HTTP is a stateless protocol. ASP.NET framework helps in storing the information regarding the state of the application, which consists of:

Page state

Session state

When a user requests an ASP.NET page, the IIS delegates the processing of the page to the ASP.NET runtime system.

The ASP.NET runtime transforms the .aspx page into an instance of a class, which inherits from the base class page of the .Net framework. Therefore, each ASP.NET page is an object and all its components

The ASP.NET life cycle could be divided into two groups:

Application Life Cycle

Page Life Cycle

ASP.NET Application Life Cycle

The application life cycle has the following stages:

* User makes a request for accessing application resource, a page. Browser sends this request to the web server.
* A unified pipeline receives the first request and the following events take place:
  + An object of the class ApplicationManager is created.
  + An object of the class HostingEnvironment is created to provide information regarding the resources.
  + Top level items in the application are compiled.
* Response objects are created. The application objects such as HttpContext, HttpRequest and HttpResponse are created and initialized.
* An instance of the HttpApplication object is created and assigned to the request.
* The request is processed by the HttpApplication class. Different events are raised by this class for processing the request.

An ASP.NET page is also a server side file saved with the .aspx extension. It is modular in nature and can be divided into the following core sections:

* Page Directives
* Code Section
* Page Layout

Some events cause the form to be posted back to the server immediately, these are called the postback events. For example, the click event such as, Button.Click.

Some events are not posted back to the server immediately, these are called non-postback events.

For example, the change events or selection events such as TextBox.TextChanged or CheckBox.CheckedChanged. The nonpostback events could be made to post back immediately by setting their AutoPostBack property to true.

Server Side

# ASP.NET - Server Side

We have studied the page life cycle and how a page contains various controls. The page itself is instantiated as a control object. All web forms are basically instances of the ASP.NET Page class. The page class has the following extremely useful properties that correspond to intrinsic objects:

* Session
* Application
* Cache
* Request
* Response
* Server
* User
* Trace

HttpContext.Current.Server.UrlEncode

HttpContext.Current.Server.HtmlEncode

## Request Object

The request object is an instance of the System.Web.HttpRequest class. It represents the values and properties of the HTTP request that makes the page loading into the browser.

## Response Object

The Response object represents the server's response to the client request. It is an instance of the System.Web.HttpResponse class.

The Response.Redirect() method allows transferring the user to another page, inside as well as outside the application. It requires a round trip.

# ASP.NET - HTML Server

The HTML server controls are basically the standard HTML controls enhanced to enable server side processing.

They are specifically converted to a server control by adding the attribute runat="server" and adding an id attribute to make them available for server-side processing.

# ASP.NET - Client Side

ASP.NET client side coding has two aspects:

* **Client side scripts** : It runs on the browser and in turn speeds up the execution of page. For example, client side data validation which can catch invalid data and warn the user accordingly without making a round trip to the server.
* **Client side source code** : ASP.NET pages generate this. For example, the HTML source code of an ASP.NET page contains a number of hidden fields and automatically injected blocks of JavaScript code, which keeps information like view state or does other jobs to make the page work.

**Client Side Source Code**

We have already discussed that, ASP.NET pages are generally written in two files:

* The content file or the markup file ( .aspx)
* The code-behind file

The content file contains the HTML or ASP.NET control tags and literals to form the structure of the page. The code behind file contains the class definition. At run-time, the content file is parsed and transformed into a page class.

# ASP.NET - Directives

ASP.NET directives are instructions to specify optional settings, such as registering a custom control and page language. These settings describe how the web forms (.aspx) or user controls (.ascx) pages are processed by the .Net framework.

The syntax for declaring a directive is:

<%@ directive\_name attribute=value [attribute=value] %>

## The Application Directive

The Application directive defines application-specific attributes. It is provided at the top of the global.aspx file.

The basic syntax of Application directive is:

<%@ Application Language="C#" %>

## The Assembly Directive

The Assembly directive links an assembly to the page or the application at parse time. This could appear either in the global.asax file for application-wide linking, in the page file, a user control file for linking to a page or user control.

The basic syntax of Assembly directive is:

<%@ Assembly Name ="myassembly" %>

## The Import Directive

The Import directive imports a namespace into a web page, user control page of application. If the Import directive is specified in the global.asax file, then it is applied to the entire application. If it is in a page of user control page, then it is applied to that page or control.

The basic syntax for import directive is:

<%@ namespace="System.Drawing" %>

## The Master Directive

The Master directive specifies a page file as being the mater page.

The basic syntax of sample MasterPage directive is:

<%@ MasterPage Language="C#" AutoEventWireup="true" CodeFile="SiteMater.master.cs" Inherits="SiteMaster" %>

## The Page Directive

The Page directive defines the attributes specific to the page file for the page parser and the compiler.

The basic syntax of Page directive is:

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

## The Register Directive

The Register derivative is used for registering the custom server controls and user controls.

The basic syntax of Register directive is:

<%@ Register Src="~/footer.ascx" TagName="footer" TagPrefix="Tfooter" %>

# ASP.NET - Managing State

ASP.NET manages four types of states:

* View State
* Control State
* Session State
* Application State

## View State

The view state is the state of the page and all its controls. It is automatically maintained across posts by the ASP.NET framework.

The view state could be enabled or disabled for:

* **The entire application** by setting the EnableViewState property in the <pages> section of web.config file.
* **A page** by setting the EnableViewState attribute of the Page directive, as <%@ Page Language="C#" EnableViewState="false" %>
* **A control** by setting the Control.EnableViewState property.

## Session State

When a user connects to an ASP.NET website, a new session object is created. When session state is turned on, a new session state object is created for each new request. This session state object becomes part of the context and it is available through the page.

It can store all the primitive data types and arrays composed of primitive data types, as well as the DataSet, DataTable, HashTable, and Image objects, as well as any user-defined class that inherits from the ISerializable object.

## Application State

The ASP.NET application is the collection of all web pages, code and other files within a single virtual directory on a web server. When information is stored in application state, it is available to all the users.

To provide for the use of application state, ASP.NET creates an application state object for each application from the HTTPApplicationState class and stores this object in server memory. This object is represented by class file global.asax.

Application State is mostly used to store hit counters and other statistical data, global application data like tax rate, discount rate etc. and to keep the track of users visiting the site.

# ASP.NET - Validators

ASP.NET validation controls validate the user input data to ensure that useless, unauthenticated, or contradictory data don't get stored.

ASP.NET provides the following validation controls:

* RequiredFieldValidator
* RangeValidator
* CompareValidator
* RegularExpressionValidator
* CustomValidator
* ValidationSummary

## CustomValidator

The CustomValidator control allows writing application specific custom validation routines for both the client side and the server side validation.

The client side validation is accomplished through the ClientValidationFunction property. The client side validation routine should be written in a scripting language, such as JavaScript or VBScript, which the browser can understand.

The server side validation routine must be called from the control's ServerValidate event handler. The server side validation routine should be written in any .Net language, like C# or VB.Net.

The basic syntax for the control is as given:

<asp:CustomValidator ID="CustomValidator1" runat="server"

ClientValidationFunction=.cvf\_func. ErrorMessage="CustomValidator">

</asp:CustomValidator>

## ValidationSummary

The ValidationSummary control does not perform any validation but shows a summary of all errors in the page. The summary displays the values of the ErrorMessage property of all validation controls that failed validation.

The following two mutually inclusive properties list out the error message:

* **ShowSummary** : shows the error messages in specified format.
* **ShowMessageBox** : shows the error messages in a separate window.

The syntax for the control is as given:

<asp:ValidationSummary ID="ValidationSummary1" runat="server"

DisplayMode = "BulletList" ShowSummary = "true" HeaderText="Errors:" />

## Validation Groups

Complex pages have different groups of information provided in different panels. In such situation, a need might arise for performing validation separately for separate group. This kind of situation is handled using validation groups.

To create a validation group, you should put the input controls and the validation controls into the same logical group by setting their *ValidationGroup*property.

# ASP.NET - Database Access

ASP.NET allows the following sources of data to be accessed and used:

* Databases (e.g., Access, SQL Server, Oracle, MySQL)
* XML documents
* Business Objects
* Flat files

ADO.NET is the technology that provides the bridge between various ASP.NET control objects and the backend data source.

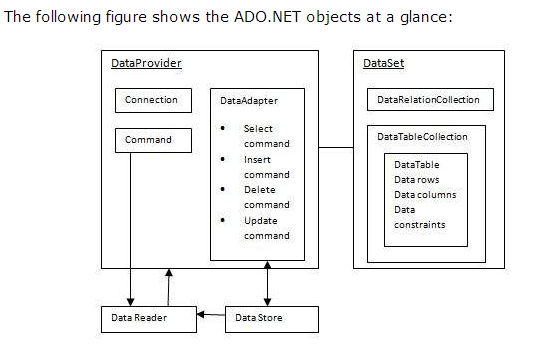
## Retrieve and display data

It takes two types of data controls to retrieve and display data in ASP.NET:

* **A data source control** - It manages the connection to the data, selection of data, and other jobs such as paging and caching of data etc.
* **A data view control** - It binds and displays the data and allows data manipulation.

# ADO.NET

ADO.NET provides a bridge between the front end controls and the back end database. The ADO.NET objects encapsulate all the data access operations and the controls interact with these objects to display data, thus hiding the details of movement of data.



## The DataSet Class

The dataset represents a subset of the database. It does not have a continuous connection to the database. To update the database a reconnection is required. The DataSet contains DataTable objects and DataRelation objects. The DataRelation objects represent the relationship between two tables.

## The DataTable Class

The DataTable class represents the tables in the database. It has the following important properties; most of these properties are read only properties except the PrimaryKey.

## The DataRow Class

The DataRow object represents a row in a table.

## The DataAdapter Object

The DataAdapter object acts as a mediator between the DataSet object and the database. This helps the Dataset to contain data from multiple databases or other data source.

## The DataReader Object

The DataReader object is an alternative to the DataSet and DataAdapter combination. This object provides a connection oriented access to the data records in the database. These objects are suitable for read-only access, such as populating a list and then breaking the connection.

## DbCommand and DbConnection Objects

The DbConnection object represents a connection to the data source. The connection could be shared among different command objects.

The DbCommand object represents the command or a stored procedure sent to the database from retrieving or manipulating data.

# ASP.NET - File Uploading

ASP.NET has two controls that allow users to upload files to the web server. Once the server receives the posted file data, the application can save it, check it, or ignore it. The following controls allow the file uploading:

* **HtmlInputFile** - an HTML server control
* **FileUpload** - and ASP.NET web control

Both controls allow file uploading, but the FileUpload control automatically sets the encoding of the form, whereas the HtmlInputFile does not do so.

The FileUpload control allows the user to browse for and select the file to be uploaded, providing a browse button and a text box for entering the filename.

# ASP.NET - Ad Rotator

The AdRotator control randomly selects banner graphics from a list, which is specified in an external XML schedule file. This external XML schedule file is called the advertisement file.

## The Advertisement File

The advertisement file is an XML file, which contains the information about the advertisements to be displayed.

# ASP.NET - Multi Views

MultiView and View controls allow you to divide the content of a page into different groups, displaying only one group at a time. Each View control manages one group of content and all the View controls are held together in a MultiView control.

The MultiView control is responsible for displaying one View control at a time. The View displayed is called the active view.

The View control cannot exist on its own. It would render error if you try to use it stand-alone. It is always used with a Multiview control as container.

The MultiView control has the following important properties:

|  |  |
| --- | --- |
| **Properties** | **Description** |
| Views | Collection of View controls within the MultiView. |
| ActiveViewIndex | A zero based index that denotes the active view. If no view is active, then the index is -1. |

# ASP.NET - Panel Controls

The Panel control works as a container for other controls on the page. It controls the appearance and visibility of the controls it contains. It also allows generating controls programmatically.

# ASP.NET - Ajax Control

AJAX stands for Asynchronous JavaScript and XML. This is a cross platform technology which speeds up response time. The AJAX server controls add script to the page which is executed and processed by the browser.

However like other ASP.NET server controls, these AJAX server controls also can have methods and event handlers associated with them, which are processed on the server side.

The control toolbox in the Visual Studio IDE contains a group of controls called the 'AJAX Extensions'

## The ScriptManager Control

The ScriptManager control is the most important control and must be present on the page for other controls to work.

## The UpdatePanel Control

The UpdatePanel control is a container control and derives from the Control class. It acts as a container for the child controls within it and does not have its own interface. When a control inside it triggers a post back, the UpdatePanel intervenes to initiate the post asynchronously and update just that portion of the page.

For example, if a button control is inside the update panel and it is clicked, only the controls within the update panel will be affected, the controls on the other parts of the page will not be affected. This is called the partial post back or the asynchronous post back.

## The UpdateProgress Control

The UpdateProgress control provides a sort of feedback on the browser while one or more update panel controls are being updated. For example, while a user logs in or waits for server response while performing some database oriented job.

It provides a visual acknowledgement like "Loading page...", indicating the work is in progress.

The UpdateProgress control displays for every asynchronous postback unless it is assigned to a single update panel using the AssociatedUpdatePanelID property.

## The Timer Control

The timer control is used to initiate the post back automatically. This could be done in two ways:

(1) Setting the Triggers property of the UpdatePanel control:

<Triggers>

<asp:AsyncPostBackTrigger ControlID="btnpanel2" EventName="Click" />

</Triggers>

(2) Placing a timer control directly inside the UpdatePanel to act as a child control trigger. A single timer can be the trigger for multiple UpdatePanels.

# ASP.NET - Data Sources

A data source control interacts with the data-bound controls and hides the complex data binding processes. These are the tools that provide data to the data bound controls and support execution of operations like insertions, deletions, sorting, and updates.

Each data source control wraps a particular data provider-relational databases, XML documents, or custom classes and helps in:

* Managing connection
* Selecting data
* Managing presentation aspects like paging, caching, etc.
* Manipulating data

Based on type of data, the data controls could be divided into two categories:

* Hierarchical data source controls
* Table-based data source controls

## Data Source Views

Data source views are objects of the DataSourceView class. Which represent a customized view of data for different data operations such as sorting, filtering, etc.

The DataSourceView class serves as the base class for all data source view classes, which define the capabilities of data source controls.

# ASP.NET - Data Binding

Every ASP.NET web form control inherits the DataBind method from its parent Control class, which gives it an inherent capability to bind data to at least one of its properties. This is known as **simple data binding** or **inline data binding**.

Simple data binding involves attaching any collection (item collection) which implements the IEnumerable interface, or the DataSet and DataTable classes to the DataSource property of the control.

On the other hand, some controls can bind records, lists, or columns of data into their structure through a DataSource control. These controls derive from the BaseDataBoundControl class. This is called **declarative data binding**.

The controls capable of simple data binding are derived from the ListControl abstract class and these controls are:

* BulletedList
* CheckBoxList
* DropDownList
* ListBox
* RadioButtonList

The controls capable of declarative data binding (a more complex data binding) are derived from the abstract class CompositeDataBoundControl. These controls are:

* DetailsView
* FormView
* GridView
* RecordList

## Simple Data Binding

Simple data binding involves the read-only selection lists. These controls can bind to an array list or fields from a database. Selection lists takes two values from the database or the data source; one value is displayed by the list and the other is considered as the value corresponding to the display.

# ASP.NET - Custom Controls

ASP.NET allows the users to create controls. These user defined controls are categorized into:

* User controls
* Custom controls

## User Controls

User controls behaves like miniature ASP.NET pages or web forms, which could be used by many other pages. These are derived from the System.Web.UI.UserControl class. These controls have the following characteristics:

* They have an .ascx extension.
* They may not contain any <html>, <body>, or <form> tags.
* They have a Control directive instead of a Page directive.

To add the user control to your web page, you must add the Register directive and an instance of the user control to the page.

## Custom Controls

Custom controls are deployed as individual assemblies. They are compiled into a Dynamic Link Library (DLL) and used as any other ASP.NET server control. They could be created in either of the following way:

* By deriving a custom control from an existing control
* By composing a new custom control combing two or more existing controls.
* By deriving from the base control class.

To use this control, this must be added as a reference to the web site before registering it on a page.

**User controls usually use to combine different existing controls.**

**Custom controls use to derive and extend the existing control. (Subclass).**

**Custom control can’t use visual editor.**

When you add a reference to the custom control, it is added to the toolbox and you can directly use it from the toolbox similar to any other control.

# ASP.NET - Personalization

Web sites are designed for repeated visits from the users. Personalization allows a site to remember the user identity and other information details, and it presents an individualistic environment to each user.

ASP.NET provides services for personalizing a web site to suit a particular client's taste and preference.

## Understanding Profiles

ASP.NET personalization service is based on user profile. User profile defines the kind of information about the user that the site needs. For example, name, age, address, date of birth, and phone number.

This information is defined in the web.config file of the application and ASP.NET runtime reads and uses it. This job is done by the personalization providers.

The user profiles obtained from user data is stored in a default database created by ASP.NET. You can create your own database for storing profiles. The profile data definition is stored in the configuration file web.config.

# ASP.NET - Error Handling

Error handling in ASP.NET has three aspects:

* **Tracing** - tracing the program execution at page level or application level.
* **Error handling** - handling standard errors or custom errors at page level or application level.
* **Debugging** - stepping through the program, setting break points to analyze the code

## Tracing

### [Viewing Trace Information](javascript:void(0))

You can view trace information at the bottom of individual pages.

To enable page level tracing, you need to modify the Page directive and add a Trace attribute as shown:

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"

Inherits="errorhandling.\_Default" **Trace** ="true" %>

It provides the following information at the top:

* Session ID
* Status Code
* Time of Request
* Type of Request
* Request and Response Encoding

The status code sent from the server, each time the page is requested shows the name and time of error if any.

Application level tracing applies to all the pages in the web site. It is implemented by putting the following code lines in the web.config file:

<system.web>

<trace enabled="true" />

</system.web>

## Error Handling

Although ASP.NET can detect all runtime errors, still some subtle errors may still be there. Observing the errors by tracing is meant for the developers, not for the users.

Hence, to intercept such occurrence, you can add error handing settings in the web.config file of the application. It is application-wide error handling. For example, you can add the following lines in the web.config file:

<configuration>

<system.web>

<customErrors mode="RemoteOnly" defaultRedirect="GenericErrorPage.htm">

<error statusCode="403" redirect="NoAccess.htm" />

<error statusCode="404" redirect="FileNotFound.htm" />

</customErrors>

</system.web>

<configuration>

# ASP.NET - LINQ

LINQ, the full name Language-Integrated Query. LINQ is set of extensions to the .Net Framework 3.5 and its managed languages that set the query as an object.   It defines a common syntax and a programming model to query different types of data using a common language.

For example, querying the Customers table in the Northwind database, using LINQ query in C#, the code would be:

var data = from c in dataContext.Customers

where c.Country == "Spain"

select c;

LINQ query can be applied to any data-bearing class that inherits from IEnumerable<T>, here T is any data type, for example, List<Book>.

### The Join clause

The 'join clause' in SQL is used for joining two data tables and displays a data set containing columns from both the tables. LINQ is also capable of that.

### The Where clause

The 'where clause' allows adding some conditional filters to the query.

### Orderby and Orderbydescending Clauses

These clauses allow sorting the query results.

### The Let clause

The let clause allows defining a variable and assigning it a value calculated from the data values.

# ASP.NET - Security

Implementing security in a site has the following aspects:

* **Authentication** : It is the process of ensuring the user's identity and authenticity. ASP.NET allows four types of authentications:
  + Windows Authentication
  + Forms Authentication
  + Passport Authentication
  + Custom Authentication
* **Authorization** : It is the process of defining and allotting specific roles to specific users.
* **Confidentiality** : It involves encrypting the channel between the client browser and the web server.
* **Integrity** : It involves maintaining the integrity of data. For example, implementing digital signature.

## IIS Authentication: SSL

The Secure Socket Layer or SSL is the protocol used to ensure a secure connection. With SSL enabled, the browser encrypts all data sent to the server and decrypts all data coming from the server. At the same time, the server encrypts and decrypts all data to and from browser.

The URL for a secure connection starts with HTTPS instead of HTTP. A small lock is displayed by a browser using a secure connection. When a browser makes an initial attempt to communicate with a server over a secure connection using SSL, the server authenticates itself by sending its digital certificate.

To use the SSL, you need to buy a digital secure certificate from a trusted Certification Authority (CA) and install it in the web server. Following are some of the trusted and reputed certification authorities:

* www.verisign.com
* www.geotrust.com
* www.thawte.com

SSL is built into all major browsers and servers. To enable SSL, you need to install the digital certificate. The strength of various digital certificates varies depending upon the length of the key generated during encryption. More the length, more secure is the certificate, hence the connection.

# ASP.NET - Data Caching

In some respect, caching is similar to storing the state objects. However, the storing information in state objects is deterministic, i.e., you can count on the data being stored there, and caching of data is nondeterministic.

## Caching in ASP.Net

ASP.NET provides the following different types of caching:

* **Output Caching** : Output cache stores a copy of the finally rendered HTML pages or part of pages sent to the client. When the next client requests for this page, instead of regenerating the page, a cached copy of the page is sent, thus saving time.
* **Data Caching** : Data caching means caching data from a data source. As long as the cache is not expired, a request for the data will be fulfilled from the cache. When the cache is expired, fresh data is obtained by the data source and the cache is refilled.
* **Object Caching** : Object caching is caching the objects on a page, such as data-bound controls. The cached data is stored in server memory.
* **Class Caching** : Web pages or web services are compiled into a page class in the assembly, when run for the first time. Then the assembly is cached in the server. Next time when a request is made for the page or service, the cached assembly is referred to. When the source code is changed, the CLR recompiles the assembly.
* **Configuration Caching** : Application wide configuration information is stored in a configuration file. Configuration caching stores the configuration information in the server memory.

## Output Caching

Rendering a page may involve some complex processes such as, database access, rendering complex controls etc. Output caching allows bypassing the round trips to server by caching data in memory. Even the whole page could be cached.

The OutputCache directive is responsible of output caching. It enables output caching and provides certain control over its behaviour.

Syntax for OutputCache directive:

<%@ OutputCache Duration="15" VaryByParam="None" %>

## Data Caching

The main aspect of data caching is caching the data source controls. We have already discussed that the data source controls represent data in a data source, like a database or an XML file. These controls derive from the abstract class DataSourceControl and have the following inherited properties for implementing caching:

* **CacheDuration** - It sets the number of seconds for which the data source will cache data.
* **CacheExpirationPolicy** - It defines the cache behavior when the data in cache has expired.
* **CacheKeyDependency** - It identifies a key for the controls that auto-expires the content of its cache when removed.
* **EnableCaching** - It specifies whether or not to cache the data.

## Object Caching

Object caching provides more flexibility than other cache techniques. You can use object caching to place any object in the cache. The object can be of any type - a data type, a web control, a class, a dataset object, etc. The item is added to the cache simply by assigning a new key name, shown as follows Like:

Cache["key"] = item;

ASP.NET also provides the Insert() method for inserting an object to the cache.

# ASP.NET - Web Services

A web service is a web application which is basically a class consisting of methods that could be used by other applications. It also follows a code-behind architecture such as the ASP.NET web pages, although it does not have a user interface.

# ASP.NET - Multi Threading

A thread is defined as the execution path of a program. Each thread defines a unique flow of control. If your application involves complicated and time consuming operations such as database access or some intense I/O operations, then it is often helpful to set different execution paths or threads, with each thread performing a particular job.

Threads are lightweight processes. One common example of use of thread is implementation of concurrent programming by modern operating systems. Use of threads saves wastage of CPU cycle and increases efficiency of an application.

In .Net, the threading is handled through the 'System.Threading' namespace.

## Creating Thread

A thread is created by creating a Thread object, giving its constructor a ThreadStart reference.

ThreadStart childthreat = new ThreadStart(childthreadcall);

## Thread Life Cycle

The life cycle of a thread starts when an object of the System.Threading.Thread class is created and ends when the thread is terminated or completes execution.

Following are the various states in the life cycle of a thread:

* **The Unstarted State** : It is the situation when the instance of the thread is created but the Start method is not called.
* **The Ready State** : It is the situation when the thread is ready to execute and waiting CPU cycle.
* **The Not Runnable State** : a thread is not runnable, when:
  + Sleep method has been called
  + Wait method has been called
  + Blocked by I/O operations
* **The Dead State** : It is the situation when the thread has completed execution or has been aborted.

## Thread Priority

The Priority property of the Thread class specifies the priority of one thread with respect to other. The .Net runtime selects the ready thread with the highest priority.

The priorities could be categorized as:

* Above normal
* Below normal
* Highest
* Lowest
* Normal

Once a thread is created, its priority is set using the Priority property of the thread class.

NewThread.Priority = ThreadPriority.Highest;

# ASP.NET - Configuration

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.

## 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>

## 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>

## 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.

# ASP.NET - Deployment

There are two categories of ASP.NET deployment:

* **Local deployment** : In this case, the entire application is contained within a virtual directory and all the contents and assemblies are contained within it and available to the application.
* **Global deployment** : In this case, assemblies are available to every application running on the server.

There are different techniques used for deployment, however, we will discuss the following most common and easiest ways of deployment:

* XCOPY deployment
* Copying a Website
* Creating a set up project

## XCOPY Deployment （not recommend）

XCOPY deployment means making recursive copies of all the files to the target folder on the target machine. You can use any of the commonly used techniques:

* FTP transfer
* Using Server management tools that provide replication on a remote site
* MSI installer application

XCOPY deployment simply copies the application file to the production server and sets a virtual directory there. You need to set a virtual directory using the Internet Information Manager Microsoft Management Console (MMC snap-in).

## Copying a Website（not recommend）

The Copy Web Site option is available in Visual Studio. It is available from the Website -> Copy Web Site menu option. This menu item allows copying the current web site to another local or remote location. It is a sort of integrated FTP tool.

Using this option, you connect to the target destination, select the desired copy mode:

* Overwrite
* Source to Target Files
* Sync UP Source And Target Projects

Then proceed with copying the files physically. Unlike the XCOPY deployment, this process of deployment is done from Visual Studio environment. However, there are following problems with both the above deployment methods:

* You pass on your source code.
* There is no pre-compilation and related error checking for the files.
* The initial page load will be slow.

## Creating a Setup Project (recommended)

In this method, you use Windows Installer and package your web applications so it is ready to deploy on the production server. Visual Studio allows you to build deployment packages. Let us test this on one of our existing project, say the data binding project.