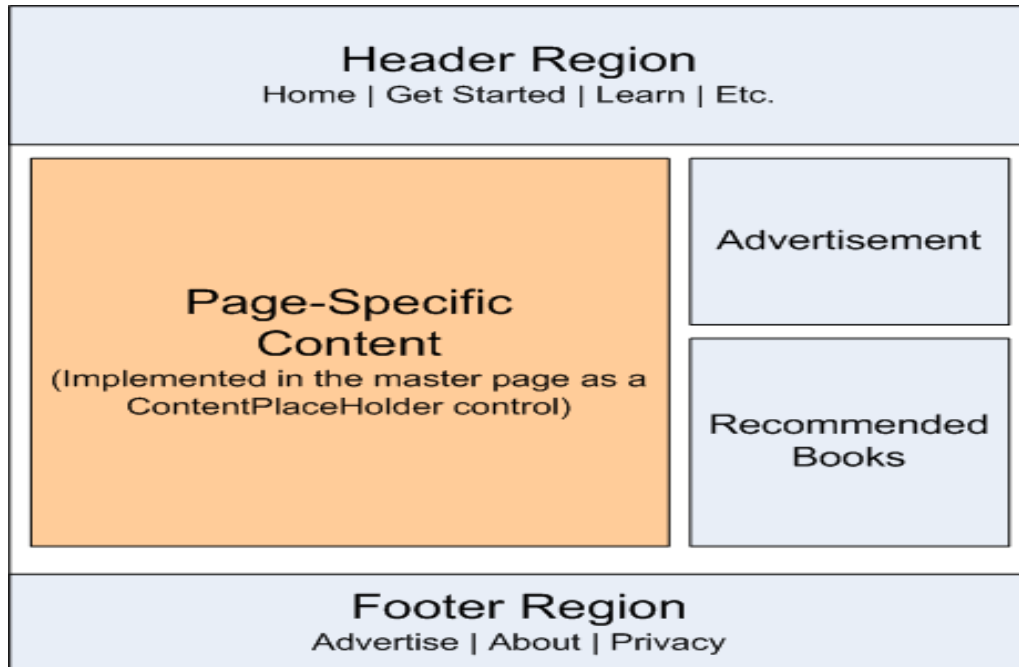
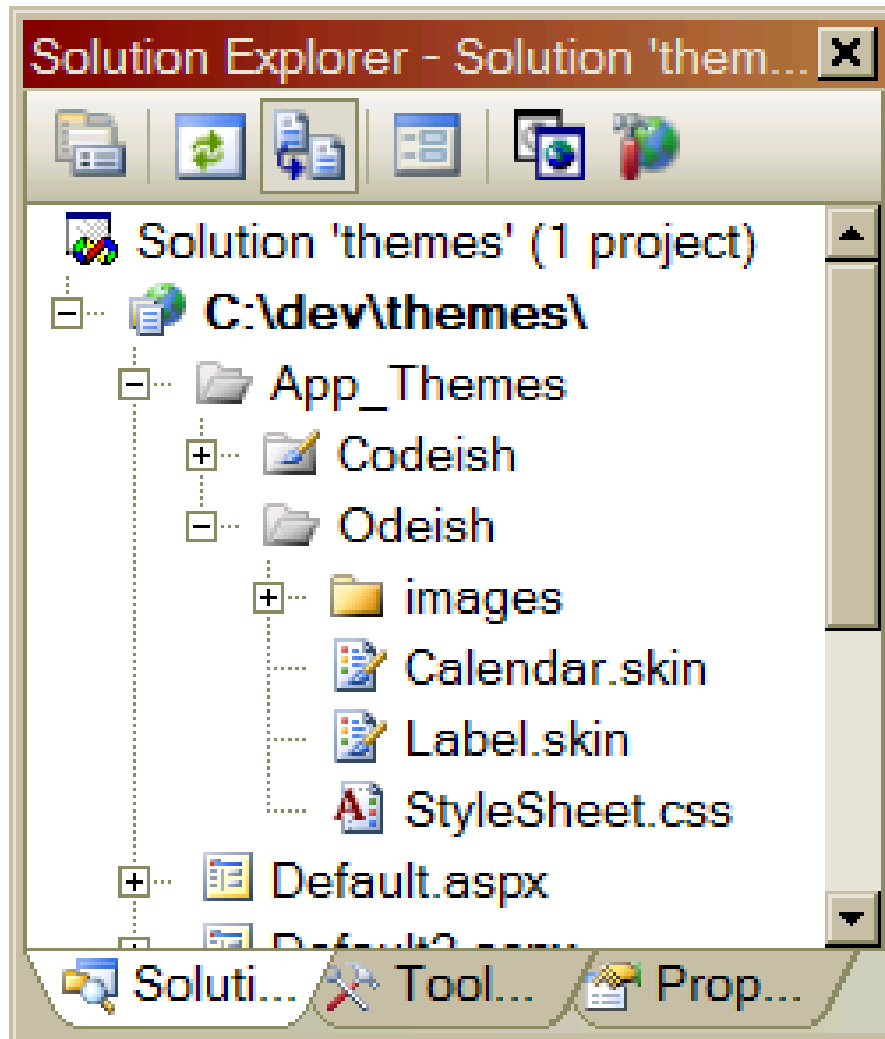


Fig – Sample Master Page

ASP.net Themes

- Themes is a cool and easy way to create a consistent look and feel across a page or an entire website
- By using themes, you can easily customize your server controls with predefined looks
- The themes feature allows to dictate the appearance of controls in your application using template files with a .skin extension, and with style sheets
- A .skin file can contain one or more control skins for one or more control types.



- The most important types of files in a Theme folder are the following:
 - Skin files
 - Cascading Style Sheet file
- Adding Skins to theme (Default)
 - You must always include a Runat attribute, but you can never include the ID attribute when declaring a control in a Skin

```
<asp:TextBox BackColor="Yellow" BorderStyle="Dotted" Runat="Server" />
```

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Site.master" AutoEventWireup="true"  
CodeFile="Default.aspx.cs" Inherits="_Default" Theme="Textbox" %>
```

- Named skin

- When you create a Named Skin, you can decide when you want to apply the Skin.

```
<asp:TextBox SkinID="DashedTextBox" BorderStyle="Dashed" BorderWidth="5px" Runat="Server" />
```

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

```
<asp:TextBox id="txtFirstName" SkinID="DashedTextBox" Runat="server" />
```

- Themes Versus StyleSheetThemes

- You might want to display every Label in your web site with an orange back-

Ground color except for one Label. In that case, it would be nice if there was a way to override the Skin property.

```
<asp:Label BackColor="Orange" Runat="Server" />
```

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

```
<asp:Label id="Label1" Text="What color background do I have?" BackColor="red" Runat="server" />
```

- Disabling Themes

```
<asp:Calendar id="Calendar2" EnableTheming="false" Runat="server" />
```

- Adding Cascading Style Sheets to Themes

```
h1 {
```

```
    color: red;
```

```
    background-color: gray;
```

```
}
```

```
<h1>.:Cascading Style Sheet as a theme:.</h1>
```

ASP.net Provider Model

- A provider is a **software module** that provides a uniform interface between a service and a data source.

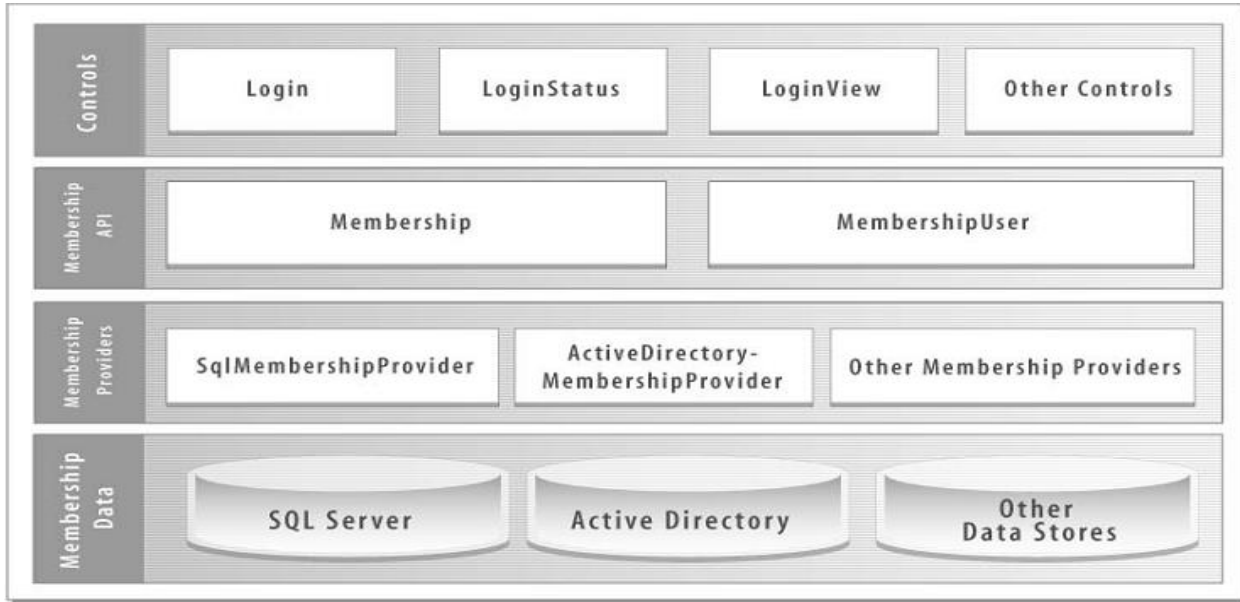


Fig – The Membership provider

- Membership
 - System.Web.Security.ActiveDirectoryMembershipProvider
 - System.Web.Security.SqlMembershipProvider
- Role management
 - System.Web.Security.AuthorizationStoreRoleProvider
 - System.Web.Security.SqlRoleProvider
 - System.Web.Security.WindowsTokenRoleProvider
- Site map
 - System.Web.XmlSiteMapProvider
- Profile
 - System.Web.Profile.SqlProfileProvider
- Session state
 - System.Web.SessionState.InProcSessionStateStore
 - System.Web.SessionState.OutOfProcSessionStateStore
 - System.Web.SessionState.SqlSessionStateStore
- Web events
 - System.Web.Management.EventLogWebEventProvider
 - System.Web.Management.SimpleMailWebEventProvider
 - System.Web.Management.TemplatedMailWebEventProvider
 - System.Web.Management.SqlWebEventProvider
 - System.Web.Management.TraceWebEventProvider
 - System.Web.Management.WmiWebEventProvider
- Web Parts personalization
 - System.Web.UI.WebControls.WebParts.SqlPersonalizationProvider
- Protected configuration

- System.Configuration.DPAPIProtectedConfigurationProvider
- System.Configuration.RSAProtectedConfigurationProvider

ASP.net Compilation

- ASP.net first compiles the code into one or more assemblies in order to service request by the user.
- ASP.net code can be write in many different languages – C#, Visual basic, J#, others.
- Code is compiled to MSIL
 - MSIL is language and CPU independent
- At run time, MSIL runs in the context of the .NET Framework, which translates MSIL into CPU-specific instructions for the processor
- Benefits of compiling application code
 - Performance
 - Compiled code is much faster than scripting code
 - Security
 - Compiled code is difficult to reverse engineering
 - Stability
 - Code is checked at compile time for **syntax errors, type safety**, and other problems
 - Interoperability
 - MSIL code is supported by .NET, you can use assemblies that were originally written in other languages in your code
- ASP.net compilation architecture features
 - Multiple language supports
 - 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

- Flexible deployment
 - ASP.NET compiler tool (ASPNET_Compiler.exe) provides the following pre compilation options:
 - In place compilation
 - Non-updateable full pre compilation
 - Updateable pre compilation
- Extensible build system
 - Extend and customize the ASP.NET build system to compile custom resources by creating classes that inherit from the BuildProvider class
- Dynamic compilation
 - ASP.NET dynamic compilation enables you to modify your source code without having to explicitly compile your code before you deploy your Web application
 - Compiling on first request
 - Recompiling on change
 - Compilation dependencies
 - Compilation output
 - Compilation folder output
 - **%SystemRoot%\Microsoft.NET\Framework\versionNumber\Temporary ASP.NET Files**
 - Compilation Folder Required Permissions
 - Compilation Folder Configurability
 - %FrameworkInstallLocation%\Temporary ASP.NET Files
 - Support multiple languages
- Disadvantages of dynamic compilation
 - Slower
 - Pages and code files must be compiled the first time they are requested
 - Dynamic compilation does not offer a means to identify compile-time bugs

- Does not provide the ability to create a compiled version

Website configuration

- ASP.net uses a hierarchical system of configuration
- At the top of the hierarchy is Machine.config file
 - C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\CONFIG\Machine.config

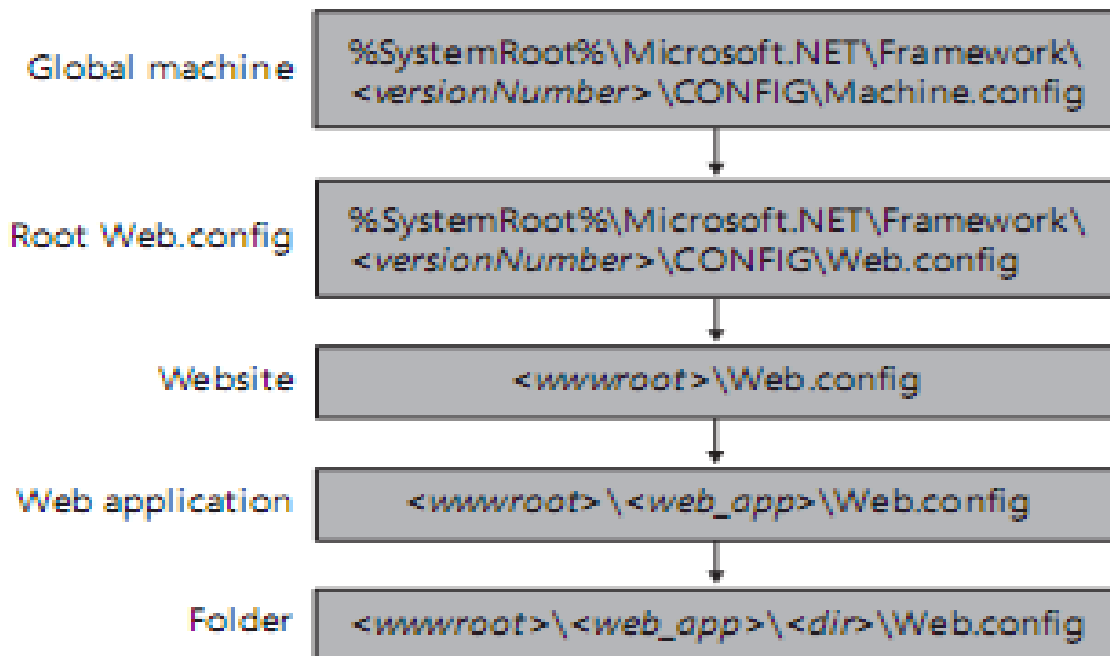


FIGURE 1-7 The configuration file hierarchy.

- The \CONFIG folder includes the following six files:
 - Machine.config—Contains the actual configuration settings.
 - Machine.config.default—Contains the default values for all configuration settings.
 - Machine.config.comments—Contains comments on each configuration setting.
 - Web.config—Contains the actual configuration settings.
 - Web.config.default—Contains the default values for all configuration settings.
 - Web.config.comments—Contains comments on each configuration setting


```
<system.web>
  <!--<httpRuntime executionTimeout="60"/>-->
  <httpRuntime executionTimeout="6000" enableKernelOutputCache="false"/>
  <compilation debug="false" targetFramework="4.5">
    <assemblies>
      <add assembly="CrystalDecisions.ReportAppServer.ClientDoc, Version=13.0.2000.0, Culture=neutral, PublicKeyToken=692F8EA5521E1304"/>
    </assemblies>
  </compilation>
  <httpRuntime targetFramework="4.5"/>
</system.web>
```

```
<customErrors defaultRedirect="Error.aspx" mode="Off">
  <error statusCode="401" redirect="Unauthorized.aspx"/>
</customErrors>
```

Elements of Web.config file

Connection string settings

```
<connectionStrings>
  <add name="SQLServerConnectionString" connectionString="Data Source=192.168.20.67;Initial Catalog=Security08_DBL;User ID=sa;Password=leads@123" providerName="System.Data.SqlClient" />
</connectionStrings>
```

App Settings

```
<appSettings>
  <add key="Company" value="LEADS Corporation Limited"/>
</appSettings>
```

Session State Settings

```
<sessionState mode="InProc" />
<!--Name of the server: BluechipServer-->
<sessionState mode="StateServer" stateConnectionString="tcpip=BluechipServer:42424" />
<!--Connection String Name: SQLServerConnectionString-->
<sessionState mode="SQLServer" sqlConnectionString="SQLServerConnectionString" />
```

Programmatically accessing web.config file

- Reading appSettings values

```
string key = ConfigurationManager.AppSettings["AppKey"];
```


- Reading connectionstring values

```
string cnn = ConfigurationManager.ConnectionStrings["conn"].ConnectionString;
```

- Update the configuration section values

```
Configuration config = WebConfigurationManager.OpenWebConfiguration("~/");
```

```
//Get the required section of the web.config file by using configuration object.
```

```
CompilationSection compilation =
```

```
(CompilationSection)config.GetSection("system.web/compilation");
```

```
//Update the new values.
```

```
compilation.Debug = true;
```

```
config.Save();
```

State Management

- The State or Cache Management is nothing but the way to storing the data in Client-Side and in Server-Side using small memory.
- There are two ways to manage state:
 - Client Side State management
 - Stores information on the client's computer
 - Server side state management
 - Store information on application server
 - Higher security than client-side options
 - Used more web server resources
- Client side state management
 - View State
 - Hidden fields

- Cookies
- Query Strings
- Control State
- Server side state management
 - Application state
 - Session state

View state

- It stores data in the generated HTML using hidden field not on the server.
- View State provides page level state management
- As long as the user is on the current page, state is available and the user redirects to the next page and the current page state is lost
- View State can store any type of data because it is object type but it is preferable not to store a complex type of data due to the need for serialization and deserialization on each post back
- View state...

```
protected void Page_Load(object sender, EventArgs e)
{
    if (IsPostBack)
    {
        if (ViewState["count"] != null)
        {
            int ViewstateVal = Convert.ToInt32(ViewState["count"]) + 1;
            this.txtViewState.Text = ViewstateVal.ToString();
            ViewState["count"] = ViewstateVal.ToString();
        }
        else
        {
            ViewState["count"] = "1";
        }
    }
}

protected void Button1_Click(object sender, EventArgs e)
{
    this.txtViewState.Text = ViewState["count"].ToString();
}
```

Hidden Fields

- Hidden fields are used to store data at the page level
- Hidden fields are not rendered by the browser.
- It's just like a standard control for which you can set its properties
- Advantages:
 - Simple to implement for a page specific data
 - Can store small amount of data so they take less size.
- Disadvantages:
 - Inappropriate for sensitive data
 - Hidden field values can be intercepted (clearly visible) when passed over a network

Hidden Fields...

```
<asp:HiddenField ID="HiddenField1" runat="server" />
```

```
protected void Page_Load(object sender, EventArgs e)
```

```
{  
    if (HiddenField1.Value != null)  
    {  
        int val= Convert.ToInt32(HiddenField1.Value) + 1;  
        HiddenField1.Value = val.ToString();  
        Label1.Text = val.ToString();  
    }  
}
```

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

```
{  
  
}
```

Cookies

- A cookie is a small piece of text stored on user's computer

- Every time a user visits a website, cookies are retrieved from user machine and help identify the user
- Advantages:
 - Simplicity
- Disadvantages:
 - Cookies can be disabled on user browsers
 - Cookies are transmitted for each HTTP request/response causing overhead on bandwidth
 - Inappropriate for sensitive data

Cookies..

- **Persistence Cookie:** Cookies which you can set an expiry date time are called persistence cookies. Persistence cookies are permanently stored till the time you set.

```
Response.Cookies["nameWithPCookies"].Value = "This is A Persistence Cookie";
```

```
Response.Cookies["nameWithPCookies"].Expires = DateTime.Now.AddSeconds(10);
```

And the second one is:

```
HttpCookie aCookieValPer = new HttpCookie("Persistence");
```

```
aCookieValPer.Value = "This is A Persistence Cookie";
```

```
aCookieValPer.Expires = DateTime.Now.AddSeconds(10);
```

```
Response.Cookies.Add(aCookieValPer);
```

Cookies...

- **Non-Persistence Cookie:** Non persistence cookies are not permanently stored on the user client hard disk folder. It maintains user information as long as the user accesses the same browser. When user closes the browser the cookie will be discarded.

```
Response.Cookies["nameWithNPCookies"].Value = "This is A Non Persistence Cookie";
```

And the second way is:

```
HttpCookie aCookieValNonPer = new HttpCookie("NonPersistence");
```

```
aCookieValNonPer.Value = "This is A Non Persistence Cookie";  
Response.Cookies.Add(aCookieValNonPer);
```

How to read a cookie:

```
if (Request.Cookies["NonPersistence"] != null)  
  
    Label2.Text = Request.Cookies["NonPersistence"].Value;
```

QueryString

- Query strings are usually used to send information from one page to another page.
- They are passed along with URL in clear text.
- **Advantages:**
 - Simple to Implement
- **Disadvantages:**
 - Human Readable
 - Client browser limit on URL length
 - Cross paging functionality makes it redundant
 - Easily modified by end user

<http://www.productdetails.aspx?productid=4>

```
string v = Request.QueryString["productid"];
```

Application State

- Application object is used to store data which is visible across entire application
- Shared across multiple user sessions.
- Data which needs to be persisted for entire life of application should be stored in application object.

```
Application.Lock();
```

```
Application["mydata"]="mydata";
```

```
Application.Unlock();
```

Session Object

- Session object is used to store state specific information per client basis.
- It is specific to particular user.
- Session data persists for the duration of user session you can store session's data on web server in different ways.

Session["userName"] = "mahedee";//Set Value to The Session

Label2.Text = Session["userName"].ToString(); //Get Value from the Session

Session State Mode

- There are four session storage mechanisms provided by ASP.NET
 - InProc mode
 - StateServer mode
 - SQLServer mode
 - Custom mode
- InProc Mode
 - InProc mode is the default mode provided by ASP.NET
 - In this mode, session values are stored in the web server's memory (in IIS).

<sessionstate mode="InProc" cookieless="false" timeout="10" />

- **StateServer mode**

- This mode could store session in the web server but out of the application pool.
- Usually if this mode is used there will be a separate server for storing sessions, i.e., stateServer.
- The benefit is that when IIS restarts, the session is available.
- It stores session in a separate Windows service.
- For State server session mode, we have to configure it explicitly in the web config file and start the aspnet_state service.

<sessionstate mode="stateserver" cookieless="false"

timeout="10" stateConnectionString="tcpip=127.0.0.1:42424"/>

- **SQLServer mode**
- Session is stored in a SQL Server database.
- This kind of session mode is also separate from IIS, i.e., session is available even after restarting the IIS server.
- This mode is highly secure and reliable but also has a disadvantage that there is overhead from **serialization and deserialization** of session data.
- This mode should be used when reliability is more important than performance.

```
<sessionstate mode="sqlserver" cookieless="false" timeout="10"
```

```
stateConnectionString="tcpip=127.0.0.1:4 2424"
```

```
sqlConnectionString="Data Source=.\SqlDataSource;User ID=userid;Password=password"/>
```

- Custom mode
 - Generally we should prefer InProc state server mode or SQL Server mode
 - If you need to store session data using other than these techniques then ASP.NET provides a custom session mode.
 - This way we have to maintain everything customized even generating session ID, data store, and also security.

Caching

- Caching is a technique of storing data in memory which takes time to create
- There are two main types of caching:
 - Output caching
 - Cache the final render HTML of the page.
 - When the same page is requested again, the cached page is served
 - ASP.NET does not start the page life cycle and does not execute the code

```
<%@ OutputCache Duration="60" VaryByParam="None"%>
```

- Data caching
 - Flexible way to cache a DataSet object or any collection object

```
date1 = DateTime.Now;
```

```
Cache.Insert("Date1", date1, null, DateTime.Now.AddSeconds(20), TimeSpan.Zero);
```

```
Or DataSet ds = new DataSet();
```

```
ds = (DataSet) Cache["MyDataSet"];
```

- Valid parameter for output caching
 - VaryByParam
 - <%@ OutputCache Duration="60" VaryByParam="CategoryID" %>
 - <%@ OutputCache Duration="60" VaryByParam="id;langid" %>
 - <%@ OutputCache Duration="60" VaryByParam="*" %>
 - VaryByControl
 - <%@ OutputCache Duration="20" VaryByControl="MyControlId" %>
 - VaryByCustom
 - <%@ OutputCache Duration="60" VaryByCustom="browser" %>
- Cache Profile
 - Provides a facility to define cache settings in the web.config

```
<system.web>
```

```
<caching>
```

```
<outputCacheSettings >
```

```
<outputCacheProfiles>
```

```
<add name ="ProductItemCacheProfile" duration ="60"/>
```

```
</outputCacheProfiles>
```

```
</outputCacheSettings>
```

```
</caching>
```


</system.web>

<%@ OutputCache CacheProfile ="ProductItemCacheProfile" VaryByParam="None" %>

Global.asax File

- The Global.asax, also known as the ASP.NET application file
- It is used to serve application-level and session-level events.
- 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 **HttpApplication** base 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.
-

```
<%@ Application Language="C#" %>
<script runat="server">

    void Application_Start(object sender, EventArgs e)
    {
        // Code that runs on application startup
    }

    void Application_End(object sender, EventArgs e)
    {
        // Code that runs on application shutdown
    }

    void Application_Error(object sender, EventArgs e)
    {
        // Code that runs when an unhandled error occurs
    }

    void Session_Start(object sender, EventArgs e)
    {
        // Code that runs when a new session is started
    }

    void Session_End(object sender, EventArgs e)
    {
        // Code that runs when a session ends.
        // Note: The Session_End event is raised only when the sessionstate mode
        // is set to InProc in the Web.config file. If session mode is set to StateServer
        // or SQLServer, the event is not raised.
    }

</script>
```

Culture (Localization and globalization)

- Localization is the process of adapting a software application for a specific locale.
- Globalization is the process of identifying the localizable resources of the application.
- To support Localization and Globalization, we use
 - System.Globalization - can define culture specific information
 - System.Resources - ResourceManager Class
 - System.Threading namespaces - supports for multithreaded programming

- Has two culture values
 - Culture
 - UICulture
- Culture value is used for date and number formatting
- UICulture values are used to determine culture specific resources
- Can set culture and UICulture values in the application as follows.

Using <globalization> element of Web.Config.

Using @Page directive in the Page.

e.g.

```
Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture ("en-GB");
```

```
Thread.CurrentThread.CurrentUICulture=new CultureInfo("en-GB");
```

- Two main properties of Page class have an effect on localization:
 - UICulture
 - Used to specify which resource files are loaded for the page
 - Culture
 - Determines how strings such as dates, numerals, and currency amounts are formatted
- Setting culture manually

```
<%@ Page Language="C#" Culture="bn-BD" UICulture="bn-BD" AutoEventWireup="true"
CodeFile="ManualCulture.aspx.cs" Inherits="ManualCulture" %>
```

```
this.lblMsg.Text = DateTime.Today.ToString();
```

- Setting the Culture in the Web Configuration File

```
<globalization culture="en-US" uiCulture="en-US"/>
```

- Creating local resource

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Site.master" AutoEventWireup="true"
CodeFile="LocalizablePage.aspx.cs" UICulture="auto" Inherits="LocalizablePage" %>
```

```
<asp:Button id="btnSubmit" meta:resourceKey="btnSubmit" Text="Click Me!" ToolTip="Click to show
message" OnClick="btnSubmit_Click" Runat="server" />
```











```
<asp:Label id="lblMessage" meta:resourceKey="lblMessage" Text="Thank You!" Visible="false"
Runat="server" />
```

LISTING 26.12 App_LocalResources\LocalizablePage.aspx.es.resx

Name	Value
ClickHere	chasque aquí
ThankYou	¡Gracias!

ASP.net Grid view

- Powerful server side control
- Excellent data binding architecture
- Including Sorting, paging, deleting option

Full Name	Nick Name	Designation	Edit	Delete
Md. Mahedee Hasan	Mahedee	Senior Software Engineer		
Md. Asrafuzzaman	Emon	Senior Software Engineer		
Md. Khondakar Enamul Haque	Rony	Broadcast Engineer		
Md. Jakir Hossain	Jakir	Senior Accountants		
Md. Safiul Bashar	Sonnet	Software Engineer		

IIS

- IIS stands for Internet Information Server
- IIS is web server application and set of feature extension modules
- A Web server is primarily a server application that can be contacted using a bunch of Internet protocols, such as
 - HTTP
 - File Transfer Protocol (FTP)
 - Simple Mail Transfer Protocol (SMTP)
 - Developed by Microsoft
- HTTP Request
 - When the client request for the resource using HTTP Pipeline.
 - Example

GET /index.html HTTP/1.1

Host: www.example.com

- HTTP Response
 - The response from the server against a request
 - Example

HTTP/1.1 200 OK

Date: Mon, 23 May 2005 22:38:34 GMT

Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)

Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT

Content-Type: text/html; charset=UTF-8

Processing server page

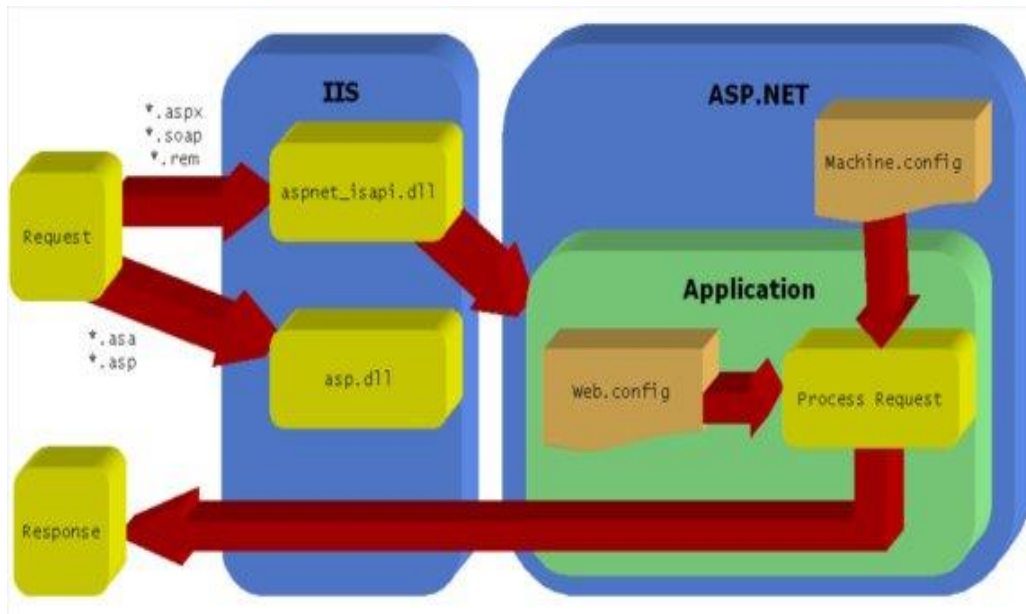


Fig – Processing server page

- Processing server page
 - Request is sent by the web browser through the HTTP Protocol.
 - The request hits the IIS where the request has the resource location.
 - HTTP carries the request in a hyper text format. When IIS gets the request then it checks in the **isapi extension to map the worker process**.
 - Inetinfo.exe at the IIS is responsible for the process. aspnet_isapi.dll is mapped for the asp. net application. Then according to the mapping table if it is an aspx page then **aspnet_wp.exe** handles the request from asp. net.
 - It checks for the resource and after finding the framework executes the assembly and generates the html page for the client.
 - IIS then again takes the response from .NET and passes it via HTTP protocol.

- Application pool
 - Concept of application pool is introduced in IIS 6.0
 - Application pools are used to separate sets of IIS worker processes
 - Application pools used to isolate our web application for better security, reliability, and availability and performance
 - Keep running without impacting each other
 - Isolation of Different Web Application
 - Individual worker process for different web application
 - More reliably web application
 - Better Performance

ASP.net Page Lifecycle

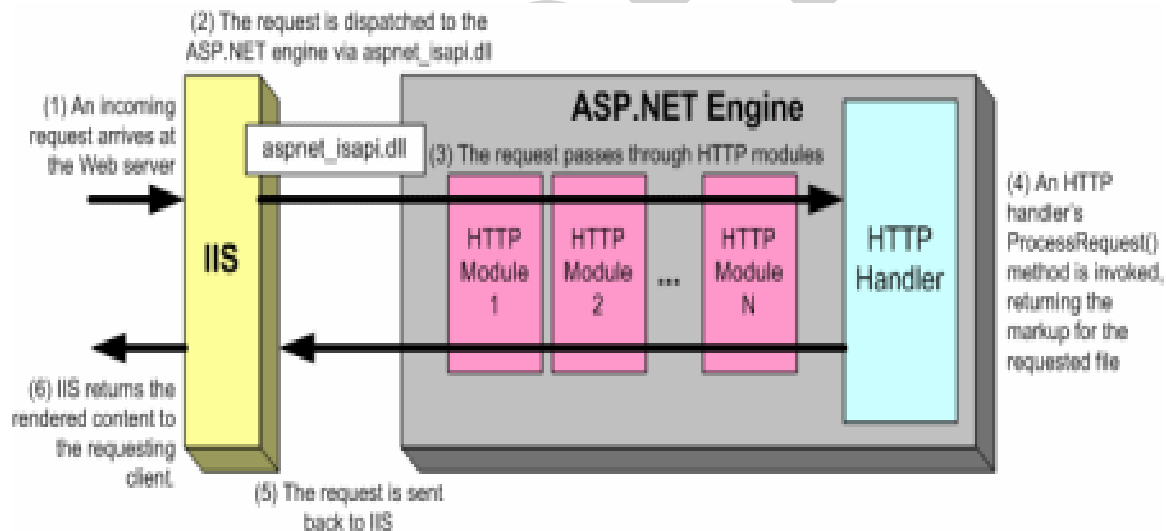


Fig – ASP.net page lifecycle

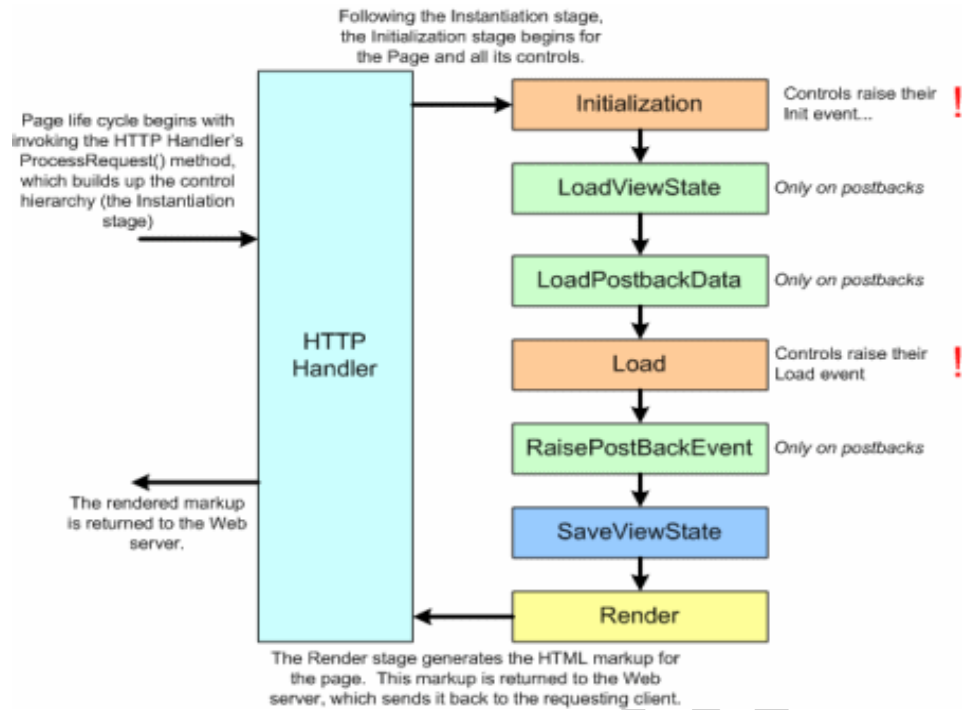


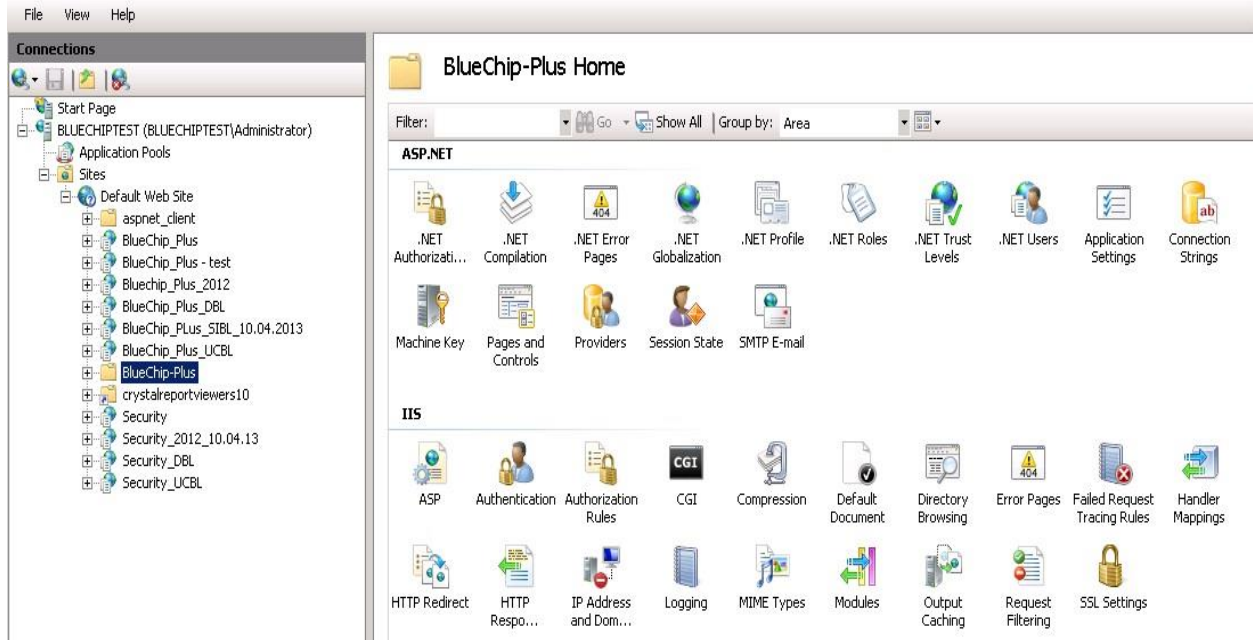
Fig – Events in the page lifecycle

Table Showing Stage and Corresponding Events

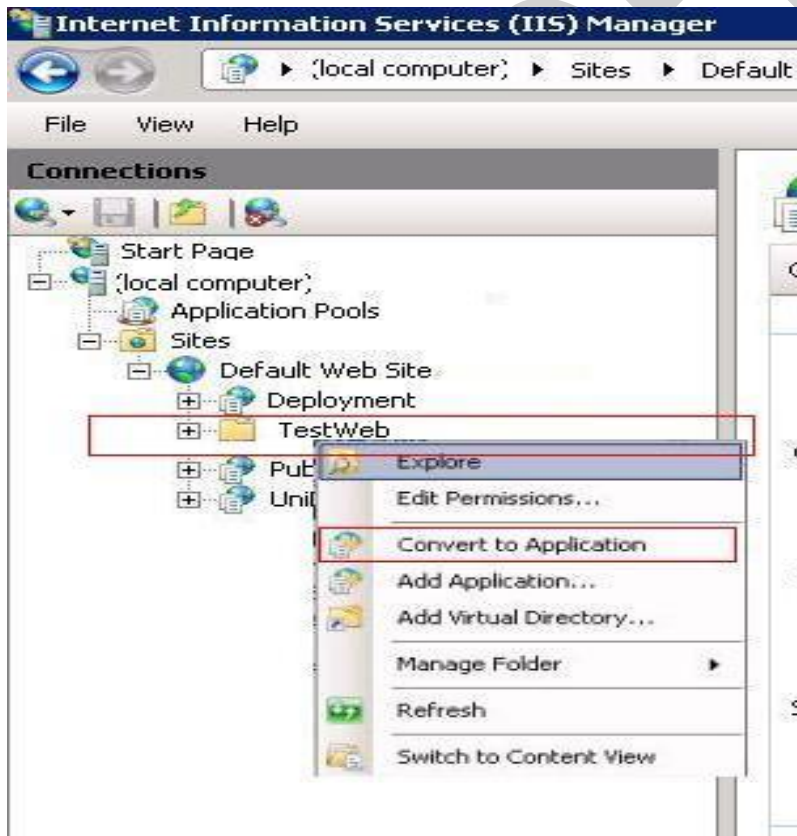
Stage	Events/Method
Initialization of the page	Page_Init
Loading of the View State	LoadViewState
Processing of the Postback data	LoadPostData
Loading of Page	Page_Load
Notification of PostBack	RaisePostDataChangedEvent
Handling of PostBack Event	RaisePostBackEvent
Pre Rendering of Page	Page_PreRender
Saving of view state	SaveViewState
Rendering of Page	Page_Render
Unloading of the Page	Page_UnLoad

ASP.net Website deployment in IIS

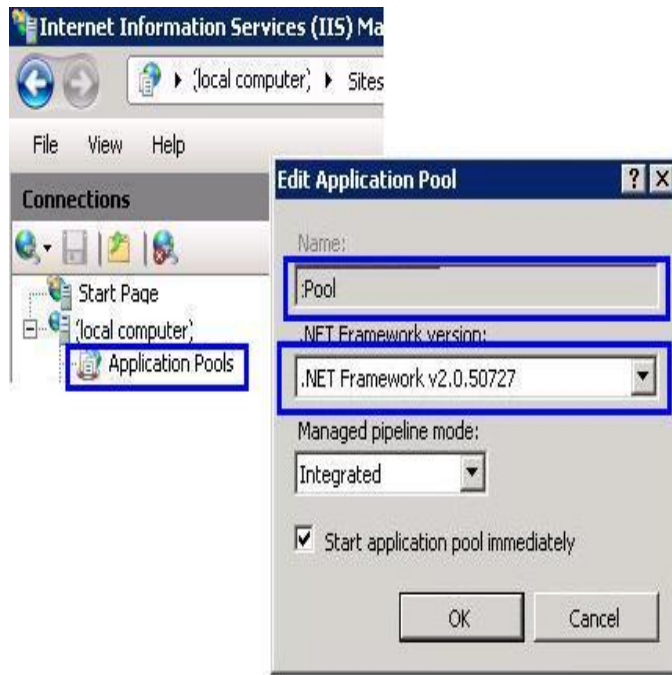
ASP.NET in C#



Convert to Web Application



Create application pool



Assign application pool to the application

