
ASP.NET
Reputation
CRM
Quality
Goals
Team
Finance
Customers
Costs
Income
Plan
Marketing
Management
Service

Jump2Learn
The Online Learning Place

ASP.NET

CHAPTER 5 : ADVANCED ASP.NET

AUTHORS

Dr. Ashoksinh V. Solanki
M.C.A., Ph.D.
COLLEGE OF APPLIED SCIENCE AND PROFESSIONAL STUDIES, CHIKHLI

Mr. Yatin Solanki
M.C.A., Ph.D.(Pursuing)
DOLAT USHA INSTITUTE OF APPLIED SCIENCES, VALSAD

Mr. Chetan Parmar
M.C.A., NET, SET
NARMADA COLLEGE OF SCIENCE & COMMERCE, ZADESHWAR

Website : www.jump2learn.com | **Email :** info@jump2learn.com | **Instagram :** www.instagram.com/jump2learn
Facebook : www.facebook.com/Jump2Learn | **Whatsapp :** +91-909-999-0960 | **YouTube :** Jump2Learn

WHAT IS WEB.CONFIG FILE?

- Web.config file, as it sounds like is a configuration file for the Asp .net web application.
- An Asp .net application has one web.config file which keeps the configurations required for the corresponding application.
- Web.config file is written in XML with specific tags having specific meanings.

WHAT IS MACHINE.CONFIG FILE?

- As web.config file is used to configure one asp .net web application, same way Machine.config file is used to configure the application according to a particular machine.
- That is, configuration done in machine.config file is affected on any application that runs on a particular machine.
- Usually, this file is not altered and only web.config is used which configuring applications.

WHAT CAN BE STORED IN WEB.CONFIG FILE?

- There are number of important settings that can be stored in the configuration file. Here are some of the most frequently used configurations, stored conveniently inside Web.config file.

    1) **DATABASE CONNECTIONS**
    2) **SESSION STATES**
    3) **ERROR HANDLING**
    4) **SECURITY**

1) DATABASE CONNECTIONS:

- The most important configuration data that can be stored inside the web.config file is the database connection string.
- Storing the connection string in the web.config file makes sense, since any modifications to the database configurations can be maintained at a single location. As otherwise we'll have to keep it either as a class level variable in all the associated source files or probably keep it in another class as a public static variable.
- But it this is stored in the Web.config file, it can be read and used anywhere in the program.
- This will certainly save us a lot of alteration in different files where we used the old connection.
- Lets see a small example of the connection string which is stored in the web.config file .

```
<configuration>
 <appSettings>
   <add key="ConnectionString"
      value="server=localhost;uid=sa;
            pwd=;database=DBPerson" />
 </appSettings>
</configuration>
```

- As you can see it is really simple to store the connection string in the web.config file. The connection string is referenced by a key which in this case is "ConnectionString"
- The value attribute of the configuration file denotes the information about the database
- Here we can see that if has database name, userid and password. You can define more options if you want.
- Lets see how we access the connection string from our Asp .net web application.

Dim cnn as New OleDbConnection = ConfigurationSettings.AppSettings("ConnectionString");

- The small code snippet above is all that is needed to access the value stored inside the Web.config file.

2) Session States

- Session in Asp .net web application is very important. As we know that HTTP is a stateless protocol and we needs session to keep the state alive
- Asp .net stores the sessions in different ways. By default the session is stored in the asp .net process. You can always configure the application so that the session will be stored in one of the following ways.

2.1 Session State Service

There are two main advantages of using the State Service.

- First the state service is not running in the same process as the asp .net application. So even if the asp .net application crashes the sessions will not be destroyed.
- Any advantage is sharing the state information across a Web garden (Multiple processors for the same computer).
- Lets see a small example of the Session State Service.
- <sessionState mode="StateServer" stateConnectionString="tcpip=127.0.0.1:55455"
- sqlConnectionString="data source=127.0.0.1;user id=sa;password='' cookieless="false"
- timeout="20"/>
- **mode:** This can be StateServer or SqlServer. Since we are using StateServer we set the mode to StateServer.
- **stateConnectionString:** connectionString that is used to locate the State Service.
- **sqlConnectionString:** The connection String of the sql server database.
- **cookieless:** Cookieless equal to false means that we will be using cookies to store the session on the client side.

2.2 SQL SERVER

- The final choice to save the session information is using the Sql Server 2000 database. To use Sql Server for storing session state you need to do the following:
- Run the InstallSqlState.sql script on the Microsoft SQL Server where you intend to store the session.
- You web.config settings will look something like this:
- <sessionState mode = "SqlServer" stateConnectionString="tcpip=127.0.0.1:45565" sqlConnectionString="data source="SERVERNAME;user id=sa;password='' cookiesless="false" timeout="20"/>
- SQL Server lets you share session state among the processors in a Web garden or the servers in a Web farm. Apart from that you also get additional space to store the session. And after that you can take various actions on the session stored.
- The downside is SQL Server is slow as compared to storing session in the state in process. And also SQL Server cost too much for a small company.

2.3 INPROC:

- This is another Session State. This one is mostly used for development purposes. The biggest advantage of using this approach is the applications will run faster when compared to other Session state types.

- But the disadvantage is Sessions are not stored when there is any problem that occurs with the application, when there is a small change in the files etc., Also there could be frequent loss of session data experienced.

Error Handling:

- Error handling is one of the most important part of any web application. Each error has to be caught and suitable action has to be taken to resolve that problem. ASP. net web.config file lets us configure, what to do when an error occurs in our application.
- Check the following xml tag in the web.config file that deals with errors:

```
<customErrors mode = "On">
    <error statusCode = "404" redirect = "errorPage.aspx" />
</customErrors>
```

- This tells the Asp.net to display custom errors from a remote client or a local client and to display a page named errorPage.aspx. Error "404" is "Page not found" error.
- If custom error mode is turned "off" than you will see Asp.net default error message. This error messages are good for debugging purposes but should never be exposed to the users. The users should always be presented with friendly errors if any.

3) SECURITY:

The most critical aspect of any application is the security. Asp.net offers many different types of security method which can be used depending upon the condition and type of security you need.

1) NO AUTHENTICATION:

No Authentication means "No Authentication", meaning that Asp.net will not implement any type of security.

2) WINDOWS AUTHENTICATION:

The Windows authentication allows us to use the windows user accounts. This provider uses IIS to perform the actual authentication, and then passes the authenticated identity to your code. If you like to see that what windows user is using the Asp.net application you can use:

User.Identity.Name;

This returns the *DOMAIN\UserName* of the current user of the local machine.

3) PASSPORT AUTHENTICATION:

Passport Authentication provider uses Microsoft's Passport service to authenticate users. You need to purchase this service in order to use it.

4) FORMS AUTHENTICATION:

- Forms Authentication uses HTML forms to collect the user information and than it takes required actions on those HTML collected values.
- In order to use Forms Authentication you must set the Anonymous Access checkbox checked Now we need that whenever user tries to run the application he/she will be redirected to the login page.

```
<authentication mode="Forms">
        <forms loginUrl = "frmLogin.aspx"  name="3345C“
         timeout="1"/>
</authentication>
<authorization>
        <deny users="?" />
</authorization>
```

- As you can see we set the Authentication mode to "Forms". The forms loginUrl is the first page being displayed when the application is run by any user.
- The authorization tags has the deny users element which contains "?", this means that
    - full access will be given to the authenticated users and none access will be given to the unauthenticated users.
    - You can replace "?" with "*" meaning that all access is given to all the users no matter what.

❖ SITEMAPPATH SERVER CONTROL

- SiteMapPath is a Navigation Control.
- The SiteMapPath control displays the navigation path to the current page respect to home page. The path acts as clickable links to previous pages.
- It is used to access web pages of the website from one webpage to another. It is displays the map of the site related to its web pages.
- Maintaining the menu of a large web site is difficult and time consuming. The SiteMapPath control obtains navigation data from a site map. This data includes information about the pages in your Web site, such as the URL, title, description, and location in the navigationhierarchy. Storing navigation data in one place makes it easier to add and remove items in the navigational menus of a Web site. If you use it on a page that is not represented in your site map, it will not be displayed.
- It provides a space-saving way to easily navigate a site and serves as a point of reference for where the currently displayed page is within a site. TreeView or Menu might require too much space on a page.
- Make site well-structured or well-linked (internal links)

SYNTAX

```
<asp:SiteMapPath runat="server" />
```

NODES

The SiteMapPath is made up of nodes. Each element in the path is called a node and is represented by a SiteMapNodeItem object.

NODE TYPE	DESCRIPTION
root	A node that anchors or represents base of a hierarchical set of nodes.
parent	A node that has one or more child nodes.
current	A node that represents the currently displayed page.

Property: common property

OTHER IMPORTANT PROPERTY

NodeStyle	Gets the style used for the display text for all nodes in the site navigation path.
ParentLevelsDisplayed	Gets or sets the number of levels of parent nodes the control displays.
PathDirection	Gets or sets the order that the navigation path nodes are rendered in. Values are: RootToCurrent or CurrentToRoot
PathSeparator	It set path separator between nodes in the navigation path.
PathSeparatorStyle	Set the style used for the PathSeparator.
PathSeparatorTemplate	sets a template to use for the path delimiter of a site navigation path.
RenderCurrentNodeAsLink	Indicates whether the site navigation node is rendered as a hyperlink.
RootNodeStyle	Set the style for the root node display text.
RootNodeTemplate	sets a template to use for the root node of a site navigation path.
Site	Gets information about the container that hosts the current control when rendered on a design surface.

METHODS

NAME	DESCRIPTION
DataBind	Infrastructure. Binds a data source to the SiteMapPath control and its child controls.
Dispose	Enables a server control to perform final clean up before it is released from memory.
Focus	Sets input focus to a control.
InitializeItem	Populates a SiteMapNodeItem

EVENT

NAME	DESCRIPTION
DataBinding	Occurs when the server control binds to a data source
Disposed	Occurs when a server control is released from memory, which is the last stage of the server control lifecycle when an ASP.NET page is requested.
Init	Occurs when the server control is initialized, which is the first step in its lifecycle.
ItemDataBound	Occurs after a SiteMapNodeItem has been bound to its underlying SiteMapNode

Load	Occurs when the server control is loaded into the Page object.
PreRender	Occurs after the Control object is loaded but prior to rendering.
Unload	Occurs when the server control is unloaded from memory.

❖ USER CONTROL

ASP.NET USER CONTROLS

- In addition to using Web server controls in your ASP.NET Web pages, you can create your own custom, reusable controls using the same techniques you use for creating ASP.NET Web pages. These controls are called **user controls.**
- A user control is a kind of composite control that works much like an ASP.NET Web page—you can add existing Web server controls and markup to a user control, and define properties and methods for the control. You can then embed them in ASP.NET Web pages, where they act as a unit.
- At times, you might need functionality in a control that is not provided by the built-in ASP.NET Web server controls. In those cases, you can create your own controls. You have two options. **You can create:**
- User controls. User controls are containers into which you can put markup and Web server controls. You can then treat the user control as a unit and define properties and methods for it.
- Custom controls. A custom control is a class that you write that derives from Control or WebControl.
- User controls are substantially easier to create than custom controls, because you can reuse existing controls. They make it particularly easy to create controls with complex user interfac elements.

How to: Create ASP.NET User Controls

You create ASP.NET user controls in much the same way that you design ASP.NET Web pages. You can use the same HTML elements and controls on a user control that you do on a standard ASP.NET page. However, the user control does not have html, body, and form elements; and the file name extension must be .ascx.

TO CREATE AN ASP.NET USER CONTROL

1. Open the Web site project to which you want to add user controls. If you do not already have a Web site project, you can create one.
2. On the **Website** menu, click **Add New Item**. The **Add New Item** dialog box appears.
3. In the **Add New Item** dialog box, under **Visual Studio installed templates**, click **Web User Control**.
4. In the **Name** box, type a name for the control By default, the .ascx file name extension is appended to the control name that you type.
5. From the **Language** list, select the programming language that you want to use.

6. Optionally, if you want to keep any code for the user control in a separate file, select the **Place code in separate file** check box.
7. Click **Add**.

The new ASP.NET user control is created and then opened in the designer. The markup for this new control is similar to the markup for an ASP.NET Web page, except that it contains an @ Control directive instead of an @ Page directive, and the user control does not have html, body, and form elements.

Add any markup and controls to the new user control, and add code for any tasks that the user control will perform, such as handling control events or reading data from a data source.

HOW TO: INCLUDE ASP.NET USER CONTROLS IN WEB PAGES VISUAL STUDIO 2010

Adding an ASP.NET user control to a Web page is similar to adding other server controls to the page. However, you must be sure to follow the procedure below so that all of the necessary elements are added to the page.

TO ADD AN ASP.NET USER CONTROL TO A WEB PAGE

1. In Visual Web Developer, open the Web page to which you want to add the ASP.NET user control.
2. Switch to Design view.
3. Select your custom user control file in Solution Explorer, and drag it onto the page.

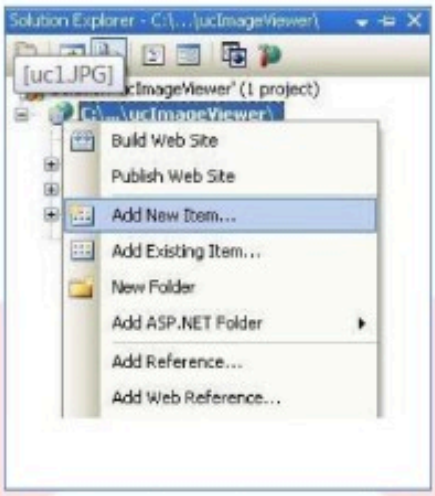
The ASP.NET user control is added to the page. In addition, the designer creates the @ Register directive, which is required for the page to recognize the user control.
You can now work with the control's public properties and methods.

CREATING WEB USER CONTROL IN ASP.NET

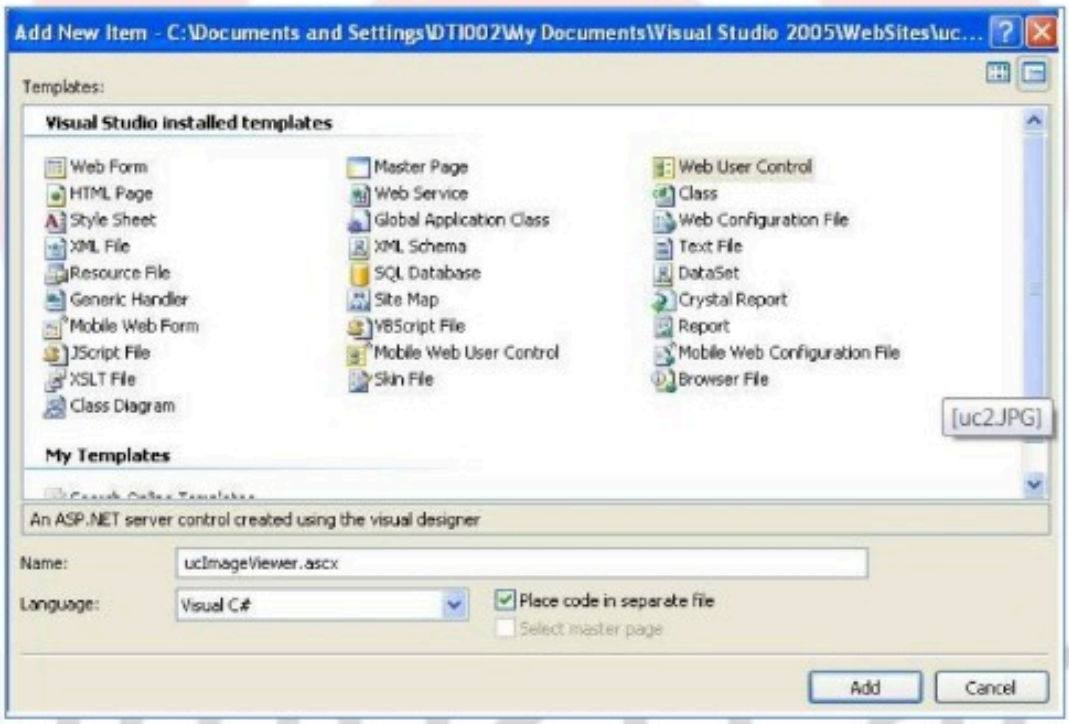
We are going to create a simple ImageViewer Control, which display thumbnail view of image and on clicking "Original View" under it will redirect you to original view of Image. Practically it has is scrap, but I have used here so that example remain simple throughout the discussion.

STEP1: Create Asp.net Website Project

STEP2: Add User Control as shown in figure.



Select "Web User Control" from New Item Window and name it "ucImageViewer"



STEP3: Now Add following code to file .ascx

Here it Adds a Image Server Control and Place a Button control below it. To organize properly it has displayed in table.

%@ Control Language="C#" AutoEventWireup="true"CodeFile="ucImageViewer.ascx.cs" Inherits="ucImageViewer"%>

<table border="0" cellpadding="0" cellspacing="0">

```
<tr>
<td align="center">

<asp:Image ID="Image1" runat="server" />

</td>
</tr>
<tr>
<td align="center">

<asp:Button ID="Button1" runat="server" Text="Original View"OnClick="Button1_Click" />

</td>
</tr>
</table>
```

STEP4: Now as we want to view "Original View" of Image, we need to write a code on button click control, so for that switch to code view i.e. open **.ascx.cs** file and add following code.

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

{
if (ImagePath == string.Empty)

return;
else
Response.Redirect(ImagePath);
}
```

STEP5: Adding public property to Web User Control

Note, here you can access the Image Control property within User Control, but it won't be available when you place user control on web form, so to make desired ImageControl property accessible you need to define public property.


```
public string ImagePath

{
get
{
return m_ImagePath;

}
```

```
set
{
m_ImagePath = value;

}
}
public string ImageName

{
get
{
return m_ImageName;

}
set
{
m_ImageName = value;

}
}

public int ImageHeight

{
get
{
return m_ImageHeight;

}
set
{
m_ImageHeight = value;

}
}

public int ImageWidth

{
get
{
return m_ImageWidth;

}
set
```

```
{
m_ImageWidth = value;

}
}
```

So we are done forming Web User Control, now its turn to render Web User Control on Web Form.

DISPLAYING WEB USER CONTROL ON WEB FORM IN ASP.NET

To display web user control you can simply drag the user control on to web form as you are dragging web server control.

Lets understand manually how we can add the Web User Control on Web Form

STEP1: Add Register directive

Like <%@ Page %> directive, you can create Register directive

```
<%@ Register Src="ucImageViewer.ascx" TagName="ImageViewer" TagPrefix="uc" %>
```

Understanding Attributes of Register directive

Src - It specifies source of web user control.

TagPrefix - It Specifies Prefix for user control, i.e. Namespace to which it belong.

TagName - It Specifies Alias to the user control. i.e. Id to Control.

Now, you need to add the control

```
<uc:ImageViewer ID="ImageViewer1" runat="server" />
```


Note: As we have form public property with Web User Control, we can access

ImagePath - To Specify path of Image.

ImageName - To display Image Name as alternate text.

ImageHeight

ImageWidth

Example:

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

{
ImageViewer1.ImagePath = "Image/shriganesh1.jpg";

ImageViewer1.ImageName = "Shri Ganesh 1st Pic";

ImageViewer1.ImageHeight = 200;

ImageViewer1.ImageWidth = 200;

}
```

DISPLAYING OUTPUT OF WEB USER CONTROL RENDERING

After clicking button, it display Image Original View as shown in figure.

Now, lets understand how to load user control dynamically.

DYNAMICALLY LOADING OF USER CONTROL IN ASP.NET

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

{
Control c = Page.LoadControl("ucImageViewer.ascx");


((ucImageViewer)c).ImagePath = "Image/shriganesh1.jpg";

((ucImageViewer)c).ImageName = "Shri Ganesh Pic";

((ucImageViewer)c).ImageHeight = 150;

((ucImageViewer)c).ImageWidth = 150;


Panel1.Controls.Add(c);
}
```

CREATING A WEB USER CONTROL FROM EXISTING WEB FORM IN ASP.NET

STEP1: Remove all <HTML>, <Body> and <Form> Tags.

STEP2: Change the @Page directive to a @Control directive and ensures that no unsupported attributes remain.
STEP3: Add a ClassName attribute to the @Control directive.

STEP4: Rename the file to a name that reflects its purpose and then change the file extension from .aspx to .ascx.

OVERVIEW OF WEB CUSTOM CONTROL

Web Cutsom Controls provides more flexibility as compare to web user control. With Web Custom Control you can provide

- Design-Time Support
- Data Binding
- Event Handling and more advance feature.

DIFFERENT WAYS TO CREATE WEB CUSTOM CONTROL

There are 3 different ways of creating a web custom control

1. Composite Control - By combining two or more controls.
2. Derived Control - By Inheriting from a server control.
3. From Scratch - By Inheriting from the generic System.Web.UI.Control class.

WEB USER CONTROL VS WEB CUSTOM CONTROL

Difference between Web User Control and Web Custom Control or Points to be consider while Choosing Web User Control or Web Custom Control

- Web User Control can only be used within the same project, while Web Custom Control can be used across many projects.
- Web User Control cannot be added to Visual Studio.Net Toolbox, while Web Custom Control can be added.
- Web Custom Control are better choice when you want dynamic layout tasks in which constituent controls must be created at runtime.

FACTS ABOUT WEB USER CONTROL IN ASP.NET

- When you include a web user control in a web form, the user control participates in the event life cycle for the web form.
- User control Load and Pre-Render events are fired only after the web form's load and PreRender events are fired respectively.
- Web User controls are not precompiled into assemblies like the components and custome controls, they can only be used in web applications that have a physical copy of the user control. Thus, every application that wants to use a user control should have its own copy of that user control.
- Web User Control cannot be added into Visual Studio.Net Toolbox
- Web User Control do not expose their properties through the properties window
- Web User Controls in asp.net are Language neutral, that is user control written in C# can be used on a web form that is written in VB.Net.

❖ WEB SERVICES

- A web service is an entity that you can program to provide a particular functionality to application over the internet.
- A web service allows a website to communicate with other websites irrespective of the programming languages in which they are created.
- A web service can be accessed by any application, regardless of the software and hardware platforms on which the application is running, because the web service complies with common industry standards, such as **Simple Object Access Protocol (SOAP)** and **Web Services Description Language (WSDL).**
- A service does not have any user interface; it only contains the logic for providing specific services to its clients.
- A web service provides an abstraction between the client and the provider of the web service.

ADVANTAGES OF WEB SERVICES:-

- Web services are simple to use; and consequently, they can be implemented on varied platforms.
- Web services are loosely coupled; as a result, their interfaces and methods can be extended.
- Web services do not carry any state information with them so that multiple requests can be processed simultaneously.
- **Interoperability** - This is the most important benefit of Web Services. Web Services typically work outside of private networks, offering developers a non-proprietary route to their

solutions. To the use of standards-based communications methods, Web Services are virtually platform-independent.

- **Usability** - Web Services allow the business logic of many different systems to be exposed over the Web. This gives your applications the freedom to chose the Web Services that they need. Instead of re-inventing the wheel for each client, you need only include additional application-specific business logic on the client-side. This allows you to develop services and/or client-side code using the languages and tools that you want.
- **Reusability** - Web Services provide not a component-based model of application development, but the closest thing possible to zero-coding deployment of such services. This makes it easy to reuse Web Service components as appropriate in other services.
- **Deployability** - Web Services are deployed over standard Internet technologies. This makes it possible to deploy Web Services even over the fire wall to servers running on the Internet on the other side of the globe.

DISADVANTAGES OF WEB SERVICES:-

- Although the simplicity of Web services is an advantage in some respects, it can also be a obstacle. Web services use plain text protocols that use a fairly long-winded method to identify data. This means that Web service requests are larger than requests encoded with a binary protocol. The extra size is really only an issue over low-speed connections, or over extremely busy connections.
- Although HTTP is simple, they weren't really meant for long-term sessions. Typically, a browser makes an HTTP connection, requests a Web page and maybe some images, and then disconnects. The server may periodically send data back to the client. This kind of interaction is difficult with Web services, and you need to do a little extra work to make up for what HTTP doesn't do for you.

- The problem with HTTP, when it comes to Web services is that these protocols are "**stateless**"—the interaction between the server and client is typically brief and when there is no data being exchanged, the server and client have no knowledge of each other. More specifically, if a client makes a request to the server, receives some information, and then immediately crashes, the server never knows that the client is no longer active. The server needs a way to keep track of what a client is doing and also to determine when a client is no longer active.

- Typically, a server sends some kind of session identification to the client when the client first accesses the server. The client then uses this identification when it makes further requests to the server. This enables the server to recall any information it has about the client. A server must usually rely on a timeout mechanism to determine that a client is no longer active. If a server doesn't receive a request from a client after a predetermined amount of time, it assumes that the client is inactive and removes
- any client information it was keeping. This extra overhead means more work for Web service developers.

SIMPLE OBJECT ACCESS PROTOCOL (SOAP)

- It is one of the key element in web service, is a protocol used for messaging. It is completely XML-based.
- SOAP provides a complete set of rules for messages, called SOAP envelopes, as well as the rules for issues, such as data encoding, message handling and message binding, to other protocol, such as HTTP.
- One of the great advantage of using a SOAP message is thatit is not restricted to any particular protocol.
- The two features that make SOAP a widely used protocol are support for remote procedure calls and a platform independency.
- The platform independent feature of SOAP allows a web service to communication with various applications, irrespective of hardware or software platform on which they are running.

WEB SERVICE DESCRIPTION LANGUAGE (WSDL)

- WSDL is an XML-based language that defines Web services.
- Every web service consists of a corresponding WSDL document that specifies the location of the web service and lists all the services that the web service can perform.
- A WSDL document is a simple XML document containing different XML elements to define a web service.

WEB SERVICE PROTOCOLS USED FOR DATA TRANSMISSION

- **HTTP-GET** :- Creates a Query String of the name /value pair and appends it to the URL of the requesting script on the server.
- **HTTP-POST** :- Passes the name/value pair in the body of the HTTP request message.
- **SOAP** :- Exchanges information in a decentralized, distributed environment.
- **MIME** :- It stands for Multipurpose Internet Mail Extension. It defines the standard representation for complex message bodies, such as messages with graphics and audio clips.

❖ ERROR HANDLING

The ***Errors*** caused by a computer program, regardless of the language in which the program is written.

It can be categorized into three major groups:

- Design-time
- Runtime
- Logic.

DESIGN-TIME

A ***Design-time error*** is also known as a **Syntax error**. These occur when the environment you're programming in doesn't understand your code. These are easy to track down in VB.NET, because you get a blue wiggly line pointing them out. If you try to run the programme, you'll get a dialogue box popping up telling you that there were Build errors.

RUNTIME

Runtime errors are a lot harder to track down. As their name suggests, these errors occur when the programme is running.

They happen when your programme tries to do something it shouldn't be doing. An example is trying to access a file that doesn't exist. Runtime errors usually cause your programme to crash. If and when that happens, you get the blame. After all, you're the programmer, and you should write code to trap runtime errors. If you're trying to open a database in a specific location, and the database has been moved, a Runtime error will occur. It's your job to predict a thing like this, and code accordingly.

LOGIC

LOGIC ERRORS also occur when the programme is running. They happen when your code doesn't quite behave the way you thought it would.

A classic example is creating an infinite loop of the type "Do While x is greater than 10". If x is always going to be greater than 10, then the loop has no way to exit, and just keeps going round and round. Logic errors tend not to crash your programme. But they will ensure that it doesn't work properly.

EXCEPTION:

An exception is a problem that arises during the execution of a program. An exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

The Exception Handling is categories into two parts:

- Structured Exception Handling
- Unstructured Exception Handling

STRUCTURED EXCEPTION HANDLING

In ***structured exception handing***, we write code surrounded by blocks. If an exception occurs, the block throws the execution control to a predefined handled code, the try, catch, finally statements define these blocks. In other words, if an application handles exceptions that occur during the execution of a block of application code, the code must be placed with in a try statement.

Syntax:

```
Try
  [ tryStatements ]
  [ Exit Try ]
[ Catch [ exception [ As type ] ] [ When expression ]
  [ catchStatements ]
  [ Exit Try ] ]
[ Catch ... ]
[ Finally
  [ finallyStatements ] ]
End Try
```

Assuming a block will raise and exception, a method catches an exception using a combination of the Try and Catch keywords. A ***Try/Catch*** block is placed around the code that might generate an exception. Code within a ***Try/Catch*** block is referred to as protected code.

You can list down multiple catch statements to catch different type of exceptions in case your try block raises more than one exception in different situations.

The codes reside in ***Finally*** block is executed every time if the Exception is generated or not.

Example:

```
Module Module1

  Sub division(ByVal num1 As Integer, ByVal num2 As Integer)
    Dim result As Integer
    Try
      result = num1 \ num2
    Catch e As DivideByZeroException
      Console.WriteLine("Exception caught: {0}", e)
    Finally
      Console.WriteLine("Result: {0}", result)
    End Try
  End Sub

  Sub Main()
    division(25, 0)
    Console.Read()
  End Sub

End Module
```

Result:

```
Exception caught: System.DivideByZeroException: Attempted to divide by zero.
  at ConsoleApplication1.Module1.division(Int32 num1, Int32 num2) in
E:\CP\Book\ConsoleApplication1\ConsoleApplication1\Module1.vb:line 5
Result: 0
```

If we change the statement division(25, 5) then output is Result : 5

THROWING OBJECTS

You can ***throw*** an object if it is either directly or indirectly derived from the **System.Exception** class. You can use a throw statement in the catch block to throw the present object as:

Syntax:

Throw [expression]

Example:

```
Module Module1

  Sub Main()
    Try
      Console.WriteLine("Inside the Try block")
      Throw New ApplicationException("A custom exception is being thrown here...")
    Catch e As Exception
      Console.WriteLine(e.Message)
    Finally
      Console.WriteLine("Now inside the Finally Block")
    End Try
    Console.Read()
  End Sub

End Module
```

Result:

```
Inside the Try block
A custom exception is being thrown here...
Now inside the Finally Block
```

UNSTRUCTURED EXCEPTION HANDLING

The ***Unstructured Exception*** handling revolves around the ***On Error GoTo*** statement.

ON ERROR GOTO STATEMENT.

You use this statement to tell VB .NET where to transfer control to in case there's been an exception,

Synatx:

On Error { GoTo [*line* | 0 | -1] | Resume Next }

Where
GoTo *line* - Calls the error-handling code that starts at the line specified at *line*. Here, *line* is a line label or a line number. If a runtime error occurs, program execution goes to the given location. The specified line must be in the same procedure as the On Error statement.
GoTo 0 - Disables the enabled error handler in the current procedure.
GoTo -1 - Same as GoTo 0.
Resume Next - Specifies that when an exception occurs, execution skips over the statement that caused the problem and goes to the statement immediately following. Execution continues from that point.

Example:

```
Module Module1

  Sub Main()
    Dim a, b, c  As Integer
    a=1
    b=0
    On Error GoTo Handler
    c=a / b
    System.Console.WriteLine("The answer is {0}", c)
    Console.Read()
     Exit Sub
Handler:
    System.Console.WriteLine("Number Can not be divide by zero")
    Resume Next
  End Sub
End Module
```

Result:

```
Number Can not be divide by zero
The answer is 0
```

In above example when the code performs a division by zero, which causes an exception. When the exception occurs, control will jump to the exception handler, where I am display a message and then use the ***Resume Next*** statement to transfer control back to the statement immediately after the statement that caused the exception:

RESUME STATEMENT

After an exception occurs, you can use the *Resume* ***statement*** to resume program execution in unstructured exception handling. Here are the possibilities:

- Resume resumes execution with the statement that caused the error.
- Resume Next resumes execution with the statement after the one that caused the error.
- Resume *line* resumes execution at *line*, a line number or label that specifies where to resume execution.

Example:

```
Module Module1
  Sub Main()
     On Error GoTo Handler
     Dim i1 As Integer = 10
     Dim i2 As Integer = 0
     Dim i3 As Integer
      I3 = i1 / i2
LineAfter:
   Console.WriteLine("Press Enter to continue...")
   Console.ReadLine()
   Exit Sub
Handler:
   Console.WriteLine("An overflow error occurred.")
   Resume LineAfter
  End Sub
End Module
```

Result:

```
An overflow error occurred.
Press Enter to continue...
```

Err Object

When error occur, the ***Err*** object contains information about the error, which helps you to determine whether you can attempt to fix the error or ignore the error.

It have several methods that allow you to raise errors or clear the state of the ***Err*** object.

Example:

```
Module Module1

  Sub Main()
   On Error GoTo Handler
```

```
        Dim i1 As Integer = 10
        Dim i2 As Integer = 0
        Dim i3 As Integer
        i3 = i1 / i2
        Console.WriteLine("Press Enter to continue...")
        Console.ReadLine()
        Exit Sub
    Handler:
        Console.WriteLine(Err.Description)
        Console.WriteLine("Press Enter to continue...")
        Console.ReadLine()
    End Sub

    End Module
```

Result:

```
Arithmetic operation resulted in an overflow.
Press Enter to continue...
```