



D.K.M.COLLEGE FOR WOMEN (AUTONOMOUS), VELLORE



eRESOURCES

Digital Learning

E CONTENT TITLE: BSCS 65/

BSCA 65 ASP .NET

DEPARTMENT : (B.Sc CS / BCA)

DESIGNED BY : 1. Mrs.B. Arulmozhi, M. Phil.,
2. Mrs. K.Ayesha, M. Phil.,
3. Mrs. S.Shanthi, M.Phil.,
4. Mrs. V. Lakshmi Pratha, M.Phil.,
5. Mrs. P.Ramya, M.Phil.,

UNIT - I**Skilled Based Subject IV****ASP .NET****UNIT I : ASP.NET Basics**

Introduction to ASP.NET: .NET Framework (CLR, CLI, BCL), ASP.NET Basics, ASP.NET Page Structure, Page Life Cycle. Controls: HTML Server Controls, Web Server Controls, Web User Controls, Validation Controls, Custom Web Controls.

UNIT II: Form

Form validation: Client side validation, Server side validation, Validation Controls: Required Field Comparison Range, Calendar Control, Ad rotator Control, Internet Explorer Control. State Management: View State, Control State, Hidden Fields, Cookies, Query Strings, Application State, Session State.

UNIT III: ADO.NET

Architecture of ADO .NET, Connected and Disconnected Database, Create Database, Create connection Using ADO.NET Object model, Connection Class, Command Class, Data Adapter Class, Dataset Class, Display data on data bound controls and Data Grid.

UNIT IV: Database accessing

Database accessing on Web Applications: Data Binding Concept with web, Creating Data Grid, Binding standard web server controls, Display data on web form using Data Bound Controls.

UNIT V: XML

Writing Datasets to XML, Reading datasets with XML. WEB services: Remote method call using XML, SOAP, Web service description language, Building and Consuming a web service, Web Application deployment.

Textbook:

Professional ASP.NET 1.1 Bill Evjen , Devin Rader , Farhan Muhammad, Scott Hanselman , Srivakumar

REFERENCE BOOKS:

1. Introducing Microsoft ASP .NET 2.0 Esposito PHI
2. Professional ADO.NET BipinJoshi,Donny Mack, Doug Seven , Fabio Claudio Ferracchiati, Jan D NarkiewiczWrox
3. Special Edition Using ASP.NET Richard Leineker Person Education
4. The Complete Reference ASP.NET Matthew MacDonald TMH
5. ASP.NET Black Book Dream Tech

UNIT – I

ASP.NET BASICS

1.1. INTRODUCTION TO ASP.NET

- ASP.NET is a web application framework designed and developed by Microsoft. ASP.NET is open source and a subset of the .NET Framework and successor of the classic ASP (Active Server Pages).
- With version 1.0 of the .NET Framework, it was first released in January 2002. The technology used before the year 2002 for developing web applications and services is Classic ASP. So before .NET and ASP.NET there was Classic ASP.
- ASP.NET is built on the CLR(Common Language Runtime) which allows the programmers to execute its code using any .NET language(C#, VB etc.). It is specially designed to work with HTTP and for web developers to create dynamic web pages, web applications, web sites, and web services as it provides a good integration of HTML, CSS, and JavaScript.

- .NET Framework is used to create a variety of applications and services like Console, Web, and Windows, etc. ASP.NET is only used to create web applications and web services. That's why we termed ASP.NET as a subset of the .NET Framework.
- Below table illustrates the ASP.Net Version History:

YEAR	VERSION
2002	1.0
2003	1.1
2005	2.0
2006	3.0
2007	3.5
2008	3.5 SP 1
2010	4.0
2012	4.5
2013	4.5.1
2014	4.5.2
2015	4.6
2015	4.6.1
2016	4.6.2
2017	4.7
2017	4.7.1

- **What is Web Application?**

A Web Application is an application installed only on the web server which is accessed by the users using a web browser like Microsoft Internet Explorer, Google Chrome, Mozilla firefox, Apple Safari etc. There are also some other technology like Java, PHP, Perl, Ruby on Rails, etc. This can be used to develop web applications. Web applications provide the cross-platform feature. The user needs only a web browser to access a web application. The web applications which are developed using the .Net Framework or its subsets required to execute under the Microsoft Internet Information Services (IIS) on the server side. The work of IIS is to provide the web application's generated HTML code result to the client browser which initiated the request as shown in the below diagram

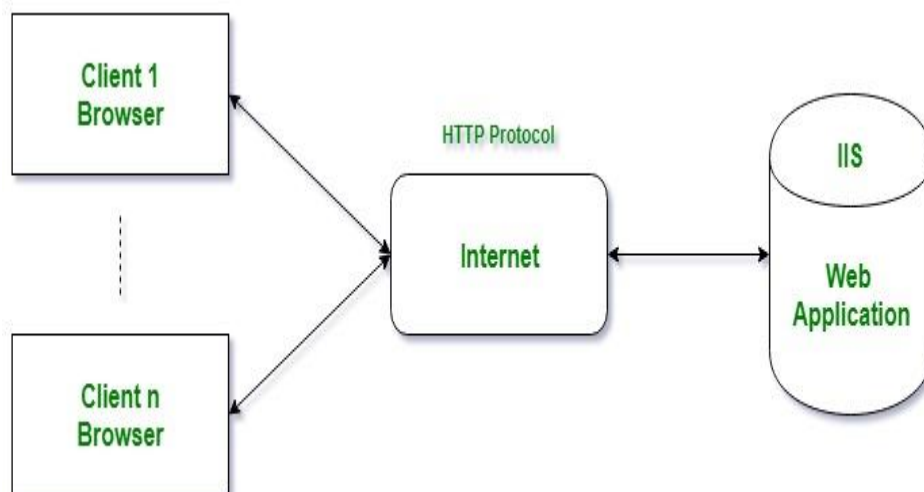


Figure 1.1 Working of IIS

1.2. NET FRAMEWORK ARCHITECTURE

- The .Net framework is a software development platform developed by Microsoft.
- The framework was meant to create applications, which would run on the Windows Platform.
- The first version of the .Net framework was released in the year 2002.
- The version was called .Net framework 1.0.

- The .Net framework has come a long way since then, and the current version is 4.7.1.
- The .Net framework can be used to create both **Form-based** and **Web-based** applications.
- Web services can also be developed using the .Net framework.
- The framework also supports various programming languages such as Visual Basic and C#. So developers can choose and select the language to develop the required application.
- The basic architecture of the .Net framework is as shown below.

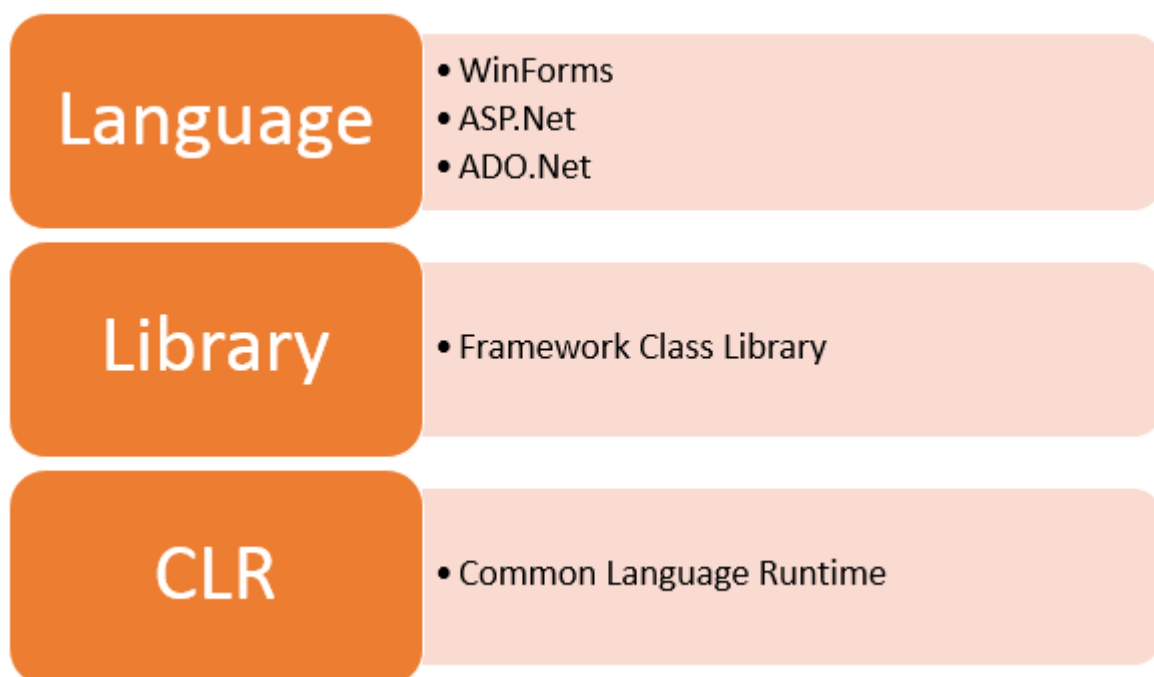


Figure 1.2 .Net Framework Architecture

.NET COMPONENTS

The architecture of the .Net framework is based on the following key components;

1.2.1. COMMON LANGUAGE RUNTIME (CLR)

- The .NET Framework provides a run-time environment called the **common language runtime**, which runs the code and provides services that make the development process easier.
- Compilers and tools expose the common language runtime's functionality and enable to write code that benefits from this **managed execution environment**.
- Code that has to develop with a language compiler that targets the runtime is called **managed code**; it benefits from features such as cross-language integration, cross-language exception handling, enhanced security, versioning and deployment support, a simplified model for component interaction, and debugging and profiling services.
- To enable the runtime to provide services to managed code, language compilers must emit **metadata** that describes the types, members, and references in the code.
- Metadata is stored with the code; every loadable common language runtime **portable executable** (PE) file contains metadata. The runtime uses metadata to locate and load classes, lay out instances in memory, resolve method invocations, generate native code, enforce security, and set run-time context boundaries.
- The runtime automatically handles object layout and manages references to objects, releasing them when they are no longer being used. Objects whose lifetimes are managed in this way are called managed data.
- Garbage collection eliminates memory leaks as well as some other common programming errors. If your code is managed, you can use managed data, unmanaged data, or both managed and unmanaged data in your .NET Framework application. Because language compilers supply their own types, such as primitive types, you might not always know (or need to know) whether your data is being managed.

- The common language runtime makes it easy to design components and applications whose objects interact across languages. Objects written in different languages can communicate with each other, and their behaviours can be tightly integrated. For example, you can define a class and then use a different language to derive a class from your original class or call a method on the original class. You can also pass an instance of a class to a method of a class written in a different language.
- This **cross-language integration** is possible because language compilers and tools that target the runtime use a common type system defined by the runtime, and they follow the runtime's rules for defining new types, as well as for creating, using, persisting, and binding to types.
- As part of their metadata, all managed components carry information about the components and resources they were built against. The runtime uses this information to ensure that your component or application has the specified versions of everything it needs, which makes your code less likely to break because of some unmet dependency.
- Registration information and state data are no longer stored in the registry where they can be difficult to establish and maintain. Instead, information about the types you define (and their dependencies) is stored with the code as metadata, making the tasks of component replication and removal much less complicated.
- Language compilers and tools expose the runtime's functionality in ways that are intended to be useful and intuitive to developers. This means that some features of the runtime might be more noticeable in one environment than in another.
- The runtime provides the following benefits:
 - Performance improvements.
 - The ability to easily use components developed in other languages.
 - Extensible types provided by a class library.

- Language features such as inheritance, interfaces, and overloading for object-oriented programming.
- Support for explicit free threading that allows creation of multithreaded, scalable applications.
- Support for structured exception handling.
- Support for custom attributes.
- Garbage collection.
- Use of delegates instead of function pointers for increased type safety and security.

1.2.2. COMMON LANGUAGE INFRASTRUCTURE (CLI)

The "Common Language Infrastructure" or CLI is a platform on which the .Net programs are executed.

The CLI has the following key features:

- **Exception Handling** - Exceptions are errors which occur when the application is executed.

Examples of exceptions are:

- If an application tries to open a file on the local machine, but the file is not present.
- If the application tries to fetch some records from a database, but the connection to the database is not valid.
- **Garbage Collection** - Garbage collection is the process of removing unwanted resources when they are no longer required.

Examples of garbage collection are

- A File handler which is no longer required. If the application has finished all operations on a file, then the file handle may no longer be required.

- The database connection is no longer required. If the application has finished all operations on a database, then the database connection may no longer be required.

- **Working with Various programming languages –**

As noted in an earlier section, a developer can develop an application in a variety of .Net programming languages.

1. Language - The first level is the programming language itself, the most common ones are VB.Net and C#.
2. Compiler – There is a compiler which will be separate for each programming language. So underlying the VB.Net language, there will be a separate VB.Net compiler. Similarly, for C#, you will have another compiler.
3. Common Language Interpreter – This is the final layer in .Net which would be used to run a .net program developed in any programming language. So the subsequent compiler will send the program to the CLI layer to run the .Net application.

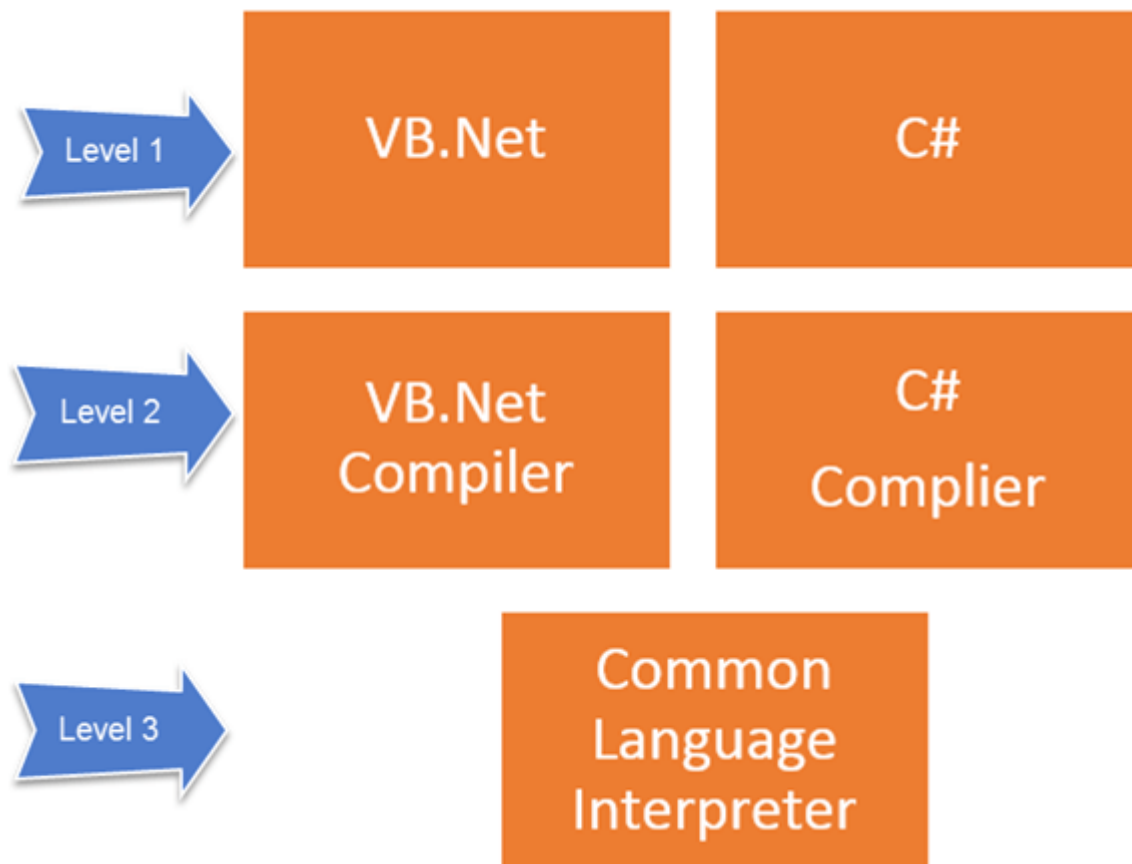


Figure 1.2.2 Working of CLI

1.2.3. BASIC CLASS LIBRARY (BCL)

The .NET Framework includes a set of standard class libraries. A class library is a collection of methods and functions that can be used for the core purpose.

For example, there is a class library with methods to handle all file-level operations. So there is a method which can be used to read the text from a file. Similarly, there is a method to write text to a file.

Most of the methods are split into either the `System.*` or `Microsoft.*` namespaces. (The asterisk * just means a reference to all of the methods that fall under the `System` or `Microsoft` namespace). A namespace is a logical separation of methods.

1.2.4. LANGUAGES

The types of applications that can be built in the .Net framework is classified broadly into the following categories.

- **WinForms** – This is used for developing Forms-based applications, which would run on an end user machine. Notepad is an example of a client-based application.
- **ASP.Net** – This is used for developing web-based applications, which are made to run on any browser such as Internet Explorer, Chrome or Firefox.
 - The Web application would be processed on a server, which would have Internet Information Services Installed.
 - Internet Information Services or IIS is a Microsoft component which is used to execute an ASP.Net application.
 - The result of the execution is then sent to the client machines, and the output is shown in the browser.
- **ADO.Net** – This technology is used to develop applications to interact with Databases such as Oracle or Microsoft SQL Server.

Microsoft always ensures that .Net frameworks are in compliance with all the supported Windows operating systems.

1.2.4. .NET FRAMEWORK DESIGN PRINCIPLE

The following design principle of the .Net framework is what makes it very relevant to create .Net based applications.

1. **Interoperability** - The .Net framework provides a lot of backward support. Suppose if you had an application built on an older version of the .Net framework, say 2.0. And if you tried to run the same application on a machine which had the higher version of the .Net framework, say 3.5. The application would still work. This is because with every release, Microsoft ensures that older framework versions gel well with the latest version.

2. **Portability**- Applications built on the .Net framework can be made to work on any Windows platform. And now in recent times, Microsoft is also envisioning to make Microsoft products work on other platforms, such as iOS and Linux.
3. **Security** - The .NET Framework has a good security mechanism. The inbuilt security mechanism helps in both validation and verification of applications. Every application can explicitly define their security mechanism. Each security mechanism is used to grant the user access to the code or to the running program.
4. **Memory management** - The Common Language runtime does all the work or memory management. The .Net framework has all the capability to see those resources, which are not used by a running program. It would then release those resources accordingly. This is done via a program called the "Garbage Collector" which runs as part of the .Net framework.

The garbage collector runs at regular intervals and keeps on checking which system resources are not utilized, and frees them accordingly.

5. **Simplified deployment** - The .Net framework also have tools, which can be used to package applications built on the .Net framework. These packages can then be distributed to client machines. The packages would then automatically install the application.

1.2.5. ASP.NET BASICS

ASP.NET provides an abstraction layer on top of HTTP on which the web applications are built. It provides high-level entities such as classes and components within an object-oriented paradigm. The key development tool for building ASP.NET applications and front ends is Visual Studio. In this tutorial, we work with Visual Studio 2008.

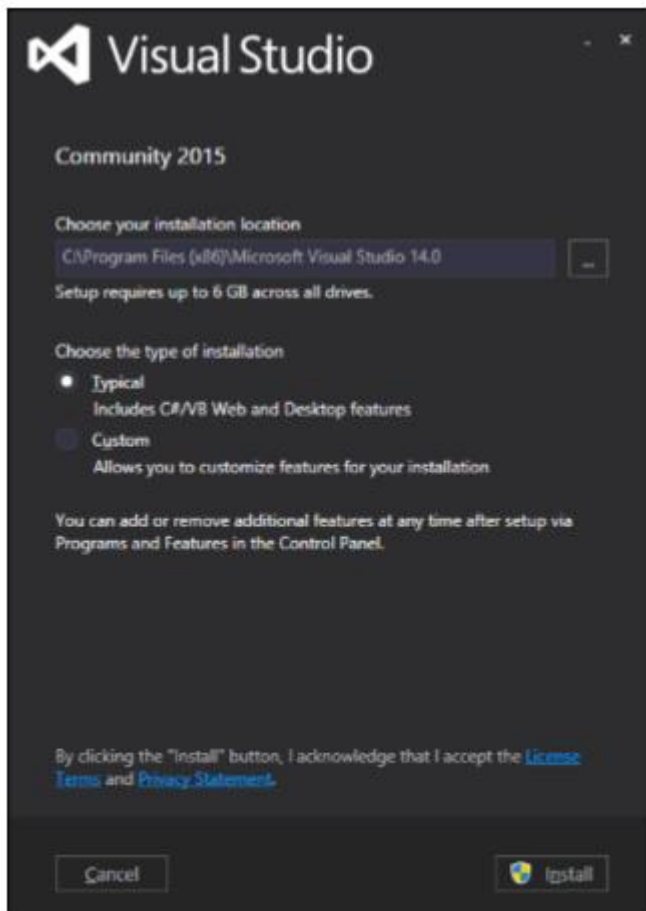
Visual Studio is an integrated development environment for writing, compiling, and debugging the code. It provides a complete set of

development tools for building ASP.NET web applications, web services, desktop applications, and mobile applications.

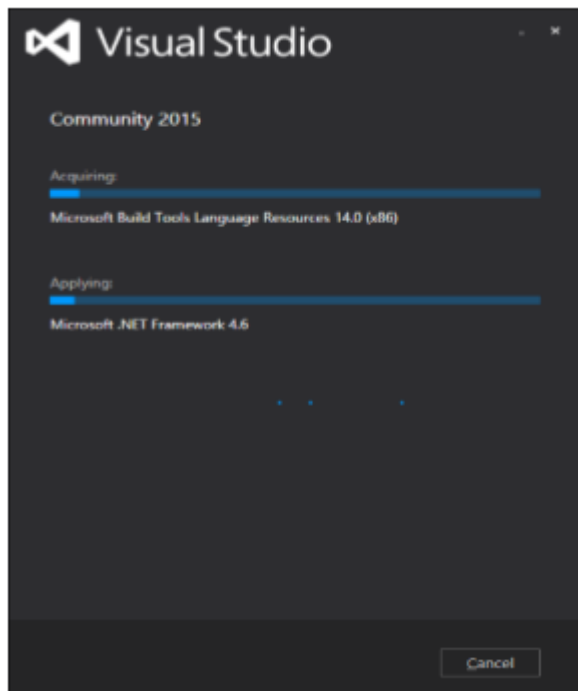
Installation

Microsoft provides a free version of visual studio which also contains SQL Server and it can be downloaded from www.visualstudio.com.

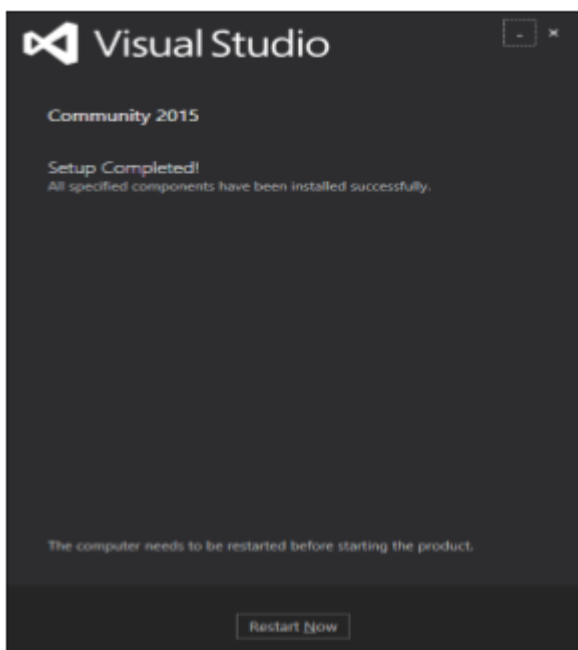
Step 1 – Once downloading is complete, run the installer. The following dialog will be displayed.



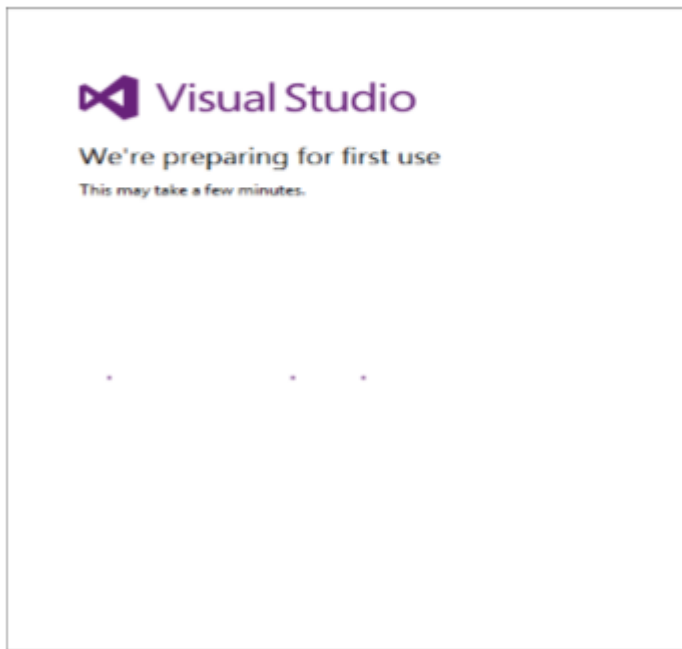
Step 2 – Click on the Install button and it will start the installation process.



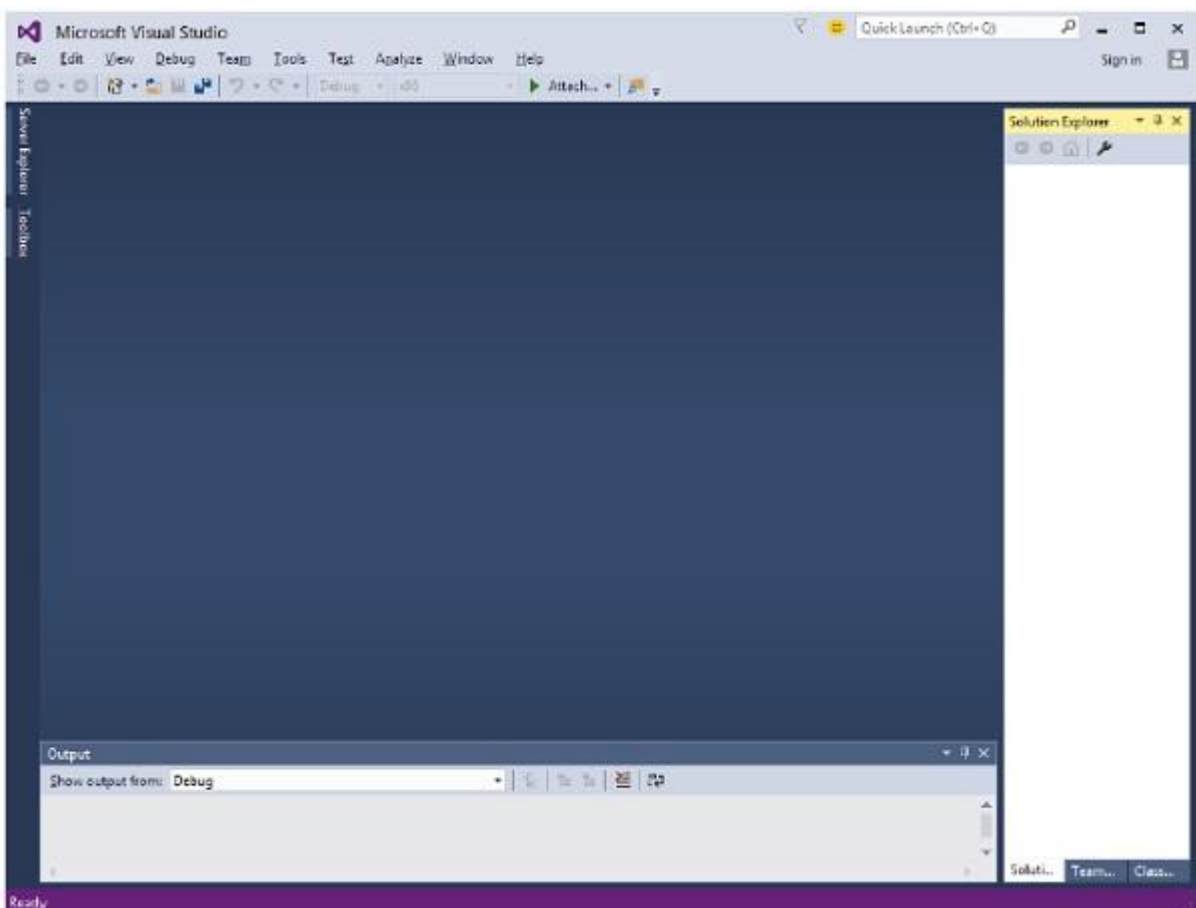
Step 3 – Once the installation process is completed successfully, you will see the following dialog. Close this dialog and restart your computer if required.



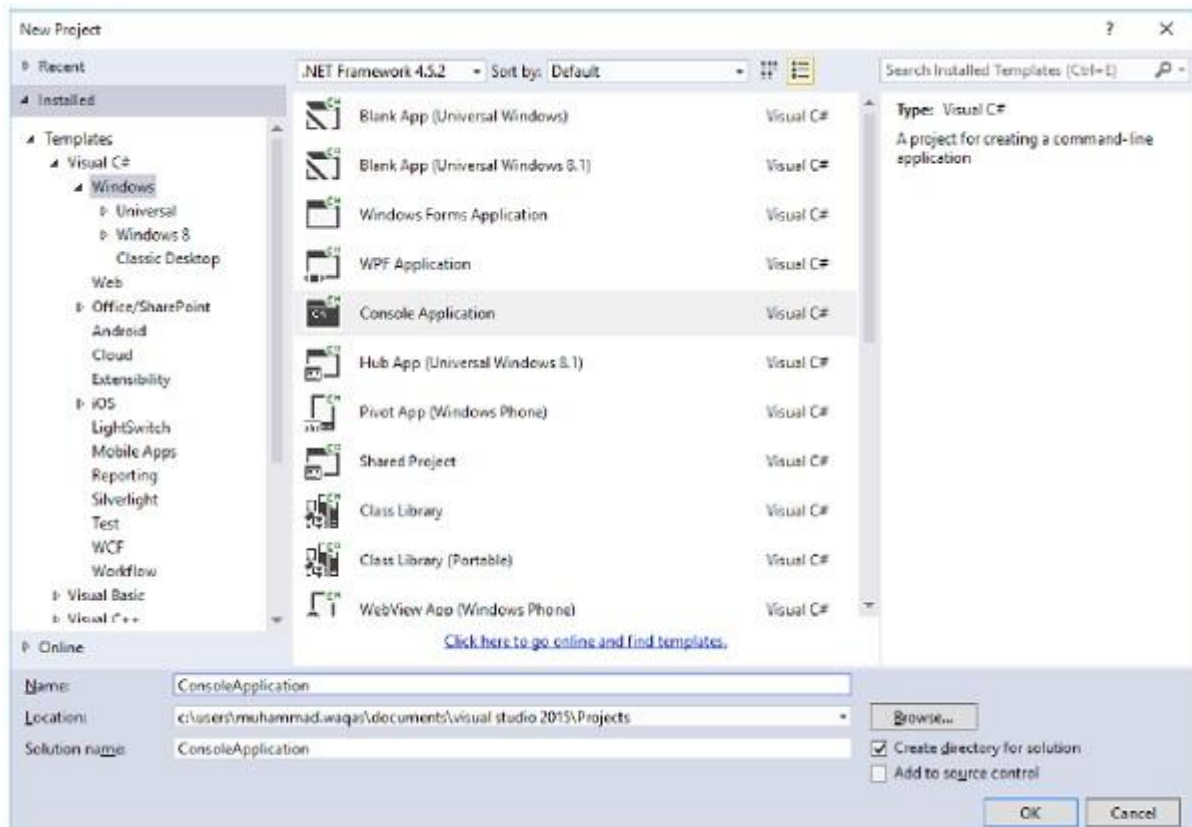
Step 4 – Open Visual Studio from start Menu which will open the following dialog. It will be a while for the first time for preparation.



Step 5 – Once all is done you will see the main window of Visual studio.

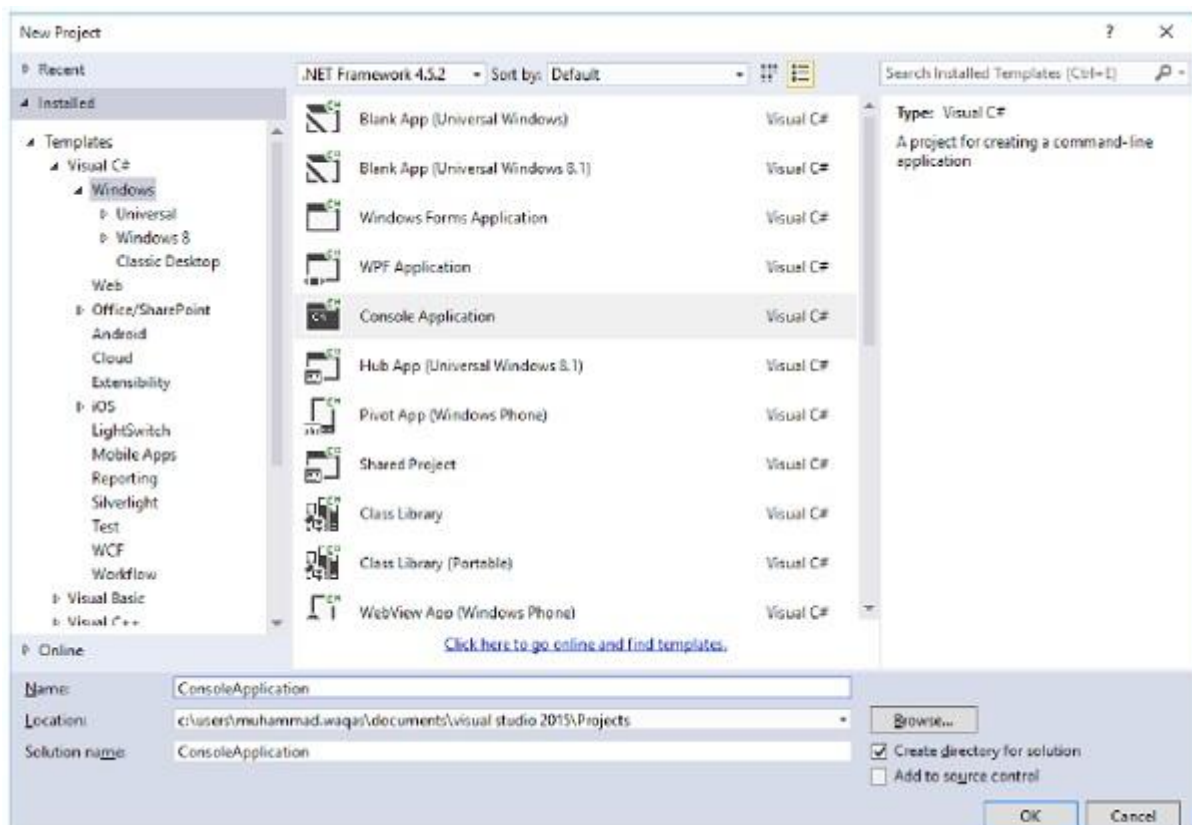


Let's create a new project from File → New → Project



The Visual Studio IDE

The new project window allows choosing an application template from the available templates.



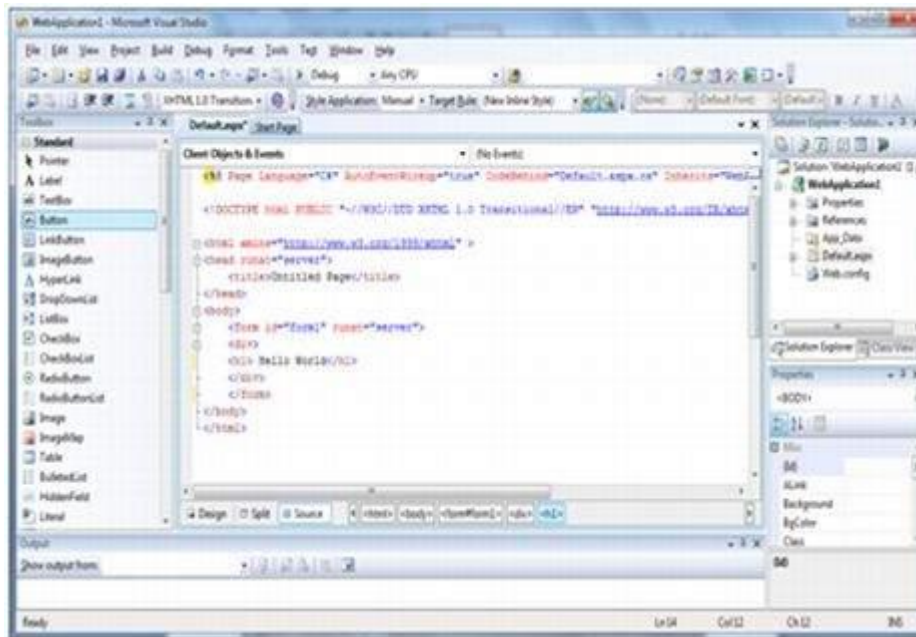
When you start a new web site, ASP.NET provides the starting folders and files for the site, including two files for the first web form of the site. The file named `Default.aspx` contains the HTML and asp code that defines the form, and the file named `Default.aspx.cs` (for C# coding) or the file named `Default.aspx.vb` (for VB coding) contains the code in the language you have chosen and this code is responsible for the actions performed on a form.

The primary window in the Visual Studio IDE is the Web Forms Designer window. Other supporting windows are the Toolbox, the Solution Explorer, and the Properties window. You use the designer to design a web form, to add code to the control on the form so that the form works according to your need, you use the code editor.

Working with Views and Windows

You can work with windows in the following ways:

- To change the Web Forms Designer from one view to another, click on the Design or source button.
- To close a window, click on the close button on the upper right corner and to redisplay, select it from the View menu.
- To hide a window, click on its Auto Hide button. The window then changes into a tab. To display again, click the Auto Hide button again.
- To change the size of a window, just drag it.



Adding Folders and Files to your Website

When a new web form is created, Visual Studio automatically generates the starting HTML for the form and displays it in Source view of the web forms designer. The Solution Explorer is used to add any other files, folders or any existing item on the web site.

- To add a standard folder, right-click on the project or folder under which you are going to add the folder in the Solution Explorer and choose New Folder.
- To add an ASP.NET folder, right-click on the project in the Solution Explorer and select the folder from the list.
- To add an existing item to the site, right-click on the project or folder under which you are going to add the item in the Solution Explorer and select from the dialog box.

Projects and Solutions

A typical ASP.NET application consists of many items: the web content files (.aspx), source files (.cs files), assemblies (.dll and .exe files), data source files (.mdb files), references, icons, user controls and miscellaneous other files and folders. All these files that make up the website are contained in a Solution. When a new website is created.

VB2008 automatically creates the solution and displays it in the solution explorer.

Solutions may contain one or more projects. A project contains content files, source files, and other files like data sources and image files. Generally, the contents of a project are compiled into an assembly as an executable file (.exe) or a dynamic link library (.dll) file.

Typically a project contains the following content files:

- Page file (.aspx)
- User control (.ascx)
- Web service (.asmx)
- Master page (.master)
- Site map (.sitemap)
- Website configuration file (.config)

Building and Running a Project

You can execute an application by:

- Selecting Start
- Selecting Start Without Debugging from the Debug menu,
- pressing F5
- Ctrl-F5

The program is built meaning, the .exe or the .dll files are generated by selecting a command from the Build menu.

ASP.NET - FIRST EXAMPLE

An ASP.NET page is made up of a number of server controls along with HTML controls, text, and images. Sensitive data from the page and the states of different controls on the page are stored in hidden fields that form the context of that page request.

ASP.NET runtime controls the association between a page instance and its state. An ASP.NET page is an object of the Page or inherited from it.

All the controls on the pages are also objects of the related control class inherited from a parent Control class. When a page is run, an instance of the object page is created along with all its content controls.

An ASP.NET page is also a server side file saved with the .aspx extension. The following code snippet provides a sample ASP.NET page explaining Page directives, code section and page layout written in C#:

```
<!-- directives -->
<% @Page Language="C#" %>

<!-- code section -->
<script runat="server">

    private void convertoupper(object sender, EventArgs e)
    {
        string str = mytext.Value;
        changed_text.InnerHtml = str.ToUpper();
    }
</script>

<!-- Layout -->
<html>
    <head>
        <title> Change to Upper Case </title>
    </head>

    <body>
        <h3> Conversion to Upper Case </h3>

        <form runat="server">
            <input runat="server" id="mytext" type="text" />
            <input runat="server" id="button1" type="submit" value="Enter..."
OnServerClick="convertoupper"/>

            <hr />
            <h3> Results: </h3>
            <span runat="server" id="changed_text" />
```

```

</form>

</body>

</html>

```

Copy this file to the web server root directory. Generally it is c:\inetput\wwwroot. Open the file from the browser to execute it and it generates following result:



Using Visual Studio IDE

Let us develop the same example using Visual Studio IDE. Instead of typing the code, you can just drag the controls into the design view:



The content file is automatically developed. All you need to add is the Button1_Click routine, which is as follows:

```

protected void Button1_Click(object sender, EventArgs e)
{
    string buf = TextBox1.Text;
}

```

```
changed_text.InnerHtml = buf.ToUpper();
}
```

The content file code is as given:

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

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

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

  <head runat="server">
    <title>
      Untitled Page
    </title>
  </head>

  <body>

    <form id="form1" runat="server">
      <div>

        <asp:TextBox ID="TextBox1" runat="server" style="width:224px">
        </asp:TextBox>

        <br />
        <br />

        <asp:Button ID="Button1" runat="server" Text="Enter..."
style="width:85px" onclick="Button1_Click" />
        <hr />
```

```

    <h3> Results: </h3>
    <span runat="server" id="changed_text" />

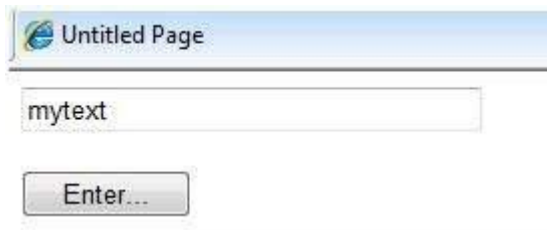
</div>
</form>

</body>

</html>

```

Execute the example by right clicking on the design view and choosing 'View in Browser' from the popup menu. This generates the following result:



Results:

MYTEXT

1.3. ASP.NET PAGE STRUCTURE

The following is a list of the important elements of an ASP.NET page:

1. Directives
2. Code declaration blocks
3. ASP.NET controls
4. Code render blocks
5. Server-side comments
6. Server-side include directives
7. Literal text and HTML tags

Each element is discussed in the following sections.

1.3.5. Directives

A *directive* controls how an ASP.NET page is compiled. The beginning of a directive is marked with the characters `<%@` and the end of a directive is marked with the characters `%>`. A directive can appear anywhere within a page. By convention, however, a directive typically appears at the top of an ASP.NET page.

There are several types of directives that you can add to an ASP.NET page. Two of the most useful types are page and import directives.

i) Page Directives

Page directives used to specify the default programming language for a page. Page directives can also be used to enable tracing and debugging for a page. To change the default programming language of an ASP.NET page from Visual Basic to C#, for example, you would use the following page directive:

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

The keyword `Page` in a page directive is optional. The following two directives are equivalent:

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

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

Another extremely useful page directive is the `Trace` directive. If you enable tracing for a page, additional information about the execution of the page is displayed along with the content of the page. To enable runtime error messages to be displayed on a page, use the `Debug` directive. To display errors in your ASP.NET page, include the following directive:

```
<%@ Page Debug="True" %>
```

When this directive is included, if an error is encountered when processing the page, the error is displayed. In most cases, the source code for the exact statement can be viewed that generated the error. To enable both tracing

and debugging for a page, combine the directives like this (the order of Debug and Trace is not important):

```
<%@ Page Debug="True" Trace="True" %>
```

ii. Import Directives

By default, only certain name spaces are automatically imported into an ASP.NET page. If you want to refer to a class that isn't a member of one of the default namespaces, then you must explicitly import the namespace of the class or you must use the fully qualified name of the class.

For example, suppose that you want to send an email from an ASP.NET page by using the Send method of the SmtpMail class. The SmtpMail class is contained in the System.Web.Mail namespace. This is not one of the default namespaces imported into an ASP.NET page.

The easiest way to use the SmtpMail class is to add an Import directive to your ASP.NET page to import the necessary namespace. The page in Listing 12 illustrates how to import the System.Web.Mail namespace and send an email message.

ImportNamespace.aspx

```
<%@ Import Namespace="System.Web.Mail" %>
<Script Runat="Server">
Sub Page_Load
  Dim mailMessage As SmtpMail
  mailMessage.Send( _
    "bob@somewhere.com", _
    "alice@somewhere.com", _
    "Sending Mail!", _
    "Hello!" )
End Sub
```

```

</Script>
<html>
<head><title>ImportNamespace.aspx</title></head>
<body>
<h2>Email Sent!</h2>
</body>
</html>

```

The first line in Listing 12 contains an import directive. Without the import directive, the page would generate an error because it would not be able to resolve the `SmtpMail` class.

Instead of importing the `System.Web.Mail` namespace with the import directive, you could alternatively use its fully qualified name. In that case, you would declare an instance of the class like this:

```
Dim mailMessage As System.Web.Mail.SmtpMail
```

1.3.2 Code Declaration Blocks

A *code declaration block* contains all the application logic for your ASP.NET page and all the global variable declarations, subroutines, and functions. It must appear within a `<Script Runat="Server">` tag. Subroutines and functions can be declared only within a code declaration block. An error if an attempt to define a function or subroutine in any other section of an ASP.NET page is made.

```

<Script runat="Server">
Sub mySub
...subroutine code
End Sub
</Script>

```

The `<Script Runat="Server">` tag accepts two optional attributes. To Specify the programming language within the `<Script>` tag by including a language attribute like this:

```
<Script Language="C#" Runat="Server">
```

If no language is specified in a page, the language defaults to Visual Basic.

1.3.3. ASP.NET Controls

ASP.NET controls can be freely interspersed with the text and HTML content of a page. The only requirement is that the controls should appear within a `<form Runat="Server">` tag. And, for certain tags such as `` and `<ASP:Label Runat="Server"/>`, this requirement can be ignored without any dire consequences.

One significant limitation of ASP.NET pages is that they can contain only one `<form Runat="Server">` tag.

1.3.4. Code Render Blocks

If there is a need to execute code within the HTML or text content of ASP.NET page, it can be done so within *code render blocks*. The two types of code render blocks are *inline code* and *inline expressions*. Inline code executes a statement or series of statements. This type of code begins with the characters `<%` and ends with the characters `%>`.

Inline expressions, on the other hand, display the value of a variable or method (this type of code is shorthand for `Response.Write`). Inline expressions begin with the characters `<%=` and end with the characters `%>`. The ASP.NET page in Listing 15 illustrates how to use both inline code and inline expressions in an ASP.NET page.

CodeRender.aspx

```
<Script Runat="Server">
Dim strSomeText As String
Sub Page_Load
```

```

    strSomeText = "Hello!"
End Sub
</Script>

<html>
<head><title>CodeRender.aspx</title></head>
<body>
<form Runat="Server">
The value of strSomeText is:
<%=strSomeText%>
<p>
<% strSomeText = "Goodbye!" %>
The value of strSomeText is:
<%=strSomeText%>
</form>
</body>
</html>

```

Notice that you can use variables declared in the code declaration block within the code render block. However, the variable has to be declared with page scope. The variable could not, for example, be declared within the Page_Load subroutine.

1.3.5. Server-side Comments

Comments can be added to the ASP.NET pages by using *server-side comment* blocks. The beginning of a server-side comment is marked with the characters `<%--` and the end of the comment is marked with the characters `-->`. Server-side comments can be added to a page for the purposes of documentation.

ServerComments.aspx

```

<Script Runat="Server">
  Dim strSomeText As String
  Sub Page_Load
    strSomeText = "Hello!"
  End Sub
</Script>

<html>
<head><title>ServerComments.aspx</title></head>
<body>

<form Runat="Server">

<%--
This is inside the comments
<asp:Label Text="hello!" Runat="Server" />
<%= strSomeText %>
--%>
This is outside the comments
</form>
</body>
</html>

```

1.3.6. Server-side Include Directives

Files can be included in an ASP.NET page by using one of the two forms of the *server-side include directive*. If you want to include a file that is located in the same directory or in a subdirectory of the page including the file, you would use the following directive:

```
<!-- #INCLUDE file="includefile.aspx" -->
```

Alternatively, you can include a file by supplying the full virtual path. For example, if you have a subdirectory named myDirectory under the wwwroot directory, you can include a file from that directory like this:

```
<!-- #INCLUDE virtual="/myDirectory/includefile.aspx" -->
```

The include directive is executed before any of the code in a page. One implication is that you cannot use variables to specify the path to the file that you want to include. For example, the following directive would generate an error:

```
<!-- #INCLUDE file="<%=myVar%>" -->
```

1.3.7. Literal Text and HTML Tags

The final type of element that you can include in an ASP.NET page is HTML content. The static portion of your page is built with plain old HTML tags and text. HTML content in a page is represented with the LiteralControl class. You can use the Text property of the LiteralControl class to manipulate the pure HTML portion of an ASP.NET page.

1.4. ASP.NET PAGE LIFECYCLE

In ASP.NET, a web page has execution lifecycle that includes various phases. These phases include initialization, instantiation, restoring and maintaining state etc. it is required to understand the page lifecycle so that we can put custom code at any stage to perform our business logic.

1.4.3. PAGE LIFECYCLE STAGES

The following table contains the lifecycle stages of ASP.NET web page.

Stage	Description
Page request	This stage occurs before the lifecycle begins. When a page is requested by the user, ASP.NET parses and compiles that page.
Start	In this stage, page properties such as Request and response are set. It also determines the Request type.
Initialization	In this stage, each control's UniqueID property is set. Master page is applied to the page.
Load	During this phase, if page request is postback, control properties are loaded with information.
Postback event handling	In this stage, event handler is called if page request is postback. After that, the Validate method of all validator controls is called.
Rendering	Before rendering, view state is saved for the page and all controls. During the rendering stage, the page calls the Render method for each control, providing a text writer that writes its output to the OutputStream object of the page's Response property.
Unload	At this stage the requested page has been fully rendered and is ready to terminate. At this stage all properties are unloaded and cleanup is performed.

A requested page first loaded into the server memory after that processes and sent to the browser. At last it is unloaded from the server memory. ASP.NET provides methods and events at each stage of the page lifecycle

that we can use in our application. In the following table, we are tabled events.

1.4.4. ASP.NET LIFE CYCLE EVENTS

Page Event	Typical Use
PreInit	This event is raised after the start stage is complete and before the initialization stage.
Init	This event occurs after all controls have been initialized. We can use this event to read or initialize control properties.
InitComplete	This event occurs at the end of the page's initialization stage. We can use this event to make changes to view state that we want to make sure are persisted after the next postback.
PreLoad	This event is occurs before the post back data is loaded in the controls.
Load	This event is raised for the page first time and then recursively for all child controls.
Control events	This event is used to handle specific control events such as Button control' Click event.
LoadComplete	This event occurs at the end of the event-handling stage.

	We can use this event for tasks that require all other controls on the page be loaded.
PreRender	This event occurs after the page object has created all controls that are required in order to render the page.
PreRenderComplete	This event occurs after each data bound control whose DataSourceID property is set calls its DataBind method.
SaveStateComplete	It is raised after view state and control state have been saved for the page and for all controls.
Render	This is not an event; instead, at this stage of processing, the Page object calls this method on each control.
Unload	This event raised for each control and then for the page.

1.5. HTML SERVER CONTROLS

HTML server controls are HTML elements that contain attributes to accessible at server side. By default, HTML elements on an ASP.NET Web page are not available to the server. These components are treated as simple text and pass through to the browser. We can convert an HTML element to server control by adding a **runat="server"** and an **id** attribute to the component.

Now, we can easily access it at code behind.

Example

```
<input id="UserName" type="text" size="50"runat="server" />
```

All the HTML Server controls can be accessed through the **Request** object.

HTML Components

The following table contains commonly used HTML components.

Controls Name	Description
Button	It is used to create HTML button.
Reset Button	It is used to reset all HTML form elements.
Submit Button	It is used to submit form data to the server.
Text Field	It is used to create text input.
Text Area	It is used to create a text area in the html form.
File	It is used to create a input type = "file" component which is used to upload file to the server.
Password	It is a password field which is used to get password from the user.
CheckBox	It creates a check box that user can select or clear.
Radio Button	A radio field which is used to get user choice.
Table	It allows us to present information in a tabular format.
Image	It displays an image on an HTML form
ListBox	It displays a list of items to the user. You can set the size

	from two or more to specify how many items you wish to show.
Dropdown	It displays a list of items to the user in a dropdown list.
Horizontal Rule	It displays a horizontal line across the HTML page.

Example

Here, an HTML server control has been implemented in the form.

// **htmlcontrolsexample.aspx**

1. `<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="htmlcontrolsexample.aspx.cs" Inherits="asp.netexample.htmlcontrolsexample"%>`
2. `<!DOCTYPE html>`
3. `<html xmlns="http://www.w3.org/1999/xhtml">`
4. `<head runat="server">`
5. `<title></title>`
6. `</head>`
7. `<body>`
8. `<form id="form1" runat="server">`
9. `<div>`
10. `<input id="Text1" type="text" runat="server"/>`
11. `<asp:Button ID="Button1" runat="server" Text="Button" OnClick="Button1_Click"/>`
12. `</div>`
13. `</form>`
14. `</body>`
15. `</html>`

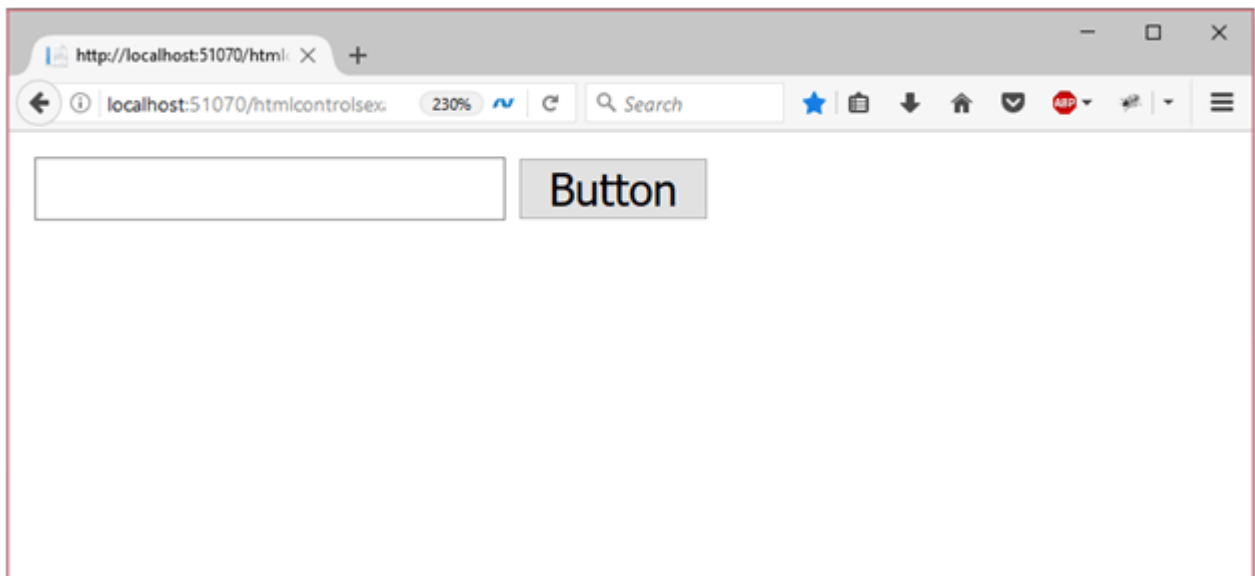
This application contains a code behind file.

// **htmlcontrolsexample.aspx.cs**

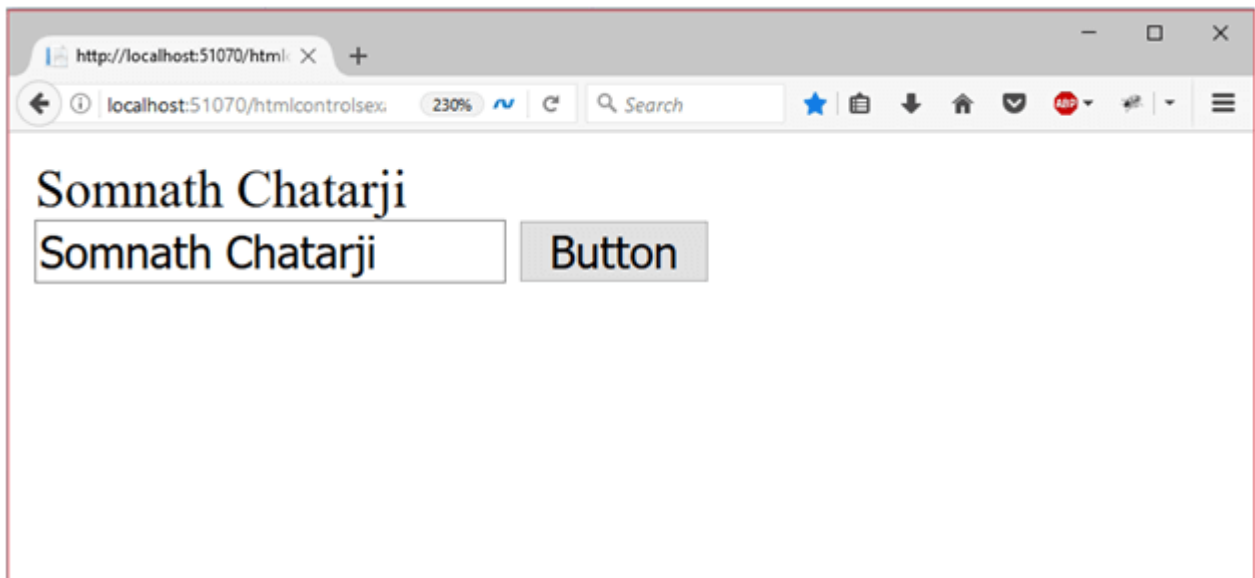
1. `using System;`
2. `namespace asp.netexample`

```
3.  {
4.    public partial class htmlcontrolsexample : System.Web.UI.Page
5.    {
6.        protected void Page_Load(object sender, EventArgs e)
7.        {
8.        }
9.        protected void Button1_Click(object sender, EventArgs e)
10.       {
11.           string a = Request.Form["Text1"];
12.           Response.Write(a);
13.       }
14.   }
15. }
```

Output:



When we click the button after entering text, it responses back to client.



1.6. WEB SERVER CONTROLS

ASP.NET provides web forms controls that are used to create HTML components. These controls are categorized as server and client based. The following table contains the server controls for the web forms.

Control Name	Applicable Events	Description
Label	None	It is used to display text on the HTML page.
TextBox	TextChanged	It is used to create a text input in the form.
Button	Click, Command	It is used to create a button.
LinkButton	Click, Command	It is used to create a button that looks similar to the hyperlink.
ImageButton	Click	It is used to create an

		imagesButton. Here, an image works as a Button.
Hyperlink	None	It is used to create a hyperlink control that responds to a click event.
DropDownList	SelectedIndexChanged	It is used to create a dropdown list control.
ListBox	SelectedIndexCnhaged	It is used to create a ListBox control like the HTML control.
DataGrid	CancelCommand, EditCommand, DeleteCommand, ItemCommand, SelectedIndexChanged, PageIndexChanged, SortCommand, UpdateCommand, ItemCreated, ItemDataBound	It used to create a frid that is used to show data. We can also perform paging, sorting, and formatting very easily with this control.
DataList	CancelCommand, EditCommand, DeleteCommand, ItemCommand, SelectedIndexChanged, UpdateCommand, ItemCreated,	It is used to create datalist that is non-tabular and used to show data.

	ItemDataBound	
CheckBox	CheckChanged	It is used to create checkbox.
CheckBoxList	SelectedIndexChanged	It is used to create a group of check boxes that all work together.
RadioButton	CheckChanged	It is used to create radio button.
RadioButtonList	SelectedIndexChanged	It is used to create a group of radio button controls that all work together.
Image	None	It is used to show image within the page.
Panel	None	It is used to create a panel that works as a container.
Placeholder	None	It is used to set placeholder for the control.
Calendar	SelectionChanged, VisibleMonthChanged, DayRender	It is used to create a calendar. We can set the default date, move forward and backward etc.
AdRotator	AdCreated	It allows us to specify a

		list of ads to display. Each time the user re-displays the page.
Table	None	It is used to create table.
XML	None	It is used to display XML documents within the HTML.
Literal	None	It is like a label in that it displays a literal, but allows us to create new literals at runtime and place them into this control.

1.7. ASP.NET VALIDATION CONTROLS

To perform validation, ASP.NET provides controls that automatically check user input and require no code. We can also create custom validation for our application. Following are the validation controls

Validator	Description
CompareValidator	It is used to compare the value of an input control against a value of another input control.
RangeValidator	It evaluates the value of an input control to check the specified range.
RegularExpressionValidator	It evaluates the value of an input control to

	determine whether it matches a pattern defined by a regular expression.
RequiredFieldValidator	It is used to make a control required.
ValidationSummary	It displays a list of all validation errors on the Web page.

1.7.1 ASP.NET CompareValidator Control

This validator evaluates the value of an input control against another input control on the basis of specified operator. We can use comparison operators like: less than, equal to, greater than etc.

CompareValidator Properties

Property	Description
AccessKey	It is used to set keyboard shortcut for the control.
TabIndex	The tab order of the control.
BackColor	It is used to set background color of the control.
BorderColor	It is used to set border color of the control.
BorderWidth	It is used to set width of border of the control.
Font	It is used to set font for the control text.
ForeColor	It is used to set color of the control text.
Text	It is used to set text to be shown for the control.
ToolTip	It displays the text when mouse is over the

	control.
Visible	To set visibility of control on the form.
Height	It is used to set height of the control.
Width	It is used to set width of the control.
ControlToCompare	It takes ID of control to compare with.
ControlToValidate	It takes ID of control to validate.
ErrorMessage	It is used to display error message when validation failed.
Operator	It is used set comparison operator.

Example

Here, in the following example, we are validating user input by using CompareValidator controller. Source code of the example is given below.

// compare_validator_demo.aspx

1. <%@ Page Language="C#" AutoEventWireup="true" CodeBehind="compare_validator_demo.aspx.cs"
2. Inherits="asp.netexample.compare_validator_demo" %>
3. <!DOCTYPE html>
4. <**html** xmlns="http://www.w3.org/1999/xhtml">
5. <**head** runat="server">
6. <**title**></**title**>
7. <**style** type="text/css">
8. .auto-style1 {
9. width: 100%;
10. }
11. .auto-style2 {
12. height: 26px;
13. }

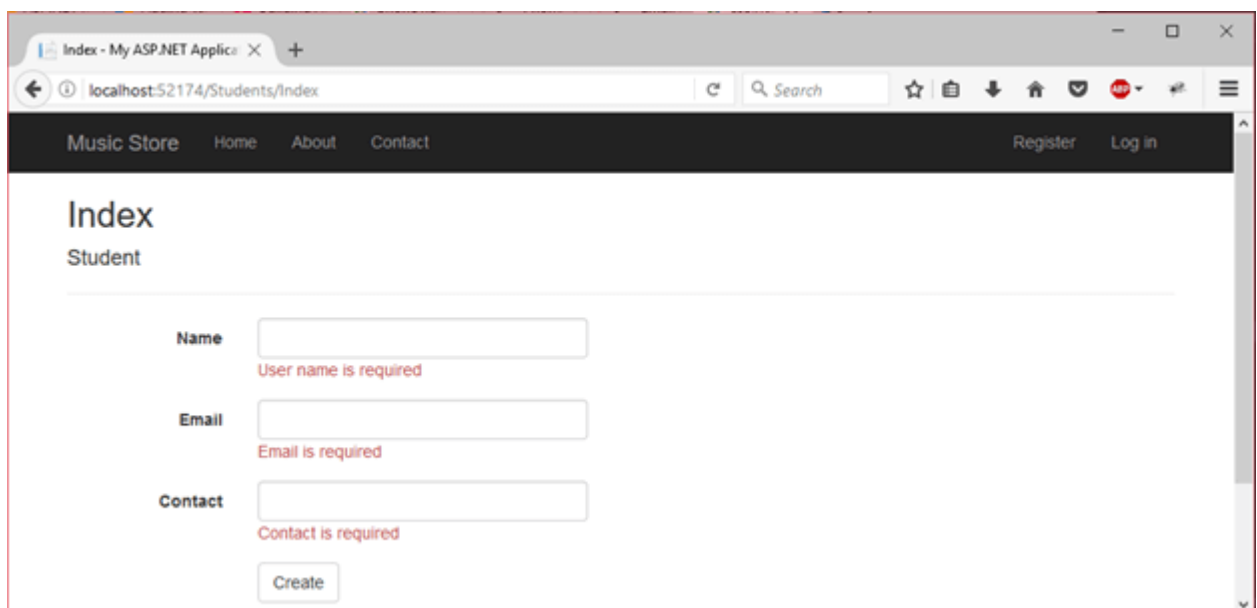
```

14.     .auto-style3 {
15.     height: 26px;
16.     width: 93px;
17.     }
18.     .auto-style4 {
19.     width: 93px;
20.     }
21. </style>
22. </head>
23. <body>
24.     <form id="form1" runat="server">
25.     <table class="auto-style1">
26.     <tr>
27.     <td class="auto-style3">
28.         First value</td>
29.     <td class="auto-style2">
30.     <asp:TextBox ID="firstval" runat="server" required="true"></asp:Text
    Box>
31.     </td>
32.     </tr>
33.     <tr>
34.     <td class="auto-style4">
35.         Second value</td>
36.     <td>
37.     <asp:TextBox ID="secondval" runat="server"></asp:TextBox>
38.         It should be greater than first value</td>
39.     </tr>
40.     <tr>
41.     <td class="auto-style4"></td>
42.     <td>
43.     <asp:Button ID="Button1" runat="server" Text="save"/>
44.     </td>
45.     </tr>

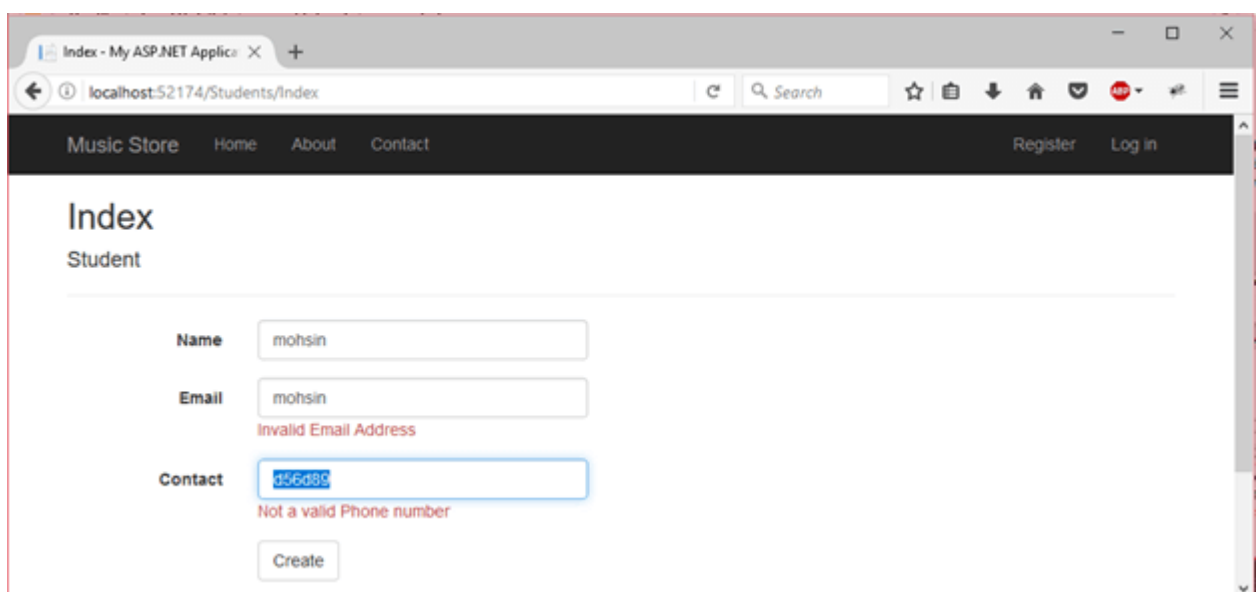
```

46. `</table>`
47. `< asp:CompareValidator ID="CompareValidator1" runat="server" ControlToCompare="secondval"`
48. `ControlToValidate="firstval" Display="Dynamic" ErrorMessage="Enter valid value" ForeColor="Red"`
49. `Operator="LessThan" Type="Integer"></asp:CompareValidator>`
50. `</form>`
51. `</body>`
52. `</html>`

Output:



The screenshot shows a web browser window with the address bar displaying 'localhost:52174/Students/Index'. The page has a dark navigation bar with links for 'Music Store', 'Home', 'About', 'Contact', 'Register', and 'Log in'. The main content area is titled 'Index' and 'Student'. Below this, there is a registration form with three input fields: 'Name', 'Email', and 'Contact'. Each field has a red error message below it: 'User name is required', 'Email is required', and 'Contact is required' respectively. A 'Create' button is located at the bottom of the form.



The screenshot shows the same web browser window, but the form fields are now filled. The 'Name' field contains 'mohsin', the 'Email' field contains 'mohsin', and the 'Contact' field contains '95688'. The 'Email' field has a red error message 'Invalid Email Address' below it. The 'Contact' field has a red error message 'Not a valid Phone number' below it. The 'Create' button is still present at the bottom of the form.

1.7.2.ASP.NET RangeValidator Control

This validator evaluates the value of an input control to check that the value lies between specified ranges. It allows us to check whether the user input is between a specified upper and lower boundary. This range can be numbers, alphabetic characters and dates.

The **ControlToValidate** property is used to specify the control to validate. The **MinimumValue** and **MaximumValue** properties are used to set minimum and maximum boundaries for the control.

RangeValidator Properties

Property	Description
AccessKey	It is used to set keyboard shortcut for the control.
TabIndex	The tab order of the control.
BackColor	It is used to set background color of the control.
BorderColor	It is used to set border color of the control.
BorderWidth	It is used to set width of border of the control.
Font	It is used to set font for the control text.
ForeColor	It is used to set color of the control text.
Text	It is used to set text to be shown for the control.
ToolTip	It displays the text when mouse is over the control.
Visible	To set visibility of control on the form.

Height	It is used to set height of the control.
Width	It is used to set width of the control.
ControlToValidate	It takes ID of control to validate.
ErrorMessage	It is used to display error message when validation failed.
Type	It is used to set datatype of the control value.
MaximumValue	It is used to set upper boundary of the range.
MinimumValue	It is used to set lower boundary of the range.

Example

In the following example, we are using **RangeValidator** to validate user input in specified range.

// RangeValidator.aspx

1. <%@ Page Language="C#" AutoEventWireup="true" CodeBehind="RangeValidator.aspx.cs"
2. Inherits="asp.netexample.RangeValidator" %>
3. <!DOCTYPE html>
4. <**html** xmlns="http://www.w3.org/1999/xhtml">
5. <**head** runat="server">
6. <**title**></**title**>
7. <**style** type="text/css">
8. .auto-style1 {
9. height: 82px;
10. }
11. .auto-style2 {
12. width: 100%;
13. }
14. .auto-style3 {
15. width: 89px;

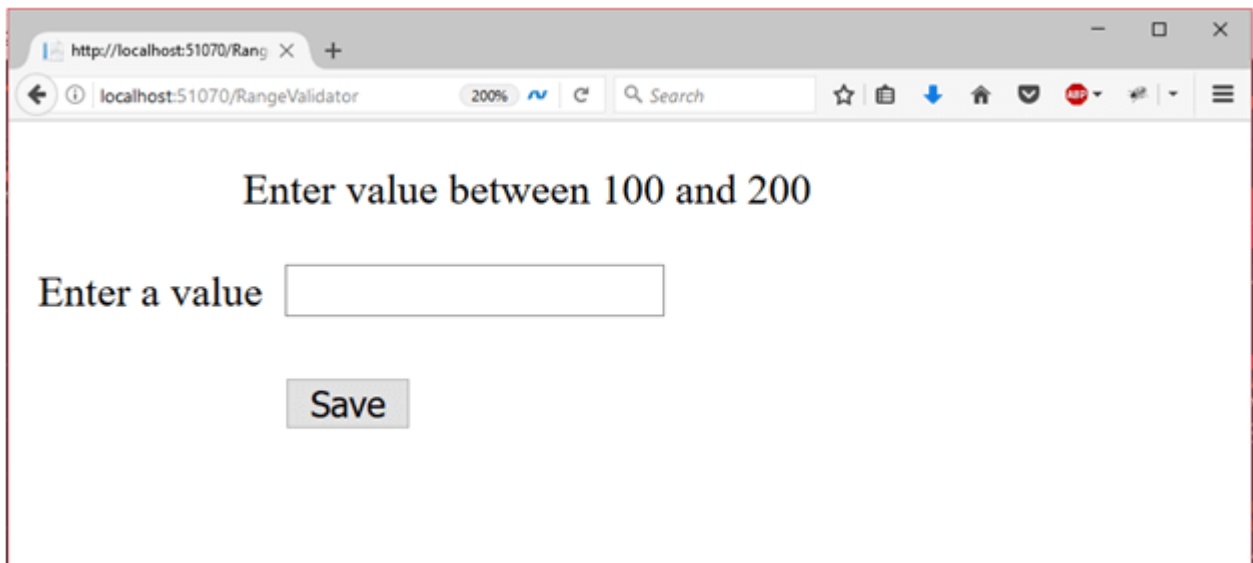
```

16.         }
17.     .auto-style4 {
18.     margin-left: 80px;
19.     }
20. </style>
21. </head>
22. <body>
23.     <form id="form1" runat="server">
24.     <div class="auto-style1">
25.     <p class="auto-style4">
26.         Enter value between 100 and 200<br/>
27.     </p>
28.     <table class="auto-style2">
29.     <tr>
30.     <td class="auto-style3">
31.     <asp:Label ID="Label2" runat="server" Text="Enter a value"></asp:La
    bel>
32.     </td>
33.     <td>
34.     <asp:TextBox ID="uesrInput"runat="server"></asp:TextBox>
35.     <asp:RangeValidator ID="RangeValidator1" runat="server" ControlTo
        Validate="uesrInput"
36.     ErrorMessage="Enter value in specified range" ForeColor="Red" Maxim
        umValue="199" MinimumValue="101"
37.     SetFocusOnError="True"Type=" Integer"></asp:RangeValidator>
38.     </td>
39.     </tr>
40.     <tr>
41.     <td class="auto-style3"> </td>
42.     <td>
43.     <br/>
44.     <asp:Button ID="Button2" runat="server" Text="Save"/>
45.     </td>

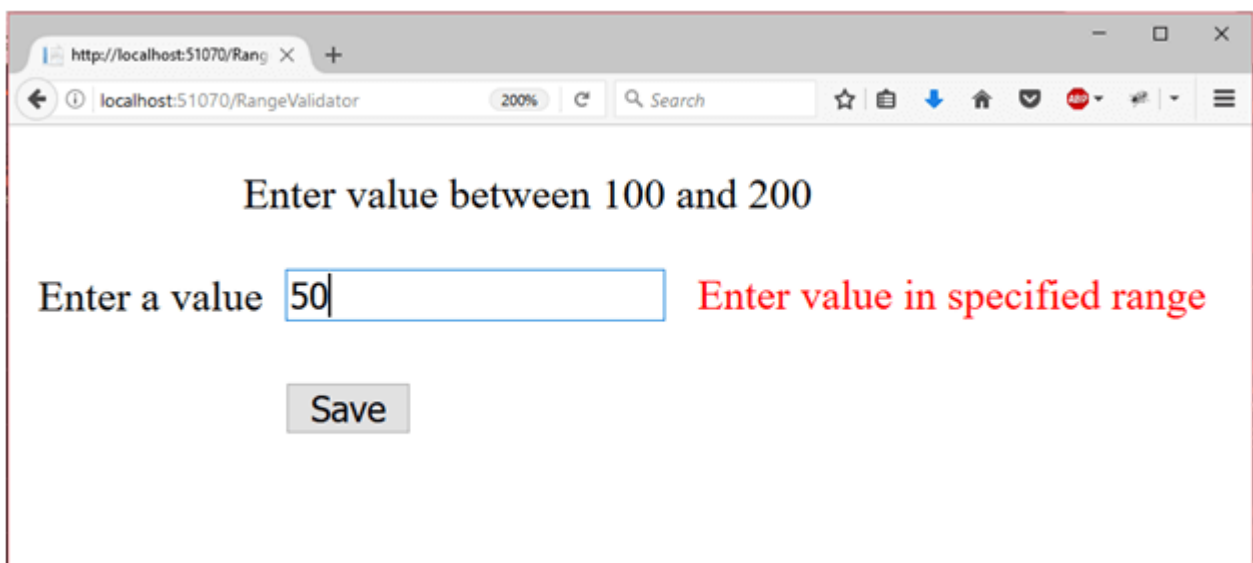
```


46. `</tr>`
47. `</table>`
48. `
`
49. `
`
50. `</div>`
51. `</form>`
52. `</body>`
53. `</html>`

Output:



It throws an error message when the input is not in range.



1.7.3.ASP.NET RegularExpressionValidator Control

This validator is used to validate the value of an input control against the pattern defined by a regular expression. It allows us to check and validate predictable sequences of characters like: e-mail address, telephone number etc. The **ValidationExpression** property is used to specify the regular expression, this expression is used to validate input control.

RegularExpression Properties

Property	Description
AccessKey	It is used to set keyboard shortcut for the control.
BackColor	It is used to set background color of the control.
BorderColor	It is used to set border color of the control.
Font	It is used to set font for the control text.
ForeColor	It is used to set color of the control text.
Text	It is used to set text to be shown for the control.
ToolTip	It displays the text when mouse is over the control.
Visible	To set visibility of control on the form.
Height	It is used to set height of the control.
Width	It is used to set width of the control.
ErrorMessage	It is used to set error message that display when validation fails.

ControlToValidate	It takes ID of control to validate.
ValidationExpression	It is used to set regular expression to determine validity.

The regular expression is set in the ValidationExpression property.

The following table summarizes the commonly used syntax constructs for regular expressions:

Character Escapes	Description
\b	Matches a backspace.
\t	Matches a tab.
\r	Matches a carriage return.
\v	Matches a vertical tab.
\f	Matches a form feed.
\n	Matches a new line.
\	Escape character.

Apart from single character match, a class of characters could be specified that can be matched, called the metacharacters.

Metacharacters	Description
.	Matches any character except \n.
[abcd]	Matches any character in the set.
[^abcd]	Excludes any character in the set.
[2-7a-mA-M]	Matches any character specified in the range.
\w	Matches any alphanumeric character and underscore.
\W	Matches any non-word character.
\s	Matches whitespace characters like, space, tab, new line etc.
\S	Matches any non-whitespace character.
\d	Matches any decimal character.
\D	Matches any non-decimal character.

Quantifiers could be added to specify number of times a character could appear.

Quantifier	Description
*	Zero or more matches.
+	One or more matches.
?	Zero or one matches.
{N}	N matches.
{N,}	N or more matches.
{N,M}	Between N and M matches.

Example

Here, in the following example, we are explaining how to use RegularExpressionValidator control to validate the user input against the given pattern.

// RegularExpressionDemo.aspx

1. `<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="RegularExpressionDemo.aspx.cs"`
2. `Inherits="asp.netexample.RegularExpressionDemo" %>`
3. `<!DOCTYPE html>`
4. `<html xmlns="http://www.w3.org/1999/xhtml">`
5. `<head runat="server">`
6. `<title></title>`
7. `</head>`
8. `<body>`
9. `<form id="form1" runat="server">`
10. `<div>`

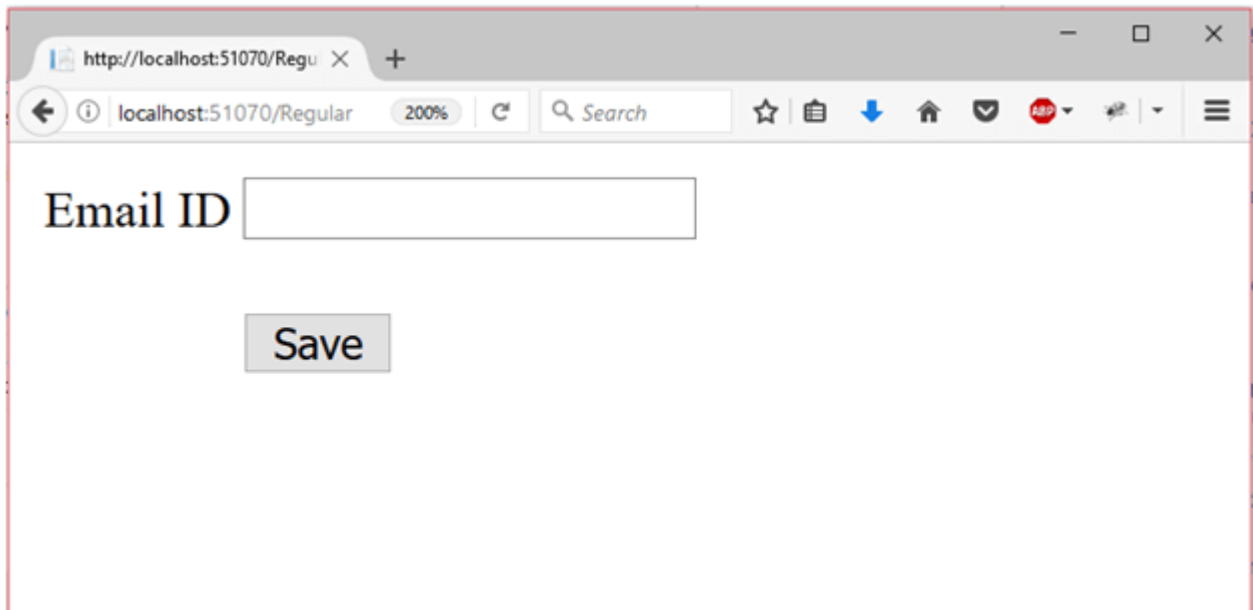
```

11.      <table class="auto-style1">
12.      <tr>
13.      <td class="auto-style2">Email ID</td>
14.      <td>
15.      <asp:TextBox ID="username" runat="server"></asp:TextBox>
16.      <asp:RegularExpressionValidator ID="RegularExpressionValidator1"
        runat="server"ControlToValidate="username"
17.      ErrorMessage="Please enter valid email" ForeColor="Red"ValidationEx
        pression="\w+([-+.']\w+)*@\w+([-.']\w+)*\.\w+([-.']\w+)*">
18.      </asp:RegularExpressionValidator>
19.      </td>
20.      </tr>
21.      <tr>
22.      <td class="auto-style2"></td>
23.      <td>
24.      <br/>
25.      <asp:Button ID="Button1" runat="server" Text="Save"/>
26.      </td>
27.      </tr>
28.      </table>
29.      </div>
30.      </form>
31.      </body>
32.      </html>

```

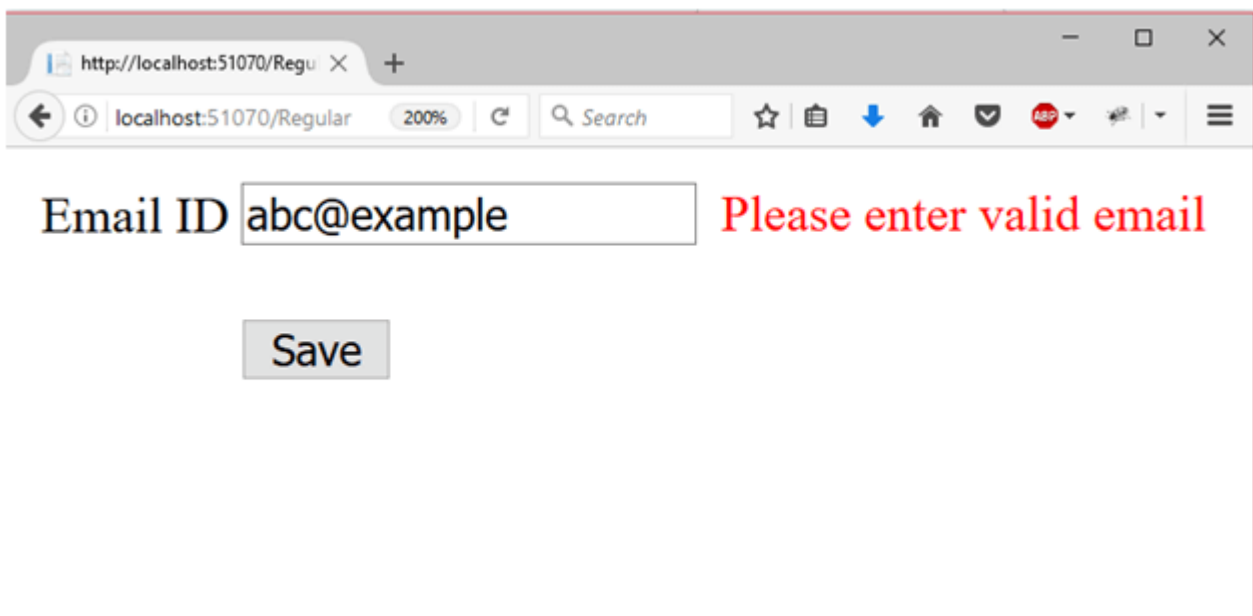
Output:

It produces the following output when view in the browser.



A screenshot of a web browser window. The address bar shows 'http://localhost:51070/Regu'. The page content includes a label 'Email ID' followed by an empty text input field. Below the input field is a grey button labeled 'Save'.

It will validate email format as we specified in regular expression. If validation fails, it throws an error message.



A screenshot of the same web browser window. The input field now contains the text 'abc@example'. To the right of the input field, a red error message 'Please enter valid email' is displayed. The 'Save' button remains below the input field.

1.7.4.ASP.NET RequiredFieldValidator Control

This validator is used to make an input control required. It will throw an error if user leaves input control empty. It is used to mandate form control required and restrict the user to provide data. It removes extra spaces from the beginning and end of the input value before validation is performed.

The `ControlToValidate` property should be set with the ID of control to validate.

RequiredFieldValidator Properties

Property	Description
AccessKey	It is used to set keyboard shortcut for the control.
BackColor	It is used to set background color of the control.
BorderColor	It is used to set border color of the control.
Font	It is used to set font for the control text.
ForeColor	It is used to set color of the control text.
Text	It is used to set text to be shown for the control.
ToolTip	It displays the text when mouse is over the control.
Visible	To set visibility of control on the form.
Height	It is used to set height of the control.
Width	It is used to set width of the control.
ErrorMessage	It is used to set error message that display when validation fails.
ControlToValidate	It takes ID of control to validate.

Example

Here, in the following example, we are explaining **RequiredFieldValidator** control and creating to mandatory TextBox controls.

// RequiredFieldValidator.aspx


```

1.      <%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Req
        uiredFieldValidator.aspx.cs"
2.      Inherits="asp.netexample.RequiredFieldValidator" %>
3.      <!DOCTYPE html>
4.      <html xmlns="http://www.w3.org/1999/xhtml">
5.      <head runat="server">
6.      <title></title>
7.      <style type="text/css">
8.      .auto-style1 {
9.      width: 100%;
10.     }
11.     .auto-style2 {
12.     width: 165px;
13.     }
14.     </style>
15.     </head>
16.     <body>
17.     <form id="form1" runat="server">
18.     <div>
19.     </div>
20.     <table class="auto-style1">
21.     <tr>
22.     <td class="auto-style2">User Name</td>
23.     <td>
24.     <asp:TextBox ID="username" runat="server"></asp:TextBox>
25.     <asp:RequiredFieldValidatorID=asp:RequiredFieldValidatorID="user" r
        unat="server" ControlToValidate="username"
26.     ErrorMessage="Please enter a user name" ForeColor="Red"></asp:Req
        uiredFieldValidator>
27.     </td>
28.     </tr>
29.     <tr>
30.     <td class="auto-style2">Password</td>

```

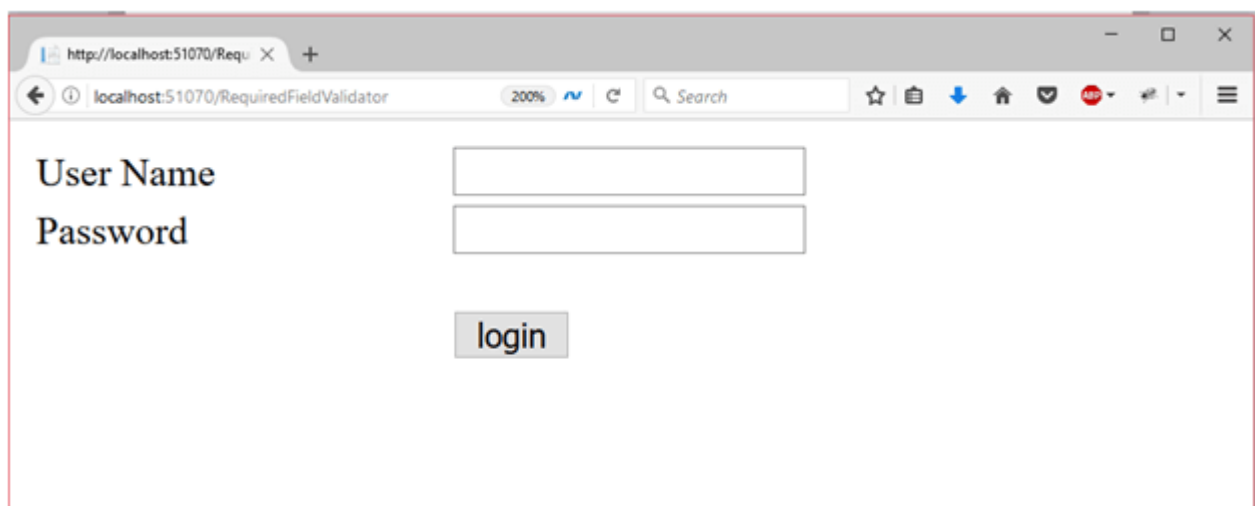
```

31.      <td>
32.      <asp:TextBox ID="password" runat="server"></asp:TextBox>
33.      <asp:RequiredFieldValidator ID="pass" runat="server" ControlToVali
        date="password" ErrorMessage="Please enter a password"
34.      ForeColor="Red"></asp:RequiredFieldValidator>
35.      </td>
36.      </tr>
37.      <tr>
38.      <td class="auto-style2"> </td>
39.      <td>
40.      <br/>
41.      <asp:Button ID="Button1" runat="server" Text="login"/>
42.      </td>
43.      </tr>
44.      </table>
45.      </form>
46.      </body>
47.      </html>

```

Output:

It produces the following output when view in the browser.



It throws error messages when user login with empty controls.

The screenshot shows a web browser window with the address bar displaying 'http://localhost:51070/RequiredFieldValidator'. The page content includes a login form with two text input fields. The first field is labeled 'User Name' and the second is labeled 'Password'. To the right of the 'User Name' field, there is a red error message: 'Please enter a user name'. Similarly, to the right of the 'Password' field, there is a red error message: 'Please enter a password'. Below these fields is a blue button labeled 'login'.

1.7.5.ASP.NET ValidationSummary Control

This validator is used to display list of all validation errors in the web form. It allows us to summarize the error messages at a single location. We can set **DisplayMode** property to display error messages as a list, bullet list or single paragraph.

ValidationSummary Properties

Property	Description
AccessKey	It is used to set keyboard shortcut for the control.
BackColor	It is used to set background color of the control.
BorderColor	It is used to set border color of the control.
Font	It is used to set font for the control text.
ForeColor	It is used to set color of the control text.
Text	It is used to set text to be shown for the control.
ToolTip	It displays the text when mouse is over the

	control.
Visible	To set visibility of control on the form.
Height	It is used to set height of the control.
Width	It is used to set width of the control.
ShowMessageBox	It displays a message box on error in up-level browsers.
ShowSummary	It is used to show summary text on the form page.
ShowValidationErrors	It is used to set whether the validation summary should be shown or not.

Example

The following example explains how to use **ValidationSummery** control in the application.

// ValidationSummeryDemo.aspx

1. <%@ Page Language="C#" AutoEventWireup="true" CodeBehind="ValidationSummeryDemo.aspx.cs"
2. Inherits="asp.netexample.ValidationSummeryDemo" %>
3. <!DOCTYPE html>
4. <**html** xmlns="http://www.w3.org/1999/xhtml">
5. <**head** runat="server">
6. <**title**></**title**>
7. </**head**>
8. <**body**>
9. <**form** id="form1" runat="server">
10. <**div**>
11. </**div**>
12. <**table** class="auto-style1">
13. <**tr**>
14. <**td** class="auto-style2">User Name</**td**>

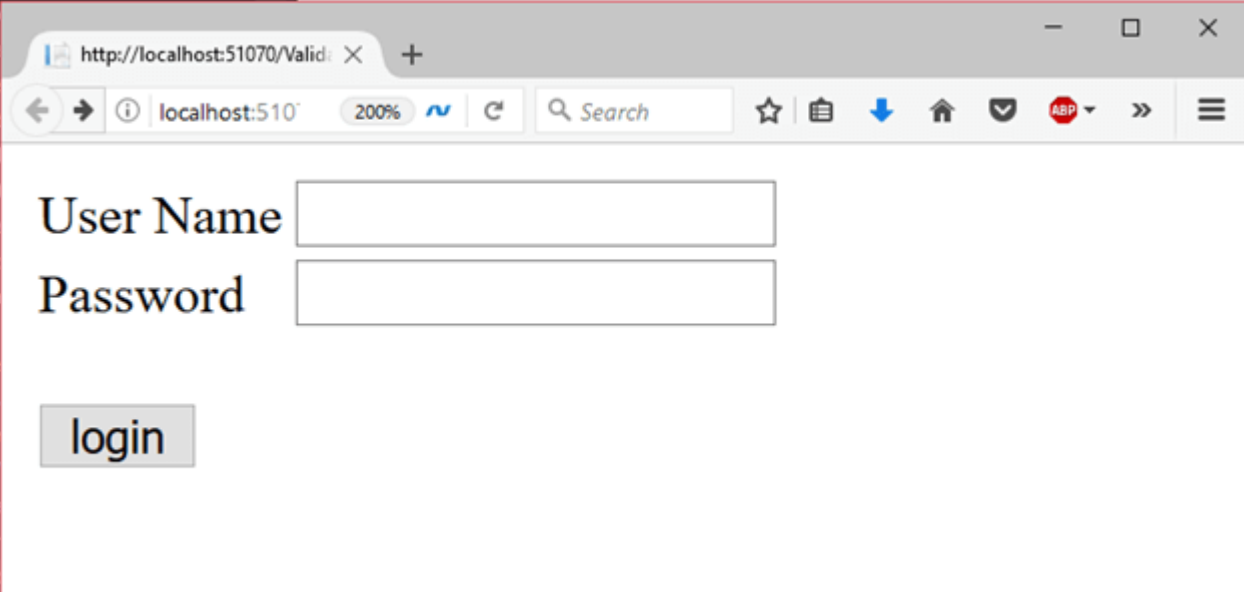
```

15.  <td>
16.  <asp:TextBox ID="username" runat="server"></asp:TextBox>
17.  <asp:RequiredFieldValidator ID="user" runat="server" ControlToValidat
    e="username"
18.  ErrorMessage="Please enter a user name" ForeColor="Red">*</asp:Requi
    redFieldValidator>
19.  </td>
20.  </tr>
21.  <tr>
22.  <td class="auto-style2">Password</td>
23.  <td>
24.  <asp:TextBox ID="password" runat="server"></asp:TextBox>
25.  <asp:RequiredFieldValidator ID="pass" runat="server" ControlToValidat
    e="password"
26.  ErrorMessage="Please enter a password" ForeColor="Red">*</asp:Requir
    edFieldValidator>
27.  </td>
28.  </tr>
29.  <tr>
30.  <td class="auto-style2">
31.  <br/>
32.  <asp:Button ID="Button1" runat="server"Text="login"/>
33.  </td>
34.  <td>
35.  <asp:ValidationSummary ID="ValidationSummary1" runat="server" Fore
    Color="Red"/>
36.  <br/>
37.  </td>
38.  </tr>
39.  </table>
40.  </form>
41.  </body>
42.  </html>

```

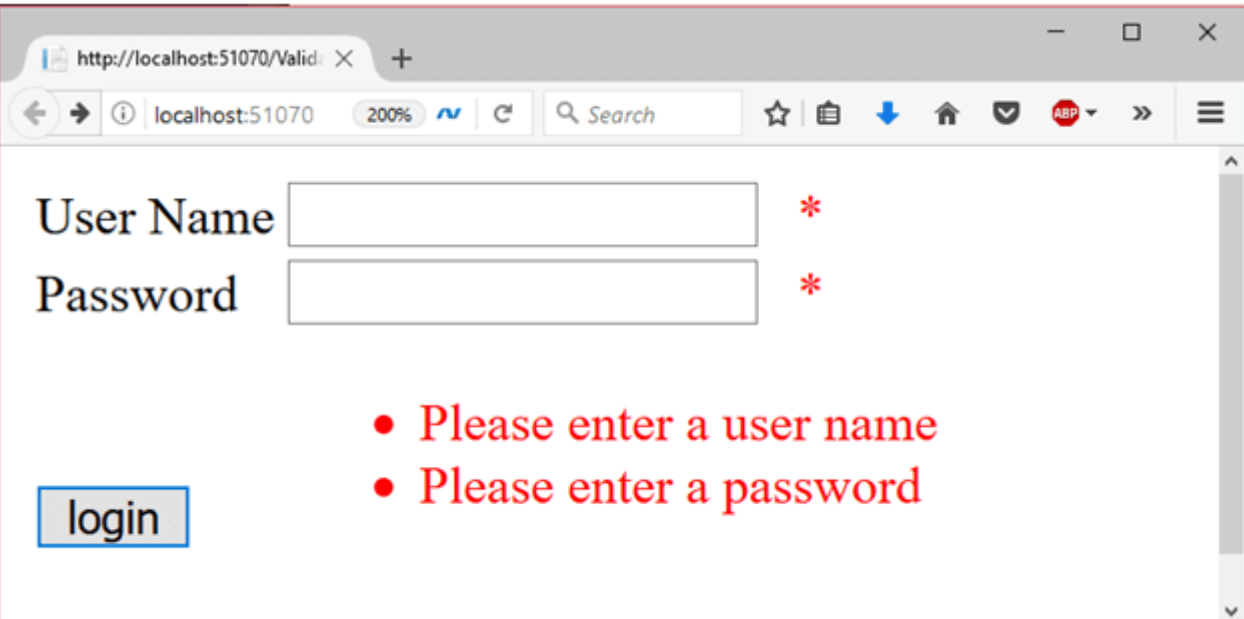
Output:

It produces the following output when view in the browser.



A screenshot of a web browser window. The address bar shows 'http://localhost:51070/Valid:'. The browser's address bar and toolbar are visible. The main content area displays a login form with two input fields: 'User Name' and 'Password'. Below these fields is a 'login' button. The browser's address bar shows 'http://localhost:51070/Valid:'. The browser's address bar and toolbar are visible. The main content area displays a login form with two input fields: 'User Name' and 'Password'. Below these fields is a 'login' button.

It throws error summary when user login without credentials.



A screenshot of a web browser window showing the same login form as before, but with validation errors. The 'User Name' and 'Password' input fields are now empty and have a red asterisk (*) next to them. Below the input fields, there is a red error message: 'Please enter a user name' and 'Please enter a password'. The 'login' button is still present. The browser's address bar shows 'http://localhost:51070/Valid:'. The browser's address bar and toolbar are visible. The main content area displays a login form with two input fields: 'User Name' and 'Password'. Below these fields is a 'login' button. The browser's address bar shows 'http://localhost:51070/Valid:'. The browser's address bar and toolbar are visible. The main content area displays a login form with two input fields: 'User Name' and 'Password'. Below these fields is a 'login' button.

1.7.6. 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.

1.8. ASP.NET CUSTOM CONTROLS

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

- User controls
- Custom controls

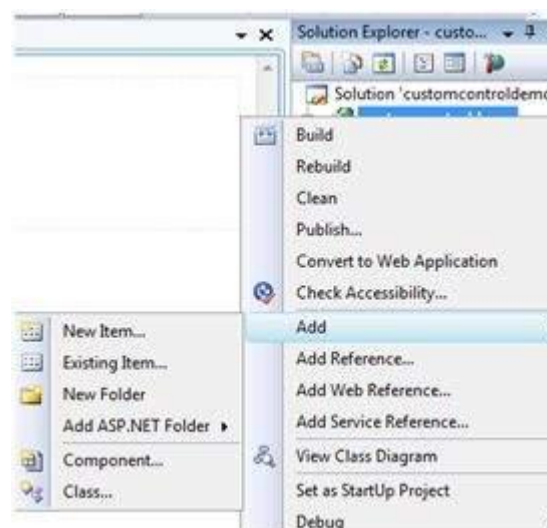
1.8.1 USER CONTROLS

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 understand the concept, let us create a simple user control, which will work as footer for the web pages. To create and use the user control, take the following steps:

- Create a new web application.
- Right click on the project folder on the Solution Explorer and choose



Add New Item.

- Select Web User Control from the Add New Item dialog box and name it footer.ascx. Initially, the footer.ascx contains only a Control directive.

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeBehind="footer.ascx.cs" Inherits="customcontroldemo.footer" %>
```

Add the following code to the file:

```
<table>
<tr>
<td align="center"> Copyright ©2010 TutorialPoints Ltd.</td>
</tr>
<tr>
<td align="center"> Location: Hyderabad, A.P </td>
</tr>
</table>
```

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. The following code shows the content file:

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

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

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

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

<head runat="server">
<title>
Untitled Page
```



```

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

      <asp:Label ID="Label1" runat="server" Text="Welcome to ASP.Net
Tutorials "></asp:Label>
      <br /> <br />
      <asp:Button ID="Button1" runat="server" onclick="Button1_Click"
Text="Copyright Info" />

    </div>
    <Tfooter:footer ID="footer1" runat="server" />
  </form>
</body>
</html>

```

When executed, the page shows the footer and this control could be used in all the pages of your website.

Welcome to ASP.Net Tutorials

Copyright Info

Copyright ©2010 TutorialPoints Ltd.

Location: Hyderabad, A.P

Observe the following:

(1) The Register directive specifies a tag name as well as tag prefix for the control.

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

(2) The following tag name and prefix should be used while adding the user control on the page:

```
<Tfooter:footer ID="footer1" runat="server" />
```

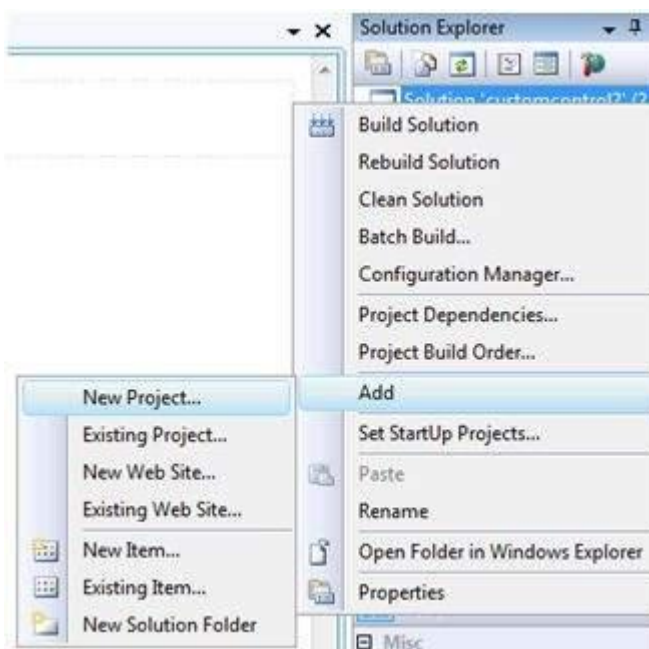
1.8.2.CUSTOM CONTROLS

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 combining two or more existing controls.
- By deriving from the base control class.

To understand the concept, let us create a custom control, which will simply render a text message on the browser. To create this control, take the following steps:

Create a new website. Right click the solution (not the project) at the top of the tree in the Solution Explorer.

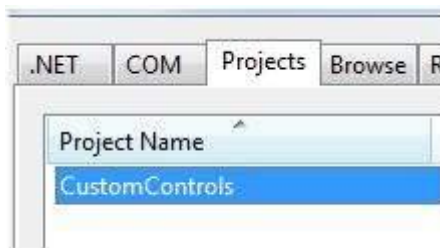


In the New Project dialog box, select ASP.NET Server Control from the project templates.



The above step adds a new project and creates a complete custom control to the solution, called ServerControl1. In this example, let us name the project CustomControls. To use this control, this must be added as a reference to the web site before registering it on a page. To add a reference to the existing project, right click on the project (not the solution), and click Add Reference.

Select the CustomControls project from the Projects tab of the Add Reference dialog box. The Solution Explorer should show the reference.



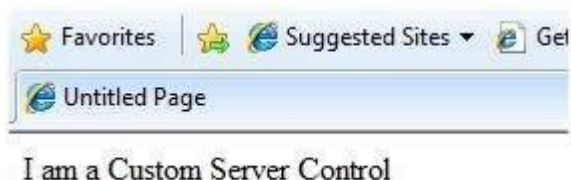
To use the control on a page, add the Register directive just below the @Page directive:

```
<%@ Register Assembly="CustomControls" Namespace="CustomControls"
TagPrefix="ccs" %>
```

Further, you can use the control, similar to any other controls.

```
<form id="form1" runat="server">
  <div>
    <ccs:ServerControl1 runat="server" Text = "I am a Custom Server
Control" />
  </div>
</form>
```

When executed, the Text property of the control is rendered on the browser as shown:



Working with Custom Controls

In the previous example, the value for the Text property of the custom control was set. ASP.NET added this property by default, when the control was created. The following code behind file of the control reveals this.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;

using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace CustomControls
{
    [DefaultProperty("Text")]
    [ToolboxData("<{0}:ServerControl1 runat=server></{0}:ServerControl1 >")]

    public class ServerControl1 : WebControl
    {
        [Bindable(true)]
        [Category("Appearance")]
        [DefaultValue("")]
        [Localizable(true)]
        public string Text
        {
            get
            {
                String s = (String)ViewState["Text"];
                return ((s == null) ? "[" + this.ID + "]" : s);
            }

            set
```

```

    {
        ViewState["Text"] = value;
    }
}

protected override void RenderContents(HtmlTextWriter output)
{
    output.Write(Text);
}
}

```

The above code is automatically generated for a custom control. Events and methods could be added to the custom control class.

Example

Let us expand the previous custom control named `SeverControl1`. Let us give it a method named `checkpalindrome`, which gives it a power to check for palindromes.

Palindromes are words/literals that spell the same when reversed. For example, Malayalam, madam, saras, etc.

Extend the code for the custom control, which should look as:

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;

using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

```

```

namespace CustomControls
{
    [DefaultProperty("Text")]
    [ToolboxData("<{0};ServerControl1 runat=server></{0};ServerControl1 >")]

    public class ServerControl1 : WebControl
    {
        [Bindable(true)]
        [Category("Appearance")]
        [DefaultValue("")]
        [Localizable(true)]

        public string Text
        {
            get
            {
                String s = (String)ViewState["Text"];
                return ((s == null) ? "[" + this.ID + "]" : s);
            }

            set
            {
                ViewState["Text"] = value;
            }
        }

        protected override void RenderContents(HtmlTextWriter output)
        {
            if (this.checkpanlindrome())
            {
                output.Write("This is a palindrome: <br />");
                output.Write("<FONT size=5 color=Blue>");
                output.Write("<B>");
            }
        }
    }
}

```

```

        output.Write(Text);
        output.Write("</B>");
        output.Write("</FONT>");
    }
    else
    {
        output.Write("This is not a palindrome: <br />");
        output.Write("<FONT size=5 color=red>");
        output.Write("<B>");
        output.Write(Text);
        output.Write("</B>");
        output.Write("</FONT>");
    }
}

```

```

protected bool checkpanlindrome()
{
    if (this.Text != null)
    {
        String str = this.Text;
        String strtoupper = Text.ToUpper();
        char[] rev = strtoupper.ToCharArray();
        Array.Reverse(rev);
        String strrev = new String(rev);

        if (strtoupper == strrev)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
}

```

```

    }
    else
    {
        return false;
    }
}
}
}

```

When you change the code for the control, you must build the solution by clicking Build --> Build Solution, so that the changes are reflected in your project. Add a text box and a button control to the page, so that the user can provide a text, it is checked for palindrome, when the button is clicked.

```

<form id="form1" runat="server">
    <div>
        Enter a word:
        <br />
        <asp:TextBox ID="TextBox1" runat="server" style="width:198px">
</asp:TextBox>

        <br /> <br />

        <asp:Button ID="Button1" runat="server" onclick="Button1_Click"
Text="Check Palindrome" style="width:132px" />

        <br /> <br />

        <ccs:ServerControl1 ID="ServerControl1" runat="server" Text = "" />
    </div>
</form>

```

The Click event handler for the button simply copies the text from the text box to the text property of the custom control.

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



```
{
    this.ServerControl11.Text = this.TextBox1.Text;
}
```

When executed, the control successfully checks palindromes.

Enter a word:

madam

Check Palindrome

This is a palindrome:

madam

Observe the following:

(1) 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.



(2) The RenderContents method of the custom control class is overridden here, as you can add your own methods and events.

(3) The RenderContents method takes a parameter of HtmlTextWriter type, which is responsible for rendering on the browser.

1.9 KEY TERMS

- **CLR** – a run-time environment called the **common language runtime**, which runs the code and provides services that make the development process easier.
- **IIS** – **Internet Information Service**, The work of IIS is to provide the web application's generated HTML code result to the client browser which initiated the request.

- **Managed code** - Code that has to develop with a language compiler that targets the runtime is called **managed code**.
- **Metadata**- To enable the runtime to provide services to managed code, language compilers must emit **metadata** that describes the types, members, and references in the code.
- **CLI**- The "Common Language Infrastructure" or CLI is a platform on which the .Net programs are executed.
- **Garbage Collection** - Garbage collection is the process of removing unwanted resources when they are no longer required.
- **Common Language Interpreter** – This is the final layer in .Net which would be used to run a .net program developed in any programming language. So the subsequent compiler will send the program to the CLI layer to run the .Net application.
- **Interoperability** – the ability of computer systems or software to exchange and make use of information. "interoperability between devices made by different manufacturers"
- Portability – **Portability** is a characteristic attributed to a computer **program** if it can be used in an operating system other than the one in which it was created without requiring major rework. Porting is the task of doing any work necessary to make the computer **program** run in the new environment

1.9.1 TWO MARKS

1. Define CLR

Ans: CLR is a run-time environment called the common language runtime, which runs the code and provides services that make the development process easier

2. Illustrate the version history of ASP.Net.

Ans :

YEAR	VERSION
2002	1.0

YEAR	VERSION
2003	1.1
2005	2.0
2006	3.0
2007	3.5
2008	3.5 SP 1
2010	4.0
2012	4.5
2013	4.5.1
2014	4.5.2
2015	4.6
2015	4.6.1
2016	4.6.2
2017	4.7
2017	4.7.1

3. What is Web Application?

Ans: A Web Application is an application installed only on the web server which is accessed by the users using a web browser like Microsoft Internet Explorer, Google Chrome, Mozilla firefox, Apple Safari etc

4. Specify the use of IIS

Ans : The web applications which are developed using the .Net Framework or its subsets required to execute under the Microsoft Internet Information Services (IIS) on the server side. The work of IIS is to provide the web application's generated HTML code result to the client browser which initiated the request.

5. What is Managed Code?

Ans: Code that has to develop with a language compiler that targets the runtime is called managed code

6. Discuss the benefits of Managed Code

Ans : Managed code benefits from features such as cross-language integration, cross-language exception handling, enhanced security, versioning and deployment support, a simplified model for component interaction, and debugging and profiling services.

7. Define Metadata.

Ans : Metadata describes the types, members, and references in the code, which is used to enable the runtime to provide services to managed code, language compilers must emit.

8. What is cross-language Integration?

Ans : It is the ability provided by the common language runtime (CLR) and the common language specification (CLS), of the .NET Framework, for interaction with code written in a different programming language. Cross language support is a language interoperability feature with advantages, such as the reuse of types defined in other languages; a single environment for debugging and profiling, due to the use of Microsoft intermediate language (MSIL); and consistent exception handling, where exceptions thrown in one language can be caught in another language.

9. Why Garbage collection is needed?

Ans : Garbage collection is the process of removing unwanted resources when they are no longer required. Examples of garbage collection are

- A File handler which is no longer required. If the application has finished all operations on a file, then the file handle may no longer be required.

- The database connection is no longer required. If the application has finished all operations on a database, then the database connection may no longer be required.

10. What is common Language Interpreter?

Ans : Common Language Interpreter is the final layer in .Net which would be used to run a .net program developed in any programming language. So the subsequent compiler will send the program to the CLI layer to run the .Net application.

11. Define Interoperability.

Ans : the ability of computer systems or software to exchange and make use of information. "interoperability between devices made by different manufacturers"

12. Define portability.

Ans : Portability is a characteristic attributed to a computer program if it can be used in an operating system other than the one in which it was created without requiring major rework. Porting is the task of doing any work necessary to make the computer program run in the new environment

13. What is the use of Validation Control?

Ans : It evaluates the value of an input control to check the specified range. It evaluates the value of an input control to determine whether it matches a pattern defined by a regular expression. It is used to make a control required.

1.9.2 FIVE MARKS

1. Give a brief introduction about ASP.Net and its Version history.
2. Discuss in detail about the working of CLR.
3. Write short notes on the benefits of CLR.

4. What are the different types of Applications supported under ASP.NET? Discuss.
5. Discuss about .Net Framework Design Principle in short.
6. Explain the ASP.Net Page Structure.
7. Give a brief introduction about Page Life Cycle.
8. Explain about HTML Web Server Controls.

1.9.1 TEN MARKS

1. Discuss in detail about .NET Framework architecture and its components.
2. Discuss in detail about CLI and its features.
3. How to create and run a web application under .NET Environment? Discuss with an example.
4. What are HTML Server Controls in ASP.NET? Discuss briefly.
5. Elaborate the various used of Validation controls and its types in ASP.NET.
6. Discuss about ASP.Net Custom Controls.

UNIT – II

ASP .Net Form Validation and State Management

2.1 INTRODUCTION

- ASP.NET Web Forms is a part of the ASP.NET web application framework. It is used to create ASP.NET web applications. Apart from this it also create ASP.NET MVC, ASP.NET Web Pages, and ASP.NET Single Page Applications.
- Web Forms are pages that users request using their browser. Any ASPX files are usually referred to as Web Forms or Web Form Pages because they normally process form input.
- The Motivation of Web Applications Development is to gather or provide information. In Web application development collecting valid data is more important. In this chapter we focus on how to collect valid data through web applications and various web server controls used to validate the data.
- This chapter also overview the state management techniques in ASP.NET and discussing about the various types of state management both client side and server side.

2.2 BASIC CONCEPT OF ASP.NET WEB FORMS

- Web forms are made up of different types of HTML elements that are constructed using raw HTML form elements, ASP.NET HTML server controls, or ASP.NET Web Form server controls.
- Data entered into a form can be sent to the server, processed, then sent back to the client in different format.
- The first time a Web form is requested, the entire page is compiled. Later requests are served from the compiled page and do not have to be recompiled.
- Any pure HTML page can be given an .aspx extension and run as an ASP.NET page. Programmers customize Web Forms by adding Web

controls, which include labels, text boxes,, images, buttons and other GUI components.

ASP.Net Web Form also offers

- Separation of HTML and other UI code from application logic.
- A rich suite of server controls for common tasks, including data access.
- Powerful data binding, with great tool support.
- Support for client-side scripting that executes in the browser.
- Support for a variety of other capabilities, including routing, security, performance, internationalization, testing, debugging, error handling and state management.

2.3 VALIDATION

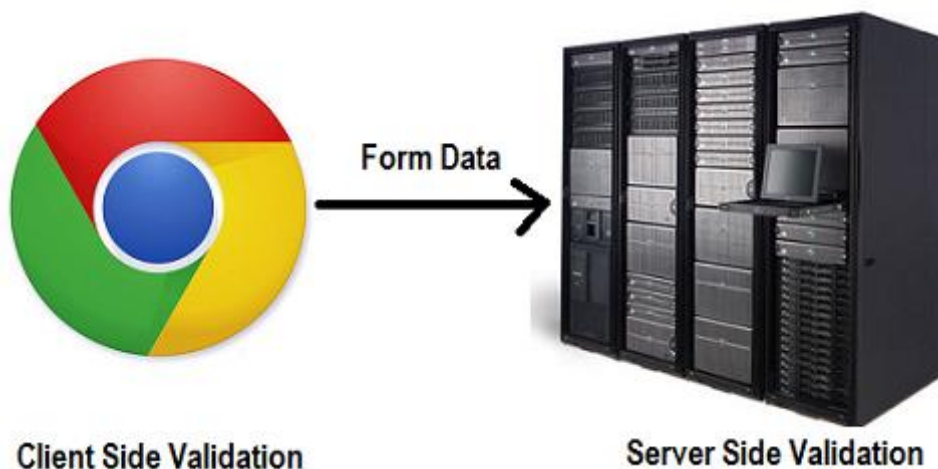
Validation is a set of rules that apply to the data that should be collected. These rules can be many or few and enforced either strictly or in a lax manner: It really depends on user. The data you collect for validation comes from the Web forms provide in the applications.

Validation is basically the act of comparing something to a given set of rules and determining if it satisfies the criteria those rules represent.

Validations can be performed on the

a) Server side Validation

b) Client side Validation



Fig(2.1) ASP .Net Form Validation Types

The user input validation take place on the Server Side during a post back session is called Server Side Validation and the user input validation take place on the Client Side (web browser) is called Client Side Validation. Client Side Validation does not require a postback. They above diagram 2.1 shows about the validation types in ASP. Net.

a) Server Side Validation

In the Server Side Validation, the input submitted by the user is being sent to the server and validated. After the validation process on the Server Side, the feedback is sent back to the client by a new dynamically generated web page.

It is better to validate user input on Server Side because we can protect against the malicious users, who can easily bypass your Client Side scripting language and submit dangerous input to the server.

For Example: Suppose that the end user clicks the Submit button on a form after filling out some information. This form is packaged in a request and sent to the server where the application resides. At this point in the request/response cycle, we can run validation checks on the information submitted. It is called server-side validation.

b) Client Side Validation

In the Client Side Validation we can provide a better user experience by responding quickly at the browser level. When we perform a Client Side Validation, all the user inputs validated in the user's browser itself. Client Side validation does not require a round trip to the server, so the network traffic which will help your server perform better.

For example: if the user enter an invalid email format, we can show an error message immediately before the user move to the next field, so the user can correct every field before they submit the form. This is called client-side validation.

2.3.1. CLIENT-SIDE VERSUS SERVER-SIDE VALIDATION

CLIENT SIDE VALIDATION	SERVER SIDE VALIDATION
Client-side validation means that the validation checks are performed on the client.	Server-side validation means that the validation checks are performed on the server instead of on the client.
Client-side validation is quick and responsive for the end user. Client-side validation also pushes the processing power required of validation to the client meaning that you don't need to spin CPU cycles on the server to process the same information because the client can do the work for you.	Server-side validation can be slow because the page has to be posted to a remote location and checked. End user might not be the happiest surfer in the world if, after waiting 20 seconds for a form to post, he is told his e-mail address isn't in the correct format.
Client-side validation is the more insecure form of validation. When a page is generated in an end user's browser, this end user can look at the code of the page quite easily	The more secure form of validation is server-side validation. It is more secure because these checks cannot be easily bypassed. Instead, the form data values are checked using server code (C# or VB) on the server.
Hackers can simply bypass the validation.	It is secure because hackers can't simply bypass the validation.

The best approach is always to perform client-side validation first and then, after the form passes and is posted to the server, to perform the validation checks again using server-side validation. This approach provides the best of both worlds. It is secure because hackers can't simply bypass the validation.

They may bypass the client-side validation, but they quickly find that their form data is checked once again on the server after it is posted. This validation technique is also highly effective — giving you both the quickness and snappiness of client-side validation.

2.4 VALIDATION CONTROLS

ASP.NET validation controls validate the user input data to ensure that useless, unauthenticated, or contradictory data don't get stored. ASP.NET not only introduces form validations as server controls, but it also makes these controls rather smart. ASP.NET performs browser detection when generating the ASP.NET page and makes decisions based on the information it gets.

ASP.NET provides the following validation controls:

1. RequiredFieldValidator
2. RangeValidator
3. CompareValidator
4. RegularExpressionValidator
5. CustomValidator
6. ValidationSummary

The following table describes the functionality of each of the available validation server controls.

Validation Server Control	Description
RequiredFieldValidator	Ensures that the user does not skip a form entry field
CompareValidator	Allows for comparisons between the user's input and another item using a comparison operator (equals, greater than, less than, and so on)
RangeValidator	Checks the user's input based

	upon a lower- and upperlevel range of numbers or characters
RegularExpressionValidator	Checks that the user's entry matches a pattern defined by a regular expression. This is a good control to use to check e-mail addresses and phone numbers
CustomValidator	Checks the user's entry using custom-coded validation logic
ValidationSummary	Displays all the error messages from the validators in one specific spot on the page

BaseValidator Class

The validation control classes are inherited from the BaseValidator class hence they inherit its properties and methods. Therefore, it would help to take a look at the properties and the methods of this base class, which are common for all the validation controls:

Members	Description
ControlToValidate	Indicates the input control to validate.
Display	Indicates how the error message is shown.
EnableClientScript	Indicates whether client side validation will take.
Enabled	Enables or disables the validator.
ErrorMessage	Indicates error string.
Text	Error text to be shown if validation fails.
IsValid	Indicates whether the value of

	the control is valid.
SetFocusOnError	It indicates whether in case of an invalid control, the focus should switch to the related input control.
ValidationGroup	The logical group of multiple validators, where this control belongs.
Validate()	This method revalidates the control and updates the IsValid property.

2.4.1. Required Field Validator Control

The RequiredFieldValidator control simply checks to see if something was entered into the HTML form element. It is a simple validation control, but it is one of the most frequently used. We use RequiredFieldValidator control for each form element on which you wish to enforce a value-required rule. It is generally tied to a text box to force input into the text box.

The syntax of the control is as given:

```
<asp:RequiredFieldValidator ID="candidate"
  runat="server" ControlToValidate ="ddlcandidate"
  ErrorMessage="Please choose a candidate"
  InitialValue="Please choose a candidate">
</asp:RequiredFieldValidator>
```

The following program shows a simple use of the RequiredFieldValidator Control.

```
<form id="form1" runat="server">
  Your name:<br />
```

```

<asp:TextBox runat="server" id="txtName" />
<asp:RequiredFieldValidator          runat="server"          id="reqName"
controltovalidate="txtName" errormessage="Please enter your name!" />
<br /><br />
<asp:Button runat="server" id="btnSubmitForm" text="Ok" />
</form>

```

Build and run this page. It is presented with a simple text box and button on the page. Don't enter any value inside the text box, and click the Submit button. The result is shown in Figure 2.2.

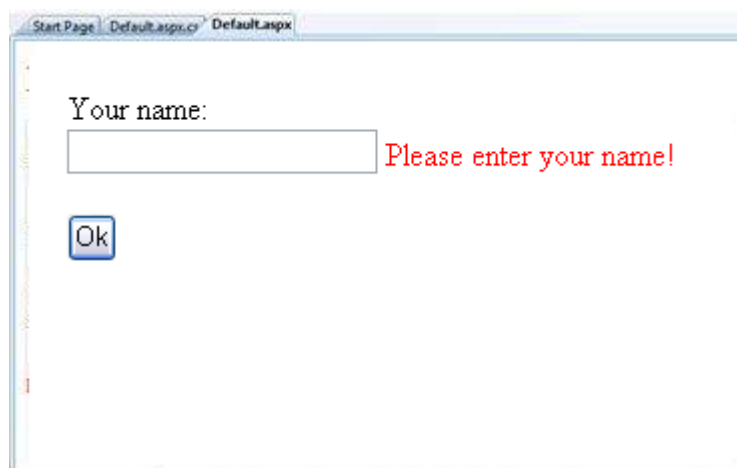


Fig (2.2) Required field Validator

a) Using the Initial Value Property

Another important property when working with the RequiredFieldValidator control is the InitialValue property. Sometimes we have form elements that are populated with some default properties (for example, from a data store), and these form elements might present the end user with values that require changes before the form can be submitted to the server.

When using the InitialValue property, we specify to the RequiredFieldValidator control the initial text of the element. The end user is then required to change that text value before submitting the form. The following Listing shows an example of using this property.

```
<asp:TextBox ID="TextBox1" Runat="server">My Initial
Value</asp:TextBox> &nbsp;
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
Runat="server" ErrorMessage="Please change the value of the textbox!"
ControlToValidate="TextBox1" InitialValue="My Initial Value">
```

In this case, the InitialValue property contains a value of My Initial Value. When the page is built and run, the text box contains this value as well. The RequiredFieldValidator control requires a change in this value for the page to be considered valid.

b) Disallowing Blank Entries and Requiring Changes at the Same Time

A blank text box does not fire a validation error because the RequiredFieldValidator control is reconstructed to force the end user only to *change* the default value of the text box..

To both require a change to the initial value of the text box and to disallow a blank entry (thereby making the element a required element), we must put an additional RequiredFieldValidator control on the page.

This second RequiredFieldValidator control is associated with the same text box as the first RequiredFieldValidator control. This is illustrated in the example shown below.

```
<asp:TextBox ID="TextBox1" Runat="server">My Initial
Value</asp:TextBox>&nbsp;

<asp:RequiredFieldValidator ID="RequiredFieldValidator1" Runat="server"
ErrorMessage="Please change value" ControlToValidate="TextBox1"
InitialValue="My Initial Value"></asp:RequiredFieldValidator>
```

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator2" Runat="server"
ErrorMessage="Do not leave empty" ControlToValidate="TextBox1">
</asp:RequiredFieldValidator>
```

In this example, you can see that the text box does indeed have two RequiredFieldValidator controls associated with it. The first, RequiredFieldValidator1, requires a change to the default value of the text box through the use of the InitialValue property. The second RequiredFieldValidator control, RequiredFieldValidator2, simply makes the TextBox1 control a form element that requires a value.

c) Validating Drop-Down Lists with the RequiredFieldValidator Control

We can use the RequiredFieldValidator control with an <asp:DropDownList> server control. Suppose we have a drop-down list that requires the end user to select her profession from a list of items. The first line of the drop-down list includes instructions to the end user about what to select, and you want to make this a required form element as well. The code to do this is shown below.

```
<asp:DropDownList id="DropDownList1" runat="server">
  <asp:ListItem Selected="True">Select a profession</asp:ListItem>
  <asp:ListItem>Programmer</asp:ListItem>
  <asp:ListItem>Lawyer</asp:ListItem>
  <asp:ListItem>Doctor</asp:ListItem>
  <asp:ListItem>Artist</asp:ListItem>
</asp:DropDownList> &nbsp;
<asp:RequiredFieldValidator id="RequiredFieldValidator1" runat="server"
ErrorMessage="Please make a selection"
ControlToValidate="DropDownList1"
InitialValue="Select a profession">
</asp:RequiredFieldValidator>
```


The RequiredFieldValidator control in this example associates itself with the DropDownList control through the use of the ControlToValidate property. The drop-down list to which the validation control is bound has an initial value — Select a profession.

4.3.2 Compare Validator Control

The CompareValidator control compares a value in one control with a fixed value or a value in another control. The CompareValidator control allows to make comparisons between two form elements as well as to compare values contained within form elements to constants that you specify. For instance, a form element's value must be an integer and greater than a specified number. we can also state that values must be strings, dates, or other data types that are at your disposal.

It has the following specific properties:

Properties	Description
Type	It specifies the data type.
ControlToCompare	It specifies the value of the input control to compare with.
ValueToCompare	It specifies the constant value to compare with.
Operator	It specifies the comparison operator, the available values are: Equal, NotEqual, GreaterThan, GreaterThanEqual, LessThan, LessThanEqual, and DataTypeCheck.

The basic syntax of the control is as follows:

```
<asp:CompareValidator ID="CompareValidator1" runat="server"
    ErrorMessage="CompareValidator">
</asp:CompareValidator>
```

a) Validating against Other Controls

One of the more common ways of using the CompareValidator control is to make a comparison between two form elements.

For example, suppose that we have an application which requires users to have passwords in order to access the site. Create one text box asking for the user's password and a second text box which asks the user to confirm the password. Because the text box is in password mode, the end user cannot see what she is typing — just the number of characters that she has typed.

To reduce the chances of the end user mistyping her password and inputting this incorrect password into the system, ask her to confirm the password. After the form is input into the system, simply have to make a comparison between the two text boxes to see if they match. If they match, it is likely that the end user typed the password correctly, and input the password choice into the system. If the two text boxes do not match, the form to be invalid. The following example, demonstrates this situation.

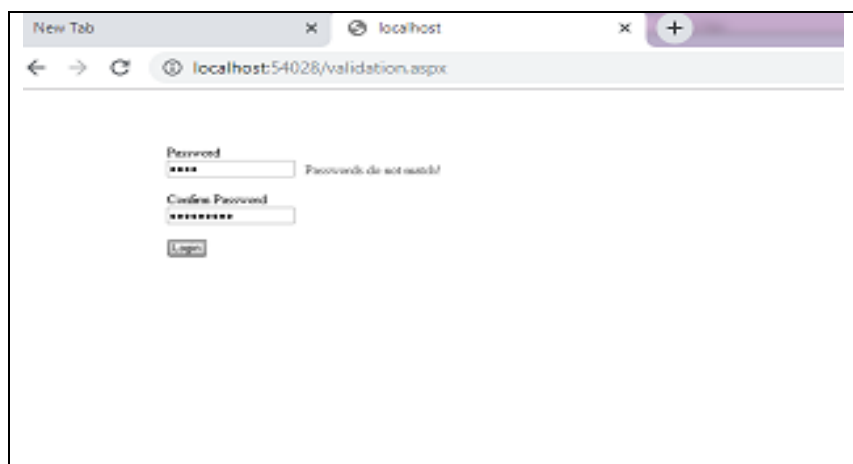
```
<%@ Page Language="VB" %>
<script runat="server">
    Protected Sub Button1_Click(sender As Object, e As EventArgs)
        Label1.Text="Passwords match" End Sub
</script>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>CompareFieldValidator</title>
</head>
<body>
    <form runat="server">
        <p>
            Password<br>
            <asp:TextBox ID="TextBox1" Runat="server"
                TextMode="Password"></asp:TextBox> &nbsp;
```

```

<asp:CompareValidator                                ID="CompareValidator1"
    Runat="server"  ErrorMessage="Passwords  do  not  match!"
    ControlToValidate="TextBox2"
    ControlToCompare="TextBox1"></asp:CompareValidator>
</p>
<p>

```

In this example, we are making a comparison between the value of TextBox2 and that of TextBox1. Therefore, use the ControlToCompare property. This specifies what value is compared to TextBox2. In this case, the value is TextBox1. If the two text boxes do not match after the page is posted by the end user, the value of the ErrorMessage property from the CompareValidator control is displayed in the browser. An example of this is shown in Figure 2.3.



Fig(2.3) Compare Validator Control

b) Validating against Constants

We also use the CompareValidator control to make comparisons against constants of specific data types. For example, suppose we have a text box on registration form that asks for the age of the user. In most cases we want to get back an actual number and not something such as aa or bb as a value. The following programs shows how to ensure that we get back an actual number.

Age:

```
<asp:TextBox ID="TextBox1" Runat="server" MaxLength="3">
</asp:TextBox> &nbsp;
<asp:CompareValidator ID="CompareValidator1" Runat="server"
  ErrorMessage="You must enter a number"
  ControlToValidate="TextBox1" Type="Integer"
  Operator="DataTypeCheck"></asp:CompareValidator>
```

In this example, the end user is required to enter in a number into the text box. If they attempts to bypass the validation by entering a fake value that contains anything other than a number, the page is identified as invalid, and the CompareValidator control displays the value of the ErrorMessage property.

To specify the data types that you want to use in these comparisons, use the Type property. The Type property can take the following values:

- Currency
- Date
- Double
- Integer
- String

Not only can you make sure that what is entered is of a specific data type, but you can also make sure that what is entered is valid when compared to specific constants. For instance, you can make sure what is entered in a form element is greater than, less than, equal to, greater than or equal to, or less than or equal to a specified value. An example of this is illustrated in the following program.

Age:

```
<asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
    &nbsp;
<asp:CompareValidator ID="CompareValidator1"
    Runat="server" Operator="GreaterThan"
    ValueToCompare="18"
    ControlToValidate="TextBox1"
    ErrorMessage="You must be older than 18 to join" Type="Integer">
</asp:CompareValidator>
```

In this case, the CompareValidator control not only associates itself with the TextBox1 control and requires that the value must be an integer, but it also uses the Operator and the ValueToCompare properties to ensure that the number is greater than 18. Therefore, if the end user enters a value of 18 or less, the validation fails, and the page is considered invalid.

The Operator property can take one of the following values:

- Equal
- NotEqual
- GreaterThan
- GreaterThanEqual
- LessThan
- LessThanEqual
- DataTypeCheck

The Value to compare property is where you place the constant value used in the comparison. In the preceding example, it is the number 18.

2.4.3 RangeValidator Control

The RangeValidator control verifies that the input value falls within a predetermined range. The RangeValidator control is quite similar to that of the CompareValidator control, but it makes sure that the end user value or selection provided is between a specified range as opposed to being just greater than or less than a specified constant.

It has three specific properties

Properties	Description
Type	It defines the type of the data. The available values are: Currency, Date, Double, Integer, and String.
MinimumValue	It specifies the minimum value of the range.
MaximumValue	It specifies the maximum value of the range.

The syntax of the control is as given:

```
<asp:RangeValidator ID="rvclass" runat="server"
ControlToValidate="txtclass"
ErrorMessage="Enter your class (6 - 12)" MaximumValue="12"
MinimumValue="6" Type="Integer">
</asp:RangeValidator>
```

For an example , go back to the text-box element that asks for the date of the end user and performs a validation on the value provided. This is illustrated in the following program.

```
Date:<br />
<asp:TextBox runat="server" id="txtDate" />
<asp:RangeValidator runat="server" id="rngDate" controltovalidate="txtDate"
type="Date" minimumvalue="01-01-2006" maximumvalue="31-12-2006"
errormessage="Please enter a valid date within 2006!" />
<br /><br />
```

In this example, this page consists of a text box asking for the date of the end user. The RangeValidator control makes an analysis of the value provided and makes sure the value is somewhere in the range of 01-01-2006 to 31-12-2006.

This is done through the use of the `MaximumValue` and `MinimumValue` properties. The `RangeValidator` control also makes sure what is entered is an integer data type. It uses the `Type` property, which is set to `Integer`. The collection of screenshots in Figure 2.3 shows this example in action.

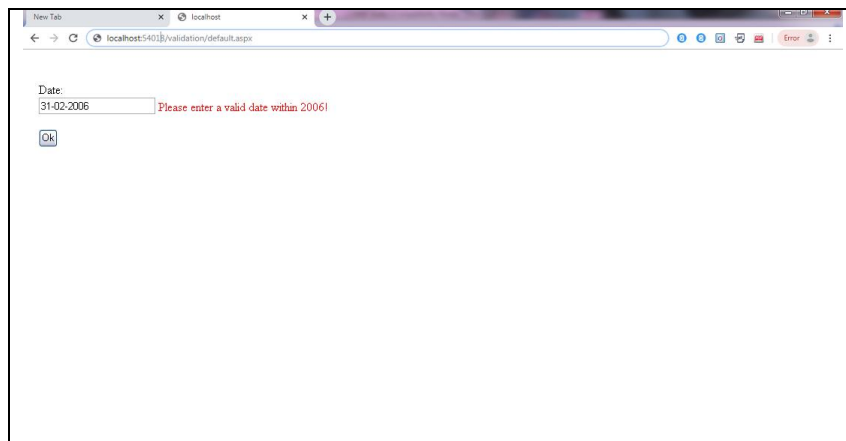


Fig (2.3) Range Validator Control

4.5 . CALENDER CONTROL

The Calendar server control is a rich control that enables to place a full-featured calendar directly on Web pages. It allows for a high degree of customization to ensure that it looks and behaves in a unique manner.

It provides the following capabilities:

- Displaying one month at a time
- Selecting a day, a week or a month
- Selecting a range of days
- Moving from month to month
- Controlling the display of the days programmatically

The Calendar control, in its simplest form, is coded in the following manner:

```
<asp:Calender ID = "Calendar1" runat = "server">
</asp:Calender>
```

This code produces a calendar on your Web page without any styles added, as shown in Figure 2.4

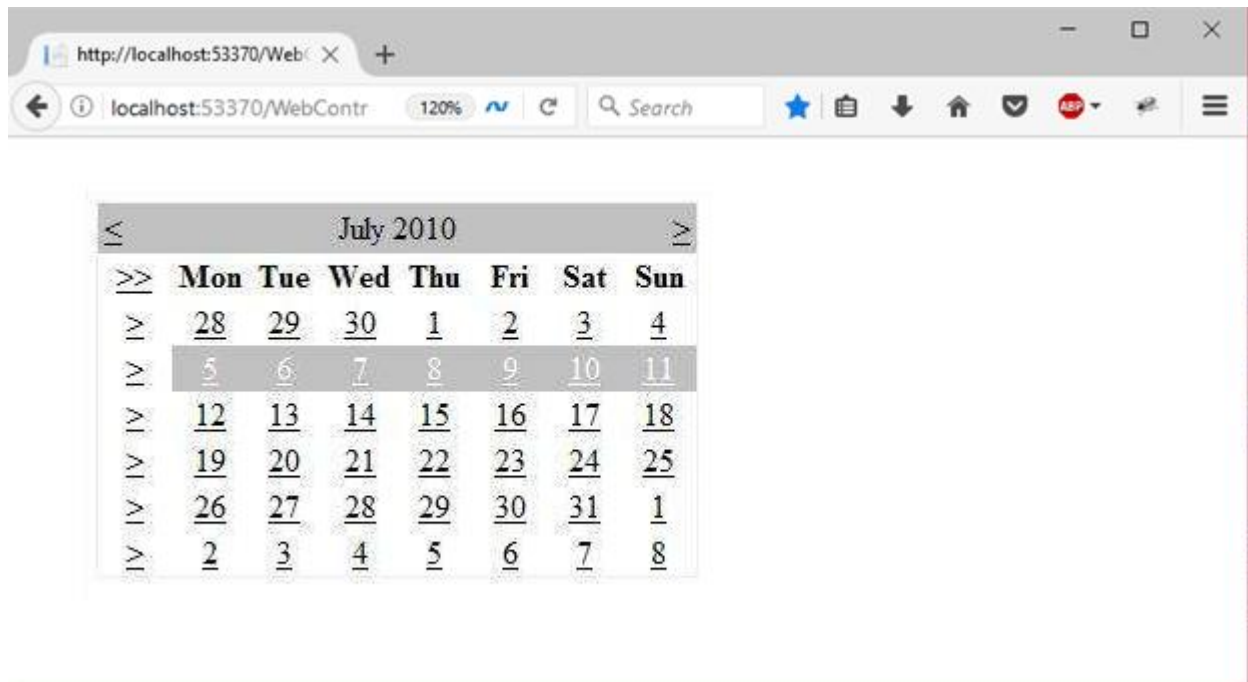


Fig 2.4 Calender Control

a)Properties and Events of the Calendar Control

The calendar control has many properties and events, using which you can customize the actions and display of the control.

The following table provides some important properties of the Calendar control:

Properties	Description
Caption	Gets or sets the caption for the calendar control.
CaptionAlign	Gets or sets the alignment for the caption.
CellPadding	Gets or sets the number of spaces between the data and the cell border.
CellSpacing	Gets or sets the space between cells.

DayHeaderStyle	Gets the style properties for the section that displays the day of the week.
DayNameFormat	Gets or sets format of days of the week.
DayStyle	Gets the style properties for the days in the displayed month.
FirstDayOfWeek	Gets or sets the day of week to display in the first column.
NextMonthText	Gets or sets the text for next month navigation control. The default value is >.
NextPrevFormat	Gets or sets the format of the next and previous month navigation control.
OtherMonthDayStyle	Gets the style properties for the days on the Calendar control that are not in the displayed month.
PrevMonthText	Gets or sets the text for previous month navigation control. The default value is <.
SelectedDate	Gets or sets the selected date.
SelectedDates	Gets a collection of DateTime objects representing the selected dates.
SelectedDayStyle	Gets the style properties for the selected dates.
SelectionMode	Gets or sets the selection mode that specifies whether the user can select a single day, a week or an entire month.
SelectMonthText	Gets or sets the text for the month selection element in the selector column.
SelectorStyle	Gets the style properties for the week and month selector column.

SelectWeekText	Gets or sets the text displayed for the week selection element in the selector column.
ShowDayHeader	Gets or sets the value indicating whether the heading for the days of the week is displayed.
ShowGridLines	Gets or sets the value indicating whether the gridlines would be shown.
ShowNextPrevMonth	Gets or sets a value indicating whether next and previous month navigation elements are shown in the title section.
ShowTitle	Gets or sets a value indicating whether the title section is displayed.
TitleFormat	Gets or sets the format for the title section.
Titlestyle	Get the style properties of the title heading for the Calendar control.
TodayDayStyle	Gets the style properties for today's date on the Calendar control.
TodaysDate	Gets or sets the value for today's date.
UseAccessibleHeader	Gets or sets a value that indicates whether to render the table header <th> HTML element for the day headers instead of the table data <td> HTML element.
VisibleDate	Gets or sets the date that specifies the month to display.
WeekendDayStyle	Gets the style properties for the weekend dates on the Calendar control.

The Calendar control has the following three most important events that allow the developers to program the calendar control. They are:

Events	Description
SelectionChanged	It is raised when a day, a week or an entire month is selected.
DayRender	It is raised when each data cell of the calendar control is rendered.
VisibleMonthChanged	It is raised when user changes a month.

b)Working with the Calendar Control

Putting a bare-bone calendar control without any code behind file provides a workable calendar to a site, which shows the months and days of the year. It also allows navigation to next and previous months.

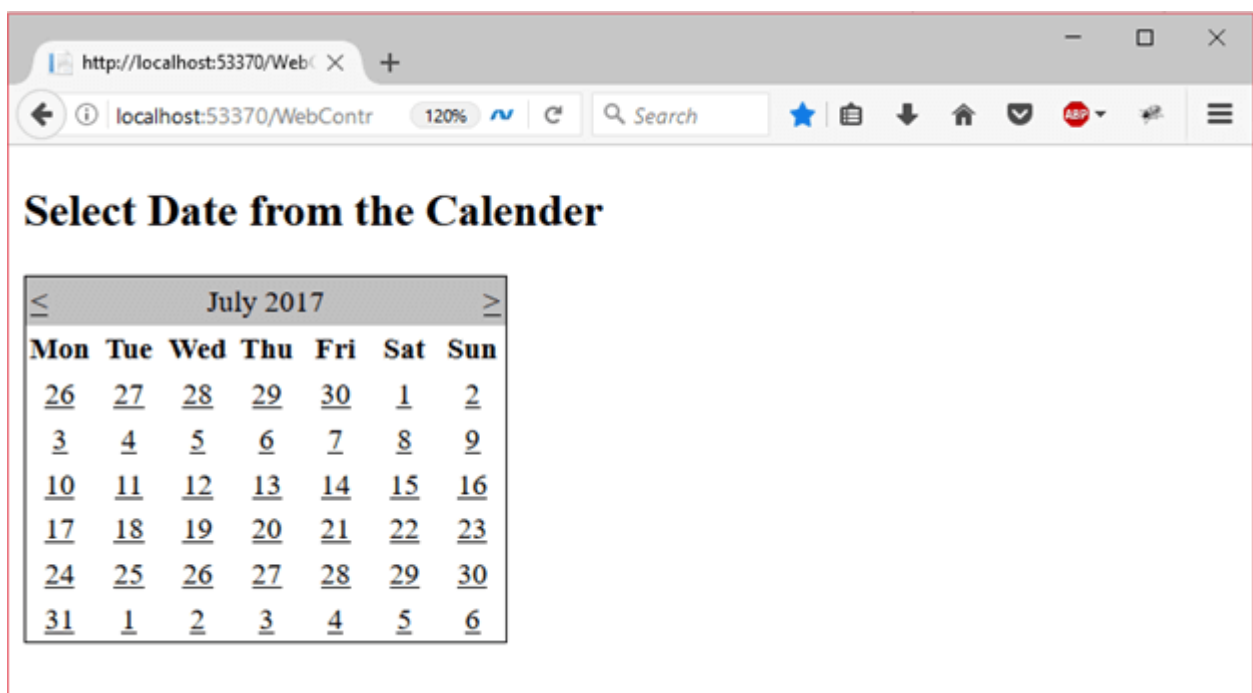


Fig 2.5 Working with Calender control

Calendar controls allow the users to select a single day, a week, or an entire month. This is done by using the SelectionMode property. This property has the following values:

Properties	Description
Day	To select a single day.
DayWeek	To select a single day or an entire week.
DayWeekMonth	To select a single day, a week, or an entire month.
None	Nothing can be selected.

The syntax for selecting days:

```
<asp:Calender ID = "Calendar1" runat = "server"
SelectionMode="DayWeekMonth">
</asp:Calender>
```

When the selection mode is set to the value DayWeekMonth, an extra column with the > symbol appears for selecting the week, and a >> symbol appears to the left of the days name for selecting the month.

<=	July 2010							>=
>>	Mon	Tue	Wed	Thu	Fri	Sat	Sun	
>	28	29	30	1	2	3	4	
>	5	6	7	8	9	10	11	
>	12	13	14	15	16	17	18	
>	19	20	21	22	23	24	25	
>	26	27	28	29	30	31	1	
>	2	3	4	5	6	7	8	

Fig 2.6 Calender control with Selection Mode

The following example demonstrates selecting a date and displays the date in a label. The content file code is as follows:

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

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

```

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

  <head runat="server">
    <title>
      Untitled Page
    </title>
  </head>

  <body>
    <form id="form1" runat="server">

      <div>
        <h3> Your Birthday:</h3>
        <asp:Calendar ID="Calendar1" runat="server"
SelectionMode="DayWeekMonth"
onselectionchanged="Calendar1_SelectionChanged">
          </asp:Calendar>
        </div>

        <p>Todays date is:
          <asp:Label ID="lblday" runat="server"></asp:Label>
        </p>

        <p>Your Birday is:
          <asp:Label ID="lblbday" runat="server"></asp:Label>
        </p>

      </form>
    </body>
  </html>

```

The event handler for the event SelectionChanged:

```
protected void Calendar1_SelectionChanged(object sender, EventArgs e)
{
    lblDay.Text = Calendar1.TodaysDate.ToShortDateString();
    lblDay.Text = Calendar1.SelectedDate.ToShortDateString();
}
```

When the file is run, it should produce the following output:

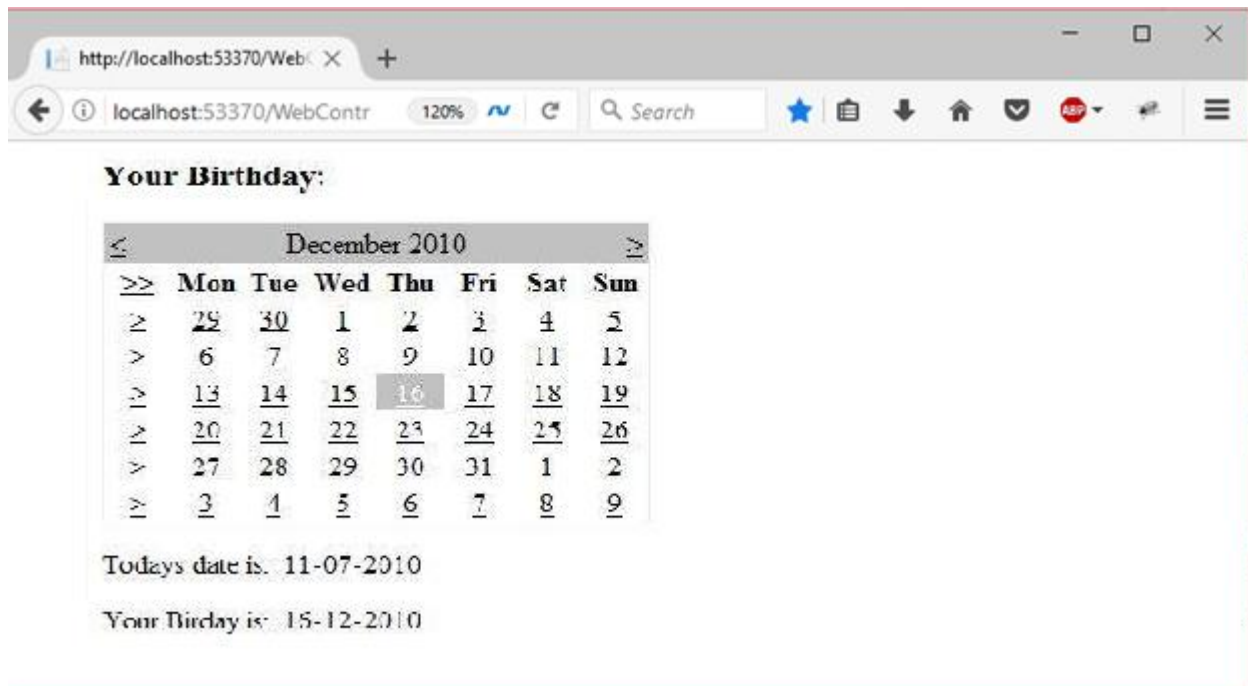


Fig 2.7 Calendar control with Selection changed event

c)Choosing A Date Format to Output from The Calendar

When you use the `Calendar1_SelectionChanged` event, the selected date is written out using the `ToShortDateString()` method. The Calendar control also allows you to write out the date in a number of other formats, as detailed in the following list:

- ❑ `ToFileTime`: Converts the selection to the local operating system file time:
1274739120000000000.
- ❑ `ToFileTimeUtc`: Converts the selection to the operating system file time, but instead of using the local time zone,

the UTC time is used: 1274736960000000000.

- ❑ `ToLocalTime`: Converts the current coordinated universal time (UTC) to local time:
12/12/2004 6:00:00 PM.
- ❑ `ToLongDateString`: Converts the selection to a human-readable string in a long format:
Monday, December 13, 2004.
- ❑ `ToLongTimeString`: Converts the selection to a time value (no date is included) of a long format: 12:00:00 AM.
- ❑ `ToOADate`: Converts the selection to an OLE Automation date equivalent: 38334.
- ❑ `ToShortDateString`: Converts the selection to a human-readable string in a short format:
12/13/2004.
- ❑ `ToShortTimeString`: Converts the selection to a time value (no date is included) in a short format: 12:00 AM.
- ❑ `ToString`: Converts the selection to the following: 12/13/2004 12:00:00 AM.
- ❑ `ToUniversalTime`: Converts the selection to universal time (UTC):
12/13/2004 6:00:00 AM.

d) Modifying the Style and Behavior of Calendar Control

Using Visual Studio, we customize the calendar controls from the Design view of the page.. Highlight the Calendar control and open the control's smart tag to see the Auto Format link. That gives a list of available styles that can be applied to the Calendar control.

Some of the styles are shown in Figure 2.8.

<div> <div><</div> <div>February 2006</div> <div>></div> </div> <table> <tr> <th>Su</th><th>Mo</th><th>Tu</th><th>We</th><th>Th</th><th>Fr</th><th>Sa</th></tr> <tr><td>29</td><td>30</td><td>31</td><td>1</td><td>2</td><td>3</td><td>4</td></tr> <tr><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td></tr> <tr><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td></tr> <tr><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td></tr> <tr><td>26</td><td>27</td><td>28</td><td>1</td><td>2</td><td>3</td><td>4</td></tr> <tr><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td></tr> </table>							Su	Mo	Tu	We	Th	Fr	Sa	29	30	31	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	1	2	3	4	5	6	7	8	9	10	11
Su	Mo	Tu	We	Th	Fr	Sa																																																	
29	30	31	1	2	3	4																																																	
5	6	7	8	9	10	11																																																	
12	13	14	15	16	17	18																																																	
19	20	21	22	23	24	25																																																	
26	27	28	1	2	3	4																																																	
5	6	7	8	9	10	11																																																	
<div> <div>January</div> <div>February 2006</div> <div>March</div> </div> <table> <tr> <th>Sun</th><th>Mon</th><th>Tue</th><th>Wed</th><th>Thu</th><th>Fri</th><th>Sat</th></tr> <tr><td>29</td><td>30</td><td>31</td><td>1</td><td>2</td><td>3</td><td>4</td></tr> <tr><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td></tr> <tr><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td></tr> <tr><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td></tr> <tr><td>26</td><td>27</td><td>28</td><td>1</td><td>2</td><td>3</td><td>4</td></tr> <tr><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td></tr> </table>							Sun	Mon	Tue	Wed	Thu	Fri	Sat	29	30	31	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	1	2	3	4	5	6	7	8	9	10	11
Sun	Mon	Tue	Wed	Thu	Fri	Sat																																																	
29	30	31	1	2	3	4																																																	
5	6	7	8	9	10	11																																																	
12	13	14	15	16	17	18																																																	
19	20	21	22	23	24	25																																																	
26	27	28	1	2	3	4																																																	
5	6	7	8	9	10	11																																																	
<div> <div>Jan</div> <div>February 2006</div> <div>Mar</div> </div> <table> <tr> <th>Sun</th><th>Mon</th><th>Tue</th><th>Wed</th><th>Thu</th><th>Fri</th><th>Sat</th></tr> <tr><td>29</td><td>30</td><td>31</td><td>1</td><td>2</td><td>3</td><td>4</td></tr> <tr><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td></tr> <tr><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td></tr> <tr><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td></tr> <tr><td>26</td><td>27</td><td>28</td><td>1</td><td>2</td><td>3</td><td>4</td></tr> <tr><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td></tr> </table>							Sun	Mon	Tue	Wed	Thu	Fri	Sat	29	30	31	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	1	2	3	4	5	6	7	8	9	10	11
Sun	Mon	Tue	Wed	Thu	Fri	Sat																																																	
29	30	31	1	2	3	4																																																	
5	6	7	8	9	10	11																																																	
12	13	14	15	16	17	18																																																	
19	20	21	22	23	24	25																																																	
26	27	28	1	2	3	4																																																	
5	6	7	8	9	10	11																																																	
<div> <div><</div> <div>February 2006</div> <div>></div> </div> <table> <tr> <th>Su</th><th>Mo</th><th>Tu</th><th>We</th><th>Th</th><th>Fr</th><th>Sa</th></tr> <tr><td>29</td><td>30</td><td>31</td><td>1</td><td>2</td><td>3</td><td>4</td></tr> <tr><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td></tr> <tr><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td></tr> <tr><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td></tr> <tr><td>26</td><td>27</td><td>28</td><td>1</td><td>2</td><td>3</td><td>4</td></tr> <tr><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td></tr> </table>							Su	Mo	Tu	We	Th	Fr	Sa	29	30	31	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	1	2	3	4	5	6	7	8	9	10	11
Su	Mo	Tu	We	Th	Fr	Sa																																																	
29	30	31	1	2	3	4																																																	
5	6	7	8	9	10	11																																																	
12	13	14	15	16	17	18																																																	
19	20	21	22	23	24	25																																																	
26	27	28	1	2	3	4																																																	
5	6	7	8	9	10	11																																																	
<div> <div>January</div> <div>February</div> <div>March</div> </div> <table> <tr> <th>Su</th><th>Mo</th><th>Tu</th><th>We</th><th>Th</th><th>Fr</th><th>Sa</th></tr> <tr><td>29</td><td>30</td><td>31</td><td>1</td><td>2</td><td>3</td><td>4</td></tr> <tr><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td></tr> <tr><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td></tr> <tr><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td></tr> <tr><td>26</td><td>27</td><td>28</td><td>1</td><td>2</td><td>3</td><td>4</td></tr> <tr><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td></tr> </table>							Su	Mo	Tu	We	Th	Fr	Sa	29	30	31	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	1	2	3	4	5	6	7	8	9	10	11
Su	Mo	Tu	We	Th	Fr	Sa																																																	
29	30	31	1	2	3	4																																																	
5	6	7	8	9	10	11																																																	
12	13	14	15	16	17	18																																																	
19	20	21	22	23	24	25																																																	
26	27	28	1	2	3	4																																																	
5	6	7	8	9	10	11																																																	
<div> <div><</div> <div>February 2006</div> <div>></div> </div> <table> <tr> <th>Su</th><th>Mo</th><th>Tu</th><th>We</th><th>Th</th><th>Fr</th><th>Sa</th></tr> <tr><td>29</td><td>30</td><td>31</td><td>1</td><td>2</td><td>3</td><td>4</td></tr> <tr><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td></tr> <tr><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td></tr> <tr><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td></tr> <tr><td>26</td><td>27</td><td>28</td><td>1</td><td>2</td><td>3</td><td>4</td></tr> <tr><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td></tr> </table>							Su	Mo	Tu	We	Th	Fr	Sa	29	30	31	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	1	2	3	4	5	6	7	8	9	10	11
Su	Mo	Tu	We	Th	Fr	Sa																																																	
29	30	31	1	2	3	4																																																	
5	6	7	8	9	10	11																																																	
12	13	14	15	16	17	18																																																	
19	20	21	22	23	24	25																																																	
26	27	28	1	2	3	4																																																	
5	6	7	8	9	10	11																																																	

Fig 2.8 Various styles of Calendar Control

2.5 AD ROTATOR SERVER CONTROL

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 AdRotator control allows you to specify the advertisement file and the type of window that the link should follow in the AdvertisementFile and the Target property respectively.

The basic syntax of adding an AdRotator is as follows:

```
<asp:AdRotator runat = "server" AdvertisementFile = "adfile.xml" Target =
"_blank" />
```

Before going into the details of the AdRotator control and its properties, let us look into the construction of the advertisement file.

The Advertisement File

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


```

<?xml          version="1.0"          encoding="utf-8"          ?>
<Advertisements>
<Ad>
<ImageUrl>~/Images/image1.gif</ImageUrl>
<NavigateUrl>http://roseindia.net</NavigateUrl>
<AlternateText>Roseindia</AlternateText>
<Keyword>Site1</Keyword>
</Ad>
<Ad>
<ImageUrl>~/Images/image2.png</ImageUrl>
<NavigateUrl>http://www.google.com</NavigateUrl>
<AlternateText>Google</AlternateText>
<Keyword>Site2</Keyword>
</Ad>
</Advertisements>

```

AdRotator.aspx (source page):

```

<%@          Page          Language="C#"          AutoEventWireup="true"
MasterPageFile="~/RoseindiaMaster.master"
CodeFile="AdRotator.aspx.cs"          Inherits="AdRotator"          %>
<asp:Content          ID="Content1"          runat="server"
contentplaceholderid="ContentPlaceHolder1">
<div>
<h2          style="color:Green">AdRotator          in          ASP.NET          4,          C#</h2>
<asp:AdRotator          ID="AdRotator1"
runat="server"
Width="468px"
Height="60px"
AdvertisementFile="~/XML/Adxml.xml"/>

```

```
</div>
</asp:Content>
```

Output:

View the page in browser is as follows:

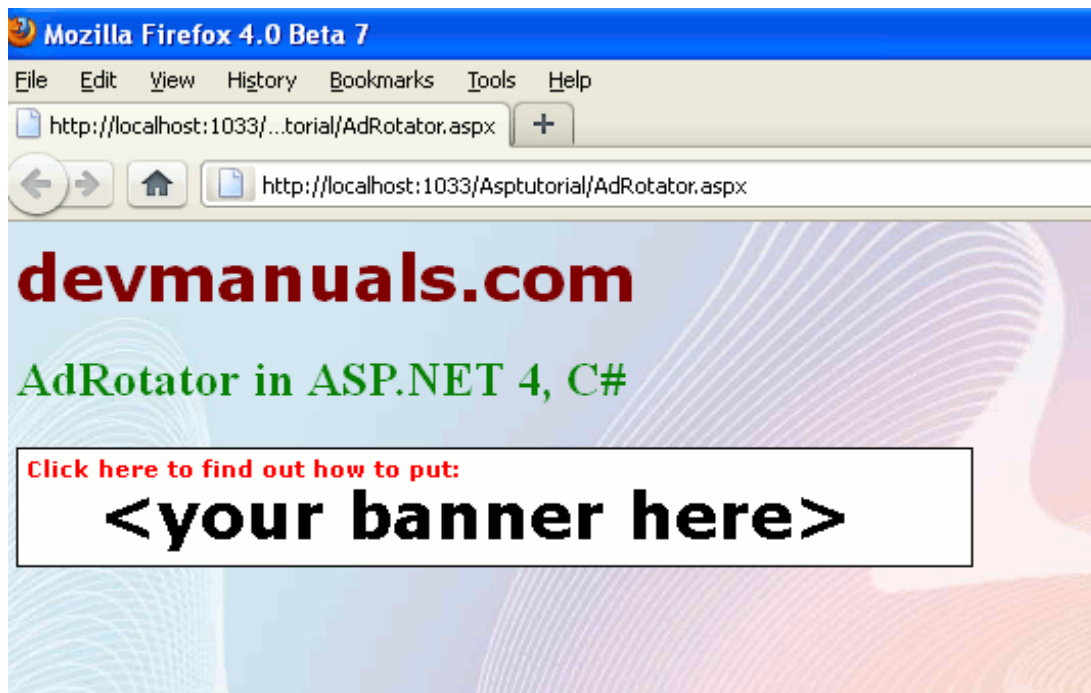


Fig 2.9 output of Adrotator before refresh

When you refresh the page the second image will be display as follows:



Fig 2.10 output of Adrotator after refresh

Like all XML files, the advertisement file needs to be a structured text file with well-defined tags delineating the data. There are the following standard XML elements that are commonly used in the advertisement file. Apart from these tags, customs tags with custom attributes could also be included

Element	Description
Advertisements	Encloses the advertisement file.
Ad	Delineates separate ad.
ImageUrl	The path of image that will be displayed.
NavigateUrl	The link that will be followed when the user clicks the ad.
AlternateText	The text that will be displayed instead of the picture if it cannot be displayed.
Keyword	Keyword identifying a group of advertisements. This is used for filtering.
Impressions	The number indicating how often an advertisement will appear.
Height	Height of the image to be displayed.
Width	Width of the image to be displayed.

a) Properties and Events of the AdRotator Class

The AdRotator class is derived from the WebControl class and inherits its properties. Apart from those, the AdRotator class has the following properties:

Properties	Description
AdvertisementFile	The path to the advertisement file.
AlternateTextFeild	The element name of the field where alternate text is provided. The default value is AlternateText.
DataMember	The name of the specific list of data to be bound when advertisement file is not used.
DataSource	Control from where it would retrieve data.
DataSourceID	Id of the control from where it would retrieve data.
Font	Specifies the font properties associated with the advertisement banner control.
ImageUrlField	The element name of the field where the URL for the image is provided. The default value is ImageUrl.
KeywordFilter	For displaying the keyword based ads only.
NavigateUrlField	The element name of the field where the URL to navigate to is provided. The default value is NavigateUrl.
Target	The browser window or frame that displays the content of the page linked.
UniqueID	Obtains the unique, hierarchically qualified identifier for the AdRotator control.

Following are the important events of the AdRotator class:

Events	Description
AdCreated	It is raised once per round trip to the server after creation of the control, but before the page is rendered

DataBinding	Occurs when the server control binds to a data source.
DataBound	Occurs after 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.
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.

2.6 INTERNET EXPLORER WEB CONTROL

Internet Explorer WebControls, which are a powerful collection of ASP.NET server controls. The WebControls implement a single-source authoring solution for four popular UI elements: MultiPage, TabStrip, Toolbar, and TreeView. The WebControls provide an authoring solution with widespread reach, by delivering HTML 3.2 compatible content to downlevel browsers .

ASP.NET Web forms detect the client browser capabilities and include Dynamic HTML (DHTML) behaviors in the Web pages downloaded to uplevel browsers.

The objects exposed by the DHTML behaviors and the ASP.NET controls are presented in the Internet Explorer WebControls Reference.

a) WebControls

This section illustrates the type of interface that can be created by each one. Each section focuses on one of the WebControls and links to the appropriate overviews and reference documentation.

i) TreeView

The TreeView control contains a hierarchy of TreeViewItem controls. We use the TreeView control to display information from a wide variety of data sources such as an XML file, site-map file, string, or from a database.

It provides a way to display information in a hierarchical structure by using collapsible nodes. The top level in a tree view are root nodes that can be expanded or collapsed if the nodes have child nodes. The following figure shows about the Treeview Control.

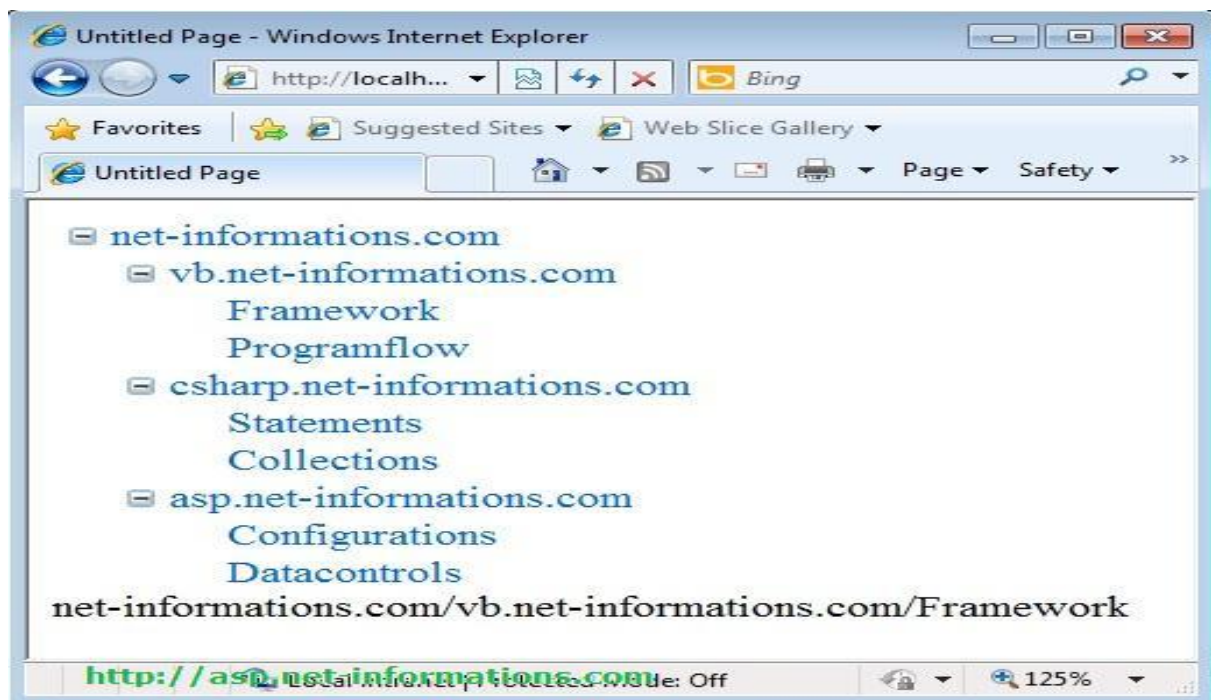


Fig 2.11 Sample Treeview Control

This control exposes both a server and client Object Model.

ii)ToolBar

The Toolbar can be used to author UI elements that render and function in ways similar to the toolbars in Windows applications. In uplevel browsers, the Toolbar can have rich interactive behaviour. For example, it can dock with other elements in a Web page or the browser window and can

modify its orientation accordingly. Like the other WebControls, the Toolbar can be customized with graphics elements and CSS. The Toolbar control exposes both a server and client Object Model.

iii) Tab Strip and MultiPage

The TabStrip is often used in combination with the MultiPage control. However, each control can also be used separately. The following example shows a typical use for these controls.

In the sample, the tabbing UI elements are authored with the TabStrip control. The MultiPage control is used as a container for pages of Web content, which are activated when a Tab is selected. A MultiPage is a container for a collection of PageView elements. Clicking the Tab navigates to a new PageView automatically.

For more detailed information on using these controls, see About the TabStrip WebControl and About the MultiPage WebControl. Also, see the TabStrip Reference and MultiPage Reference pages for links to the client-side and server-side references for these controls.

2.7 STATE MANAGEMENT

State management means to preserve **state** of a control, web page, object/data, and user in the application explicitly because all ASP.NET web applications are stateless, i.e., by default, for each page posted to the server, the state of controls is lost.

The current value of all the controls and variables for the current user in the current session is called the **State**.

2.7.1 Types of state management

There are two types of state management techniques: client side and server side.

a) Client side

1. Hidden Field
2. View State
3. Cookies
4. Control State

5. Query Strings

b) Server side

1. Session
2. Application

2.7.3 Levels of State Management

1. **Control level:** In ASP.NET, by default controls provide state management automatically.
2. **Variable or object level:** In ASP.NET, member variables at page level are stateless and thus we need to maintain state explicitly.
3. **Single or multiple page level:** State management at single as well as multiple page level i.e., managing state between page requests.
4. **User level:** State should be preserved as long as a user is running the application.
5. **Application level:** State available for complete application irrespective of the user, i.e., should be available to all users.
6. **Application to application level:** State management between or among two or more applications.

2.7.4 Client Side Methods

1. Hidden field

Hidden field is a control provided by ASP.NET which is used to store small amounts of data on the client. It store one value for the variable and it is a preferable way when a variable's value is changed frequently. Hidden field control is not rendered to the client (browser) and it is invisible on the browser. A hidden field travels with every request like a standard control's value.

Let us see with a simple example how to use a hidden field. These examples increase a value by 1 on every "No Action Button" click. The source of the hidden field control is.

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


In the code-behind page: **Default.aspx Code**

```

1. %@PageLanguage="C#"AutoEventWireup="true"CodeFile="Default.aspx.cs"Inherits="_Default"%>
2. <!DOCTYPEhtml>
3. <htmlxmlns="http://www.w3.org/1999/xhtml">
4. <headrunat="server">
5. <title></title>
6. </head>
7.
8. <body>
9. <formid="form1" runat="server">
10. <div>
11. <asp:HiddenFieldID="hdnfldCurrentDateTime" runat="server" />
12. <asp:LabelID="lblCurrentDateTime" runat="server" Text=""></asp:Label>
13. </div>
14. </form>
15. </body>
16.
17. </html>

```

Default.aspx.cs Code

```

1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Web;
5. using System.Web.UI;
6. using System.Web.UI.WebControls;
7. public partial class_Default: System.Web.UI.Page {
8.     protected void Page_Load(object sender, EventArgs e) {
9.         hdnfldCurrentDateTime.Value = DateTime.Now.ToString();

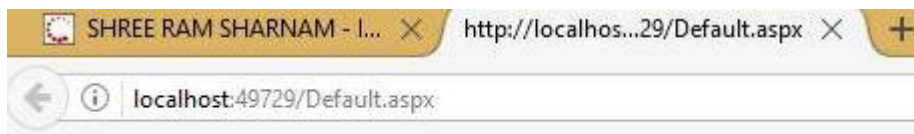
```

```

10.         lblCurrentDateTime.Text = Convert.ToString(hdnfldCurrent
           DateTime.Value);
11.     }
12. }

```

Output of the above program as follows:



HiddenField Value: 28-Dec-16 12:36:04 PM

Fig 2.11 Hidden Field demo

2. View State

View state is another client side state management mechanism provided by ASP.NET to store user's data, i.e., sometimes the user needs to preserve data temporarily after a post back, then the view state is the preferred way for doing it. It stores data in the generated HTML using hidden field not on the server.

View State provides page level state management i.e., 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 deserilization on each post back.

View state is enabled by default for all server side controls of ASP.NET with a property EnableviewState set to true.

3. Cookies

Cookie is a small text file which is created by the client's browser and also stored on the client hard disk by the browser. It does not use server memory. Generally a cookie is used to identify users. A cookie is a small file that stores user information.

Whenever a user makes a request for a page the first time, the server creates a cookie and sends it to the client along with the requested page and the client browser receives that cookie and stores it on the client machine either permanently or temporarily (persistent or non persistence). The next time the user makes a request for the same site, either the same or another page, the browser checks the existence of the cookie for that site in the folder. If the cookie exists it sends a request with the same cookie, else that request is treated as a new request.

Types of Cookies

a) 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.

Let us see how to create persistence cookies. There are two ways, the first one is:

```
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);
```

b. 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. Non Persistence cookies are useful for public computers.

Let us see how to create a non persistence cookies. There are two ways, the first one is:

```
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 create cookie :

How to read a cookie:

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

Let's understand persistence and non persistence cookies more clearly with a diagram:

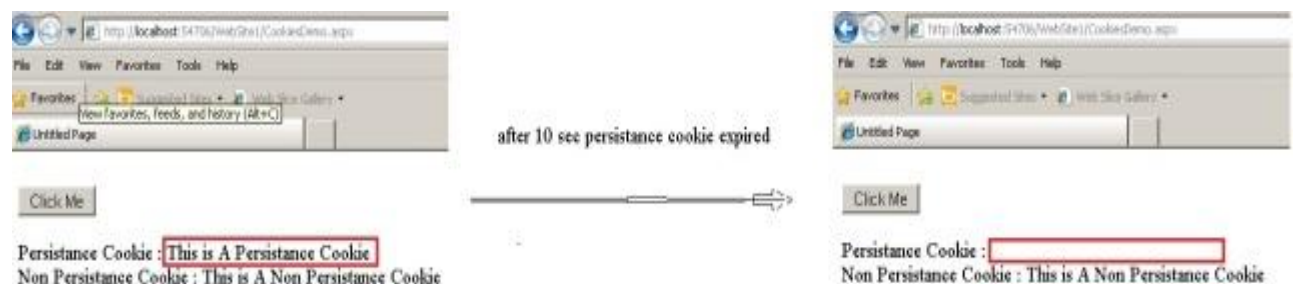


Fig 2.8 Persistence and Non-persistence of Cookies

Limitation of cookies:

The number of cookies allowed is limited and varies according to the browser. Most browsers allow 20 cookies per server in a client's hard disk folder and the size of a cookie is not more than 4096 bytes or 4 KB of data that also includes name and value data.

4. Control State

Control State is another client side state management technique. Whenever we develop a custom control and want to preserve some information, we can use view state but suppose view state is disabled explicitly by the user, the control will not work as expected. For expected results for the control we have to use Control State property. Control state is separate from view state.

How To Use Control State Property: Control state implementation is simple. First override the OnInit() method of the control and add a call for the Page.RegisterRequiresControlState() method with the instance of the control to register. Then override LoadControlState and SaveControlState in order to save the required state information.

2.7.5 Server side

1. Session

Session management is a very strong technique to maintain state. Generally session is used to store user's information and/or uniquely identify a user (or say browser). The server maintains the state of user information by using a session ID. When users make a request without a session ID, ASP.NET creates a session ID and sends it with every request and response to the same user.

How to get and set value in Session:

```
Session["Count"] = Convert.ToInt32(Session["Count"]) + 1; //Set Value to The Session
Label2.Text = Session["Count"].ToString(); //Get Value from the Sesion
```

Let us see an example where we save the count of button clicks in a session, and save the “number of redirects to the same page” button click in a query string.

Here we have set the expiry to 10 minutes. After starting the application, the application variable exists till the end of the application.

A session variable will expire after 10 minutes (if it is idle). A query string contains the value in URL so it won't depend on the user idle time and could be used by the server anytime it is passed with a request.

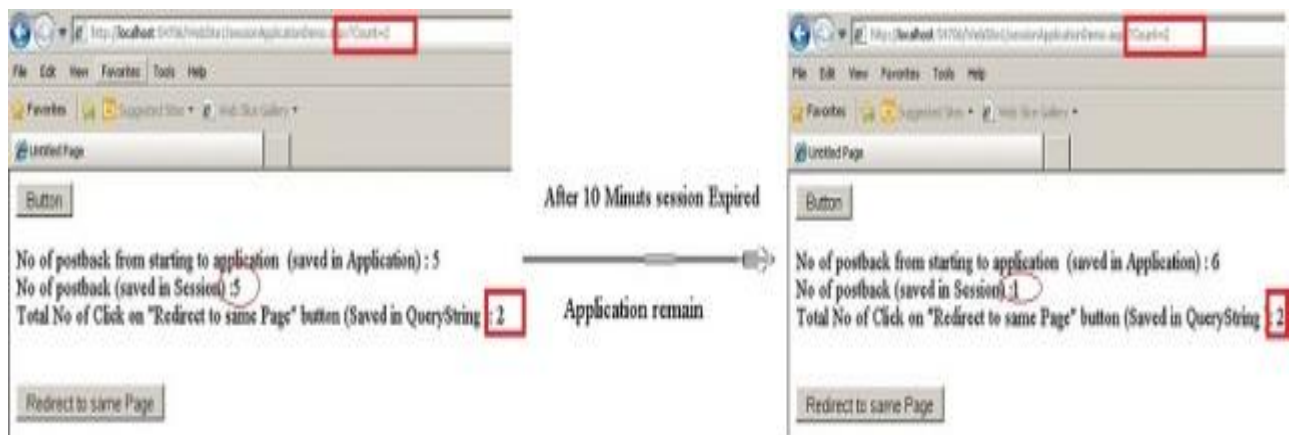


Fig 2.9 Session Management

Session Events in ASP.Net

To manage a session, ASP.NET provides two events: `session_start` and `session_end` that is written in a special file called *Global.asax* in the root directory of the project.

Session_Start: The `Session_start` event is raised every time a new user makes a request without a session ID, i.e., new browser accesses the application, then a `session_start` event raised.

```
void Session_Start(object sender, EventArgs e)
{
    Session["Count"] = 0; // Code that runs when a new session is started
}
```

Session_End: The `Session_End` event is raised when session ends either because of a time out expiry or explicitly by using `Session. Abandon()`. The `Session_End` event is raised only in the case of In proc mode not in the state server and SQL Server modes.

There are four session storage mechanisms provided by ASP.NET:

- In Proc mode
- State Server mode
- SQL Server mode
- Custom mode

In Process mode: In proc mode is the default mode provided by ASP.NET. In this mode, session values are stored in the web server's memory (in IIS). If there are more than one IIS servers then session values are stored in each server separately on which request has been made. Since the session values are stored in server, whenever server is restarted the session values will be lost.

In State Server mode: This mode could store session in the web server but out of the application pool. But 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 asp.net_state service.

In SQL Server 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.

Custom Session mode: Generally we should prefer in proc state server mode or SQL Server mode but 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.

Attributes	Description
Cookieless true/false	Indicates that the session is used with or without cookie. cookieless set to true indicates sessions without cookies is used and cookieless set to false indicates sessions with cookies is used. cookieless set to false is the

	default set.
timeout	Indicates the session will abound if it is idle before session is abounded explicitly (the default time is 20 min).
StateConnectionString	Indicates the session state is stored on the remote computer (server). This attribute is required when session mode is StateServer
SqlConnectionString	Indicates the session state is stored in the database. This attribute is required when session mode is SqlServer.

2. Application

Application state is a server side state management technique. The data stored in application state is common for all users of that particular ASP.NET application and can be accessed anywhere in the application. It is also called application level state management. Data stored in the application should be of small size.

How to get and set a value in the **application object**:

```
Application["Count"] = Convert.ToInt32(Application["Count"]) + 1; //Set
Value to The Application Object
Label1.Text = Application["Count"].ToString(); //Get Value from the
Application Object
```

Application events in ASP.NET

There are three types of events in ASP.NET. Application event is written in a special file called *Global.asax*. This file is not created by default, it is created explicitly by the developer in the root directory. An application can create more than one *Global.asax* file but only the root one is read by ASP.NET.

Application_start: The Application_Start event is raised when an app domain starts. When the first request is raised to an application then the Application_Start event is raised. Let's see *Global.asax* file.


```

void Application_Start(object sender, EventArgs e)
{
    Application["Count"] = 0;
}

```

Application_Error: It is raised when an unhandled exception occurs, and we can manage the exception in this event.

Application_End: The Application_End event is raised just before an application domain ends because of any reason, may IIS server restarting or making some changes in an application cycle.

2.8 KEY TERMS

Web Forms: Web forms are made up of different types of HTML elements that are constructed using raw HTML form elements.

Validation: Validation is basically the act of comparing something to a given set of rules and determining if it satisfies the criteria those rules represent.

Server side Validation: The user input validation take place on the Server Side during a post back session is called Server Side Validation.

Client Side Validation: The user input validation take place on the Client Side (web browser) is called Client Side Validation. Client Side Validation does not require a postback.

State management: **State Management** means to preserve **state** of a control, web page, object/data, and user in the application explicitly because all ASP.NET web applications are stateless, i.e., by default, for each page posted to the server, the state of controls is lost.

State: The current value of all the controls and variables for the current user in the current session is called the **State**.

Session: Session is used to store user's information and/or uniquely identify a user (or say browser).

Cookies: Cookie is a small text file which is created by the client's browser and also stored on the client hard disk by the browser.

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.

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.

2.8.1 TWO MARKS

1. What is Web forms?

Web Forms are pages that users request using their browser. Any ASPX files are usually referred to as Web Forms or Web Form Pages because they normally process form input.

2. What is Validation?

Validation is basically the act of comparing something to a given set of rules and determining if it satisfies the criteria those rules represent.

3. List out the types of Validation?

- a) Server side Validation
- b) Client side Validation

4. Define Server side Validation.

The user input validation take place on the Server Side during a post back session is called Server Side Validation. **For Example:** Suppose that the end user clicks the Submit button on a form after filling out some information. This form is packaged in a request and sent to the server where the application resides. At this point in the request/response cycle, we can run validation checks on the information submitted

5. Define Client Side Validation.

The user input validation take place on the Client Side (web browser) is called Client Side Validation. Client Side Validation does not require a postback. **For example:** if the user enter an invalid email format, we can

show an error message immediately before the user move to the next field, so the user can correct every field before they submit the form.

5. List various validation controls.

1. RequiredFieldValidator
2. RangeValidator
3. CompareValidator
4. RegularExpressionValidator
5. CustomValidator
6. ValidationSummary

6. Write the syntax of Required Field Validator.

```
<asp:RequiredFieldValidator ID="candidate"
  runat="server" ControlToValidate ="cname"
  ErrorMessage="error message to show"
  InitialValue="Intial value to show">
</asp:RequiredFieldValidator>
```

7. Write the purpose of compare validator control.

The CompareValidator control compares a value in one control with a fixed value or a value in another control. The CompareValidator control allows to make comparisons between two form elements as well as to compare values contained within form elements to constants that you specify.

8. Define State management.

State Management means to preserve **state** of a control, web page, object/data, and user in the application explicitly because all ASP.NET web applications are stateless, i.e., by default, for each page posted to the server, the state of controls is lost.

9. Define State.

The current value of all the controls and variables for the current user in the current session is called the **State**.

10. What is the use of Session?

Session is used to store user's information and/or uniquely identify a user (or say browser).

11. List out the session storage mechanisms?

- In Proc mode
- State Server mode
- SQL Server mode
- Custom mode

12. Define Cookies.

Cookie is a small text file which is created by the client's browser and also stored on the client hard disk by the browser.

12. List out the types of cookies.

Persistence Cookie and Non-Persistence Cookie

13. Define 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.

14. Define 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.

15. List out various elements in Internet Explorer Web Control.

MultiPage, TabStrip, Toolbar, and TreeView

16. List out the various types of State Management.

Client side and Server side

17. List out the Client side state management element

Hidden Field, View State, Cookies, Control State and Query Strings

18. List out the various types of Server side state management.

Session and Application

2.8.2 FIVE MARKS

1. Differentiate between Client side and server side validation techniques.
2. Write short note on various validation controls in ASP. Net.

3. What is the purpose of Required Field Validator Control? Explain it with example.
4. Explain Range Validator Control with example.
5. Explain Ad rotator Control.
6. Write a short note on Internet Explorer Web Control.
7. Write a short note on Session Event in ASP .net.

2.8.3 TEN MARKS

1. Enumerate State Management with example
2. Explain Internet Explorer Web controls.
3. What is the purpose of Ad Rotator Server Control with example.
4. Explain Calender control with properties and events.
5. Explain Required Field Validator Control with example.
6. Explain Range validator control with example.
7. Explain cookies with example.

UNIT – III

ADO .Net

3.1 ARCHITECTURE OF ADO.NET

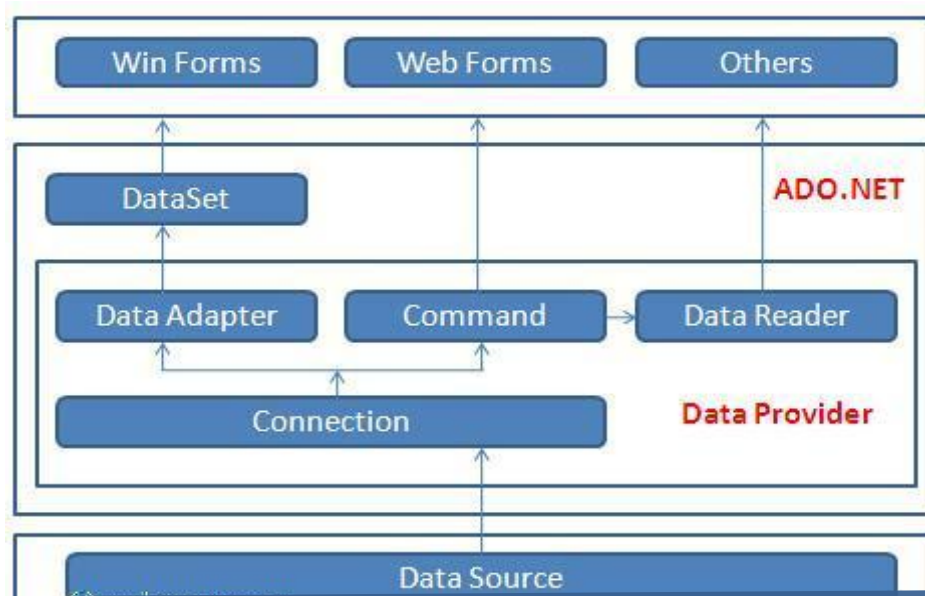


FIG 2.1 ARCHITECTURE OF ADO.Net

3.1.1 ADO.NET OBJECT MODEL:

- ADO.NET consist of a set of Objects that expose data access services to the .NET environment.
- It is a data access technology from Microsoft .Net Framework , which provides communication between relational and non relational systems through a common set of components .
- All ADO.NET related functionality appears under the System.Data namespace.
- Data Access in ADO.NET relies on two components :
 1. .NET Data Providers.
 2. DataSet

3.1.1.1 DATA PROVIDERS AND DATASET

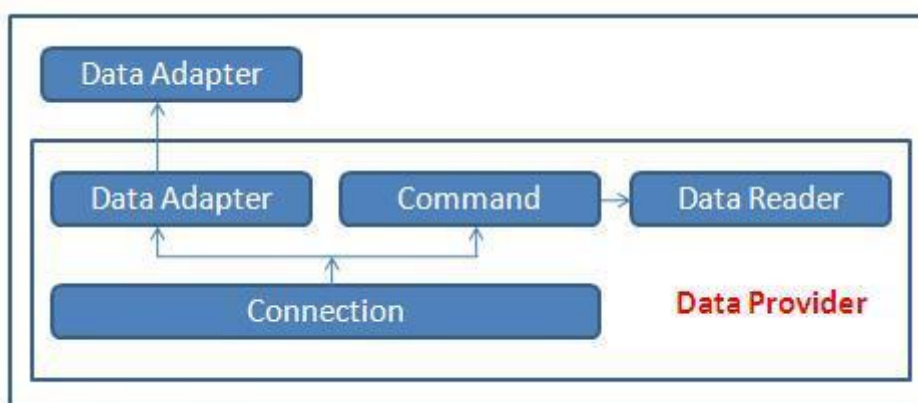


FIG 3.2 DATA PROVIDERS

- The Data Provider classes are meant to work with different kinds of data sources. They are used to perform all data-management operations on specific databases.
- Data Set class provides mechanisms for managing data when it is disconnected from the data source.

DATA PROVIDERS

The .Net Framework includes mainly three Data Providers for ADO.NET. They are

- Microsoft SQL Server Data Provider
- OLEDB Data Provider and
- ODBC Data Provider .

CONNECTION OBJECTS

- i. SQL Server uses the SqlConnection object ,
- ii. OLEDB uses the OleDbConnection Object and
- iii. ODBC uses OdbcConnection Object respectively.

A data provider contains

- ❖ Connection
- ❖ Command
- ❖ DataAdapter
- ❖ DataReader objects.

These four objects provides the functionality of Data Providers in the ADO.NET.

Connection

- The Connection Object provides physical connection to the Data Source.
- Connection object needs the necessary information to recognize the data source and to log on to it properly, this information is provided through a connection string.
 - i. SQL Server uses the SqlConnection object ,
 - ii. OLEDB uses the OleDbConnection Object and
 - iii. ODBC uses OdbcConnection Object respectively.

Command

- The Command Object uses to perform SQL statement or stored procedure to be executed at the Data Source.
- The command object provides a number of Execute methods that can be used to perform the SQL queries in a variety of fashions.

- Examples of command objects are SqlCommand, OracleCommand and so on.
- A Command needs to be able to accept parameters.
- The Parameter object of ADO.NET allows commands to be more flexible and accept input values and act accordingly.

DataReader

- The DataReader Object is a stream-based , forward-only, read-only retrieval of query results from the Data Source, which do not update the data.
- DataReader requires a live connection with the database and provides a very intelligent way of consuming all or part of the result set.
- The disadvantage of using a DataReader object is that it requires an open database connection and increases network activity.

DataAdapter

- DataAdapter Object populate a Dataset Object with results from a Data Source .
- It is a special class whose purpose is to bridge the gap between the disconnected Dataset objects and the physical data source.
- Examples of DataAdapters are SqlDataAdapter, OracleDataAdapter and so on. It has commands like Select, Insert, Update and Delete .
- Select command is used to retrieve data from the database.
- Insert, update and delete commands are used to send changes to the data in dataset to database.

3.1.1.2 DATASET

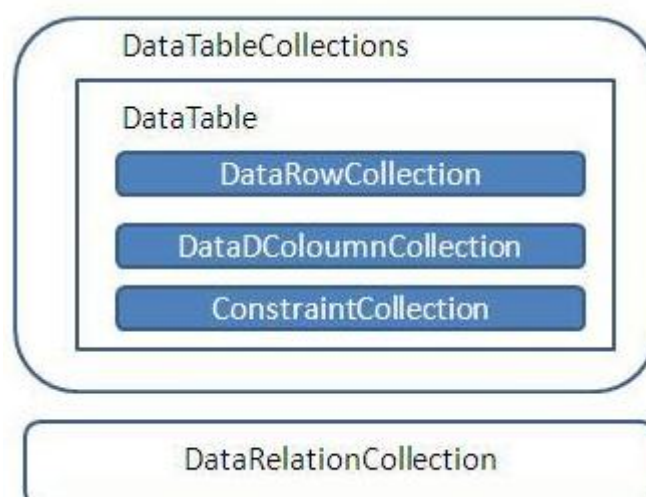


FIG 3.3 DATASET

- DataSet provides a disconnected representation of result sets from the Data Source
- It is completely independent from the Data Source.
- DataSet provides much greater flexibility when dealing with related Result Sets.
- DataSet contains rows, columns, primary keys, constraints, and relations with other DataTable objects.
- It consists of a collection of DataTable objects that you can relate to each other with DataRelation objects.
- The DataAdapter Object provides a bridge between the DataSet and the Data Source.

3.2 CONNECTED AND DISCONNECTED DATABASE

Connected Architecture of ADO.NET

- The architecture of ADO.net, in which connection must be opened to access the data retrieved from database is called as connected architecture.
- Connected architecture was built on the classes connection, command, datareader and transaction.



FIG 3.4 CONNECTED DATABASE

Disconnected Architecture in ADO.NET

- The architecture of ADO.net in which data retrieved from database can be accessed even when connection to database was closed is called as disconnected architecture.

- Disconnected architecture of ADO.net was built on classes connection, dataadapter, command builder and dataset and dataview.
- Disconnected architecture is a method of retrieving a record set from the database and storing it giving you the ability to do many operations on the data in memory.. A method of using disconnected architecture is using a Dataset.

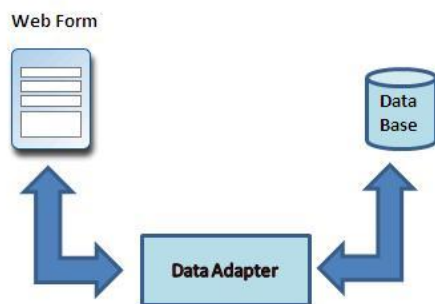


FIG 3.5 DISCONNECTED DATABASE

DataReader is Connected Architecture since it keeps the connection open until all rows are fetched one by one

DataSet is **DisConnected** Architecture since all the records are brought at once and there is no need to keep the connection alive

3.2.1 Difference between Connected and disconnected Database

Connected	Disconnected
It is connection oriented.	It is disconnection oriented.
Datareader	DataSet
Connected methods gives faster performance	Disconnected get low in speed and performance.
connected can hold the data of single table	disconnected can hold multiple tables of data
connected you need to use a read only forward only data reader	disconnected you cannot
Data Reader can't persist the data	Data Set can persist the data
It is Read only, we can't update the data.	We can update data

3.3 CREATING DATABASE IN ADO.NET

There are a few things you should make sure you understand/obtain before you begin. Take a look below:

- Starting up a command prompt in Windows or Linux.
- .NET framework 4.5 or greater installed and ready to go.
- A text editor.
- An ADO.NET Database Driver contained in products such as MySQL, PostgreSQL or RDM.

3.3.1 STEPS TO CREATING APPLICATION

Step 1 Open a command line prompt

Change to the directory in which you have installed the files for the sample.

Step 2 Viewing .cs file

Using text editor, view the file “HelloWorldADO.NET.java”.

Step 3 Viewing sample class

class can contain the same name as the .cs file containing the class. It should appear as follows:

```
Namespace HelloWorldApplication {
class HelloWorldADO.NET {
...
}
}
```

In this example everything is done within this class.

Step 4 Examining the main method

The main method is the entry point for your program. For this simple example, we are only using one .cs file. Therefore, the class will contain the main method as shown below. We will be accepting no arguments to this program.

```
static void main() {
...
}
```

Step 5 Creating and initializing your Connection Object

You will initialize your Connection object before you have access to any of the methods it contains. When you are done with the object, simply add a finally block that performs the corresponding close() method, and the outermost block will contain catch block to handle all possible Exceptions. This will be easier to see with the full code.

```
RdmConnection connection = new
RdmConnection("host=localhost;database=hello_worldADO");
try {
...
}
} catch (Exception exception) {
...
} finally {
Conn.close();
}
```

Step 6 Creating Statement Object

The newly created Connection object connection has a method in it called createCommand() that will return a RdmCommand object. Use that object with this Connection to the database.

```
RdmCommand command = connection.createCommand();
try {
...
} finally {
command.close();
}
```

Step 7 Execute Statements to Create or Open the Database

Using the RdmCommand object command just created, execute several different methods depending on the type of statement want to execute.

For example, if you would like to execute a SQL statement such as
 "OPEN database_name" or

`"DELETE * FROM table_name"`

In this example, we will create the database programmatically. In this example, the database is trivial, consisting of a single table named `hello_table` containing a single character column named `foo`. The sequence will create a table if it doesn't yet exist, or just open it if it does exist.

```
try {
RdmTransaction rdmtrans = connection.BeginTransaction();
command.CommandText = "CREATE TABLE hello_table (foo char(31))";
command.ExecuteNonQuery();
rdmtrans.commit(); // now the database physically exists
} catch (Exception exception) {
// we are here if database exists
}
```

Step 8 Inserting a new Row using the Statement Object

To insert a single row into this database, we use the `ExecuteNonQuery()` method, which is used for complete (unprepared) `INSERT`, `UPDATE` or `DELETE` statements. This implicitly starts a transaction, which will be one unit of update work applied to the database atomically. One `INSERT` is shown below with a parameter binding, but more could be added at this point.

```
command.CommandText = "INSERT INTO hello_table(foo) VALUES(?)";
command.CommandText = insertString;
RdmParameter parameter = new RdmParameter();
parameter.RdmType = RdmType.AnsiString;
parameter.Direction = ParameterDirection.Input;
parameter.Value = "Hello World!";
command.Parameters.Add(parameter);
command.ExecuteNonQuery();
```

Step 9 Committing Changes

In order to have changes finalized in the database perform a transaction commit. In ADO.NET this is done through a method in the `RdmTransaction` object. The method we will be using is `RdmTransaction.Commit()` and that will finalize any changes made during a transaction.

```
rdmtrans.Commit(); //Commits all changes
```

Step 10 Creating Result Set Object (retrieving data from the database)

In ADO.NET, when we want to retrieve data from the database, perform a SQL SELECT statement using your Command object with an execute method that returns a Result Set object. This method is called `Command.ExecuteReader()`.

This means it will execute the specified Query and return the Query results in the given Reader.

```
command.CommandText = "SELECT * FROM hello_table";
RdmDataReader reader = command.ExecuteReader();
try {
    ...
} finally {
    reader.Close();
}
```

Step 11 Accessing the Result Set

In order to access every piece of data in your Result Set, you must iterate through it. A method is provided within the Result Set to check if the next result in the Result Set is NULL, meaning no more data. If the method `reader.Read()` returns TRUE then there is data in the database and you can retrieve it from your result set.

To access the data inside the Result Set you must perform a getter method. There are numerous getter methods available to retrieve the specific data type from the Result Set. In this example we want a string, therefore we

use the `reader.getString()` method, with the parameter being the column (first/only column is 0) you are retrieving from.

Take a look at the code below to see an example of how this can be done.

```
while(reader.Read() != false)
{
    Console.WriteLine(reader.GetString(0));
}
```

This loop will retrieve all rows in the result set. When this sample program is run for the first time, there will be only one row. If you run it multiple times, you will find one row for each time it has been run.

Step 12 Deallocating Resources

Here we will deallocate all of the resources you used above. In this case, our resources are each object that we used above, being `Connection` object, `Statement`, and `Result Set` objects. For each nested try block you will have a `finally` block, which performs the corresponding close method. These statements have been shown in context above, but here are the cleanup calls in sequence from the code.

```
}
finally
{
    reader.Close ();
}
}
finally
{
    command.Close ();
}
}
catch (Exception exception) {
    Console.WriteLine ("Exception : " + exception.ToString ());
}
```

```

finally
{
connection.Close ();
}

```

Step 13 Final Catch and Finally block

The very last block contains both a catch block and a finally block. The catch block determines what to do if an exception was thrown in the code above. The finally block will be executed regardless of an exception being thrown. Here we will deallocate our Connection object.

```

}
catch (Exception exception)
{
WriteLine("Exception      :      "      +      exception.ToString());
}
finally
{
connection.Close();
}

```

Step 14 Compiling your application

If your ADO.NET is installed correctly you should have access to the c#, called csc. In order to compile, you must use the csc compiler. The format looks like this:

```
csc {main_class.cs (entry point to program)}
```

In this case you would type:

```
csc HelloWorldADO.cs
```

You should see no warnings, and after completion a .class file will have been generated. Your directory should contain:

```
HelloWorldADO.cs
```

```
HelloWorldADO.NET.a
```


Step 15 Running the program

Running the program is as simple as typing “java {executable name}”. In this case you would have “HelloWorldADO” as that is the entry point to your program. If everything works as expected you should see something like the following as displayed in a Windows Command Prompt:

3.4 CREATE CONNECTION USING ADO.NET OBJECT MODEL

Connection Object is used for connecting your application to data source or database. It carries required authentic information like username and password in the connection string and opens a connection. You need to different type of connection object for different type of data providers.

For example:

OLEDB	–	OleDbConnection
SQLServer	–	SqlConnection
ODBC	–	OdbcConnection
Oracle	–	OracleConnection

3.4.1 WHAT IS CONNECTION STRING?

Connection String combines all the required authentic information that is used for connecting to a Data Source, like Server Name, Database Name, User Name, Password etc. It is just a single line string that is used by connection object to connect to the database. A connection string looks like this.

Data Source=.\SQLEXPRESS;Initial Catalog=TestDB; Integrated Security=True

or,

Data Source=.\SQLEXPRESS;Initial Catalog=TestDB;User

ID=sa;Password=System123;Pooling=False

3.4.2 HOW TO STORE CONNECTION STRING IN WEB.CONFIG FILE?

In order to connect with Database, it is mandatory to keep connection string in a safe and centralized location. It is not recommended to writing

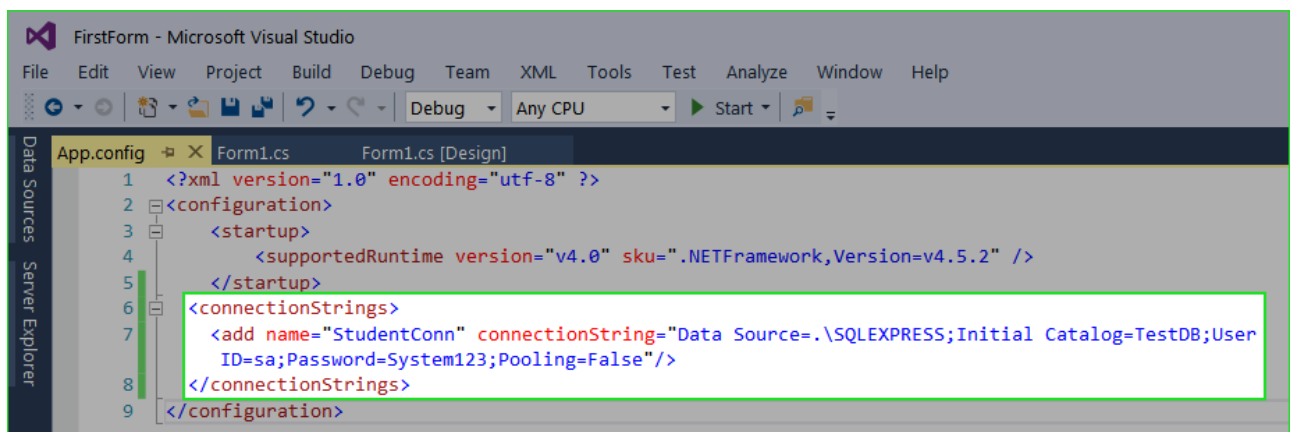
connection string in each and every connection. We can store the connection string in Web.config file, app.config file or into a class file.

Web.config **File – Add and Retrieve Connection String**

If you are developing ASP.Net Project or ASP Web Project then you can store the connection string in **Web.config** file.

1. Open **Web.config** file from solution Explorer
2. Paste following code Just Before `</Configuration>`

1. `<connectionStrings>`
2. `<add name="StudentConn" connectionString="Data Source=.\SQLEXPRESS;Initial Catalog=StudentDB;Integrated Security=True;Pooling=False"/>`
3. `</connectionStrings>`



Access Connection String from web.config

You can access this connection string in ASP.NET MVC, like this

1. `using System.Configuration;`
- 2.
3. `namespace ConnectionString_Example.Controllers`
4. `{`
5. `public class conString`
6. `{`
7. `public string getConString()`
- 8.

```

9.          {
10.          string constring =
    ConfigurationManager.ConnectionStrings["studentconn"].ToString();
11.          return constring;
12.          }
13.      }
14.  }

```

File – Add and Retrieve Connection String

If you are working on the windows form, you can save connection string in **App.config** file.

Add Connection String into app.config

```

1.      <connectionStrings>
2.      <add      name="StudentConn"      connectionString="Data
    Source=.\SQLEXPRESS;Initial      Catalog=TestDB;User
    ID=sa;Password=System123;Pooling=False"/>
3.      </connectionStrings>

```

Retrieve Connection String from app.config

```

1.  using System;
2.  using System.Windows.Forms;
3.  using System.Data.SqlClient;
4.  using System.Configuration;
5.
6.  namespace FirstForm
7.  {
8.      public partial class Form1 : Form
9.      {
10.         public Form1()
11.         {
12.             InitializeComponent();
13.         }
14.

```

```

15.     private void button1_Click(object sender, EventArgs e)
16.     {
17.         var ConString =
            ConfigurationManager.ConnectionStrings["StudentConn"].ConnectionString;
18.         SqlConnection con = new SqlConnection(ConString);
19.         con.Open();
20.     }
21. }
22. }

```

3.4.3 CONNECT TO DATASOURCE

There are 5 steps to connecting database.

1. Add Namespace: using System.Data.SqlClient;
2. Create Connection Object and Pass Connection String as Parameter.
3. Open Connection
4. Execute SQL Query
5. Close the Connection.

Example

```

1.     SqlConnection con = new SqlConnection("Data
Source=.\SQLEXPRESS;Initial Catalog=TestDB;User
ID=sa;Password=System123;Pooling=False";      );
2.     con.Open();
3.     // Update, Insert Delete Job in Table
4.     ss con.Close();

```

3.5 DISPLAY DATA ON DATABOUND CONTROLS

Data bound controls used in ASP.NET is to display data in various forms and do various database activities such as Add, Edit, Update and Delete operations. The control makes the data more organized and presents the data in an efficient way for the viewers.

Under the Data tab of the Visual Studio Toolbox, we can get several controls under the Data tab that could be used to display data from a data source like a database or XML file.

The standard ASP.NET data presentation controls are:

- DataList
- DetailsView
- FormView
- GridView
- ListView
- Repeater

We can divide these data presentation controls into the following two main groups. First of all, we go to the first group that includes the four controls Repeater, DataList, GridView and ListView.

3.5.1 REPEATER CONTROL

The Repeater control was introduced with ASP.NET 1.0. The ASP.NET Repeater is a basic container control that allows you to create custom lists from any data available to the page. It provides a highly customized interface. It renders a read-only template.

The Repeater control is a Data Bind Control, also known as container controls. The Repeater control is used to display a repeated list of items that are bound to the control. This control may be bound to a database table, an XML file, or another list of items. It has no built-in layout or styles, so you must explicitly declare all layout, formatting and style tags within the controls templates. The Repeater repeats a layout of HTML you write, it has the least functionality of the rest of the three controls.

The Repeater control supports the following features:

- List format
- No default output
- More control and complexity
- Item as row
- Paging, Sorting and Grouping requires custom code writing
- only Web control that allows you to split markup tags across the templates
- no built-in selection capabilities
- no built-in support for edit, insert and delete capabilities
- no built-in support for paging, sorting and grouping capabilities

- no built-in layout or styles, need to declare all layout, formatting and style tags explicitly within the control's templates
- Strictly emits the markup specified in its templates, nothing more and nothing less.

3.5.2 DATALIST CONTROL

The DataList control was introduced with ASP.NET 1.0. DataList allows you to repeat columns horizontally or vertically. The DataList control renders data as a table and enables you to display data records in various layouts, such as ordering them in columns or rows.

We can configure the DataList control to enable users to edit or delete a record in the table. We can use a DataList control where we need a single-column list. The DataList control works like the Repeater control, used to display the data in a repeating structure, such as a table.

It displays data in a format that you can define using a template and styles. However, it arranges the data defined in the template within various HTML structures. This includes options for horizontal or vertical layout and it also allows you to set how the data should be repeated, as flow or table layout. The DataList control does not automatically use a data source control to edit data.

The DataList control supports the following features

- Support for binding data source controls such as SqlDataSource, LinqDataSource and ObjectDataSource
- Directional rendering
- Good for columns
- Item as cell
- Updatable
- Control over Alternate item
- Paging function needs handwriting.

3.5.3 GRIDVIEW CONTROL

ASP.NET provides a number of tools for showing tabular data in a grid, including the GridView control. It was introduced with ASP.NET 2.0. The GridView control is used to display the values of a data source in a

table. Each column represents a field where each row represents a record. It can also display empty data.

The GridView control provides many built-in capabilities that allow the user to sort, update, delete, select and page through items in the control. The GridView control can be bound to a data source control, in order to bind a data source control, set the DataSourceID property of the GridView control to the ID value of the data source control. It's considered a replacement for the DataGrid control from .NET 1.1. Therefore, it is also known as a super DataGrid.

The GridView control offers improvements such as the ability to define multiple primary key fields, improved user interface customization using bound fields and templates and a new model for handling or canceling events. Performance is slow compared to DataGrid and ListView.

The GridView control supports the following features

- Improved data source binding capabilities
- Tabular rendering – displays data as a table
- Item as row
- Built-in sorting capability
- Built-in select, edit and delete capabilities
- Built-in paging capability
- Built-in row selection capability
- Multiple key fields
- Programmatic access to the GridView object model to dynamically set properties, handle events and so on
- Richer design-time capabilities
- Control over Alternate item, Header, Footer, Colors, font, borders, and so on.
- Slow performance as compared to Repeater and DataList control

3.5.4 LISTVIEW CONTROL

The ListView control was introduced with ASP.NET 3.5. The ListView control resembles the GridView control. The only difference between them is

that the ListView control displays data using user-defined templates instead of row fields.

Creating own templates gives more flexibility in controlling how the data is displayed. It enables you to bind to data items that are returned from a data source and display them. The data can be displayed in pages where you can display items individually, or you can group them.

The template contains the formatting, controls and binding expressions that are used to lay out the data. The ListView control is useful for data in any repeating structure, similar to the DataList and Repeater controls. It implicitly supports the ability to edit, insert and delete operations, as well as sorting and paging functionality.

The ListView control supports the following features

- Binding to data source controls Customizable appearance through user-defined templates and styles.
- Built-in sorting and grouping capabilities
- Built-in insert, edit and delete capabilities
- Support for paging capabilities using a DataPager control.
- Built-in item selection capabilities
- Multiple key fields
- Programmatic access to the ListView object model to dynamically set properties, handle events and so on
- Fast performance as compared to GridView

3.5.5 Repeater vs. DataList vs. GridView vs. ListView

The DataList control differs from the Repeater control in that the DataList control explicitly places items in an HTML table, whereas the Repeater control does not. The common problem of using a GridView is a large ViewState that could cause slow page loads.

Default GridView paging opens a complete data set in the server's memory. For large tables or for high traffic websites, this will overload the web server's resources. Even the DataPager control of a ListView still opens all the records in memory. Also, pages are opened using JavaScript. That means only the first page is indexed by search engines. The solution could

be to create a custom pager, but this takes time to create, test and optimize code, as well as later maintenance of a separate project. The ListView control can exceed the capabilities of a Repeater or DataList control, but GridView still has the advantage of faster implementation and short markup code.

Now for the second group. Here is the description of the two controls
DetailsView and FormView.

3.5.6 DETAILSVIEW CONTROL

The DetailsView control was introduced with ASP.NET 2.0. The DetailsView control uses a table-based layout where each field of the data record is displayed as a row in the control. Unlike the GridView control, the DetailsView control displays one row from a data source at a time by rendering an HTML table.

The Details View supports both declarative and programmatic data binding. The DetailsView control is often used in master-detail scenarios where the selected record in a master control determines the record to display in the DetailsView control. It shows the details for the row in a separate space. We can customize the appearance of the DetailsView control using its style properties. Alternatively, we can also use Cascading Style Sheets (CSS) to provide styles to a DetailsView control. A Details View control appears as a form of recording and is provided by multiple records as well as insert, update and delete record functions.

The DetailsView control supports the following features

- Tabular rendering
- Supports column layout, by default two columns at a time
- Optional support for paging and navigation.
- Built-in support for data grouping
- Built-in support for edit, insert and delete capabilities

3.5.7 FORMVIEW CONTROL

The FormView was introduced with ASP.NET 2.0. The FormView control renders a single data item at a time from a data source, even if its

data source exposes a multiple records data item from a data source. It allows for a more flexible layout when displaying a single record.

The FormView control renders all fields of a single record in a single table row. In contrast, the FormView control does not specify a pre-defined layout for displaying a record. Instead, create templates that contain controls to display individual fields from the record. The template contains the formatting, controls and binding expressions used to lay out the form.

When using templates, we can place any control such as a dropdown list, checkbox and we can even place tables and rich controls like a GridView and so on. A FormView is a databound control used to insert, display, edit, update and delete data in ASP.NET that renders a single record at a time.

A FormView control is similar to a DetailView in ASP.NET but the only difference is that a DetailsView has a built-in tabular rendering whereas a FormView requires a user-defined template to insert, display, edit, update and delete data.

The FormView control supports the following features

- Template driven
- Supports column layout
- Built-in support for paging and grouping
- Built-in support for insert, edit and delete capabilities

3.5.8 **DETAILSVIEW VS. FORMVIEW CONTROL**

Compared to the DetailsView control, the FormView control gives more flexibility over the rendering of fields. This form of rendering data enables more control over the layout of the fields. Using the FormView control is more complex as compared to the DetailsView control.

The major difference between these controls is that the Details View control displays a single database record as a table based layout. In this layout, data recorded for each field appears as a row in the control and the FormView control uses a template to display a single database record at a time.

3.6 DATA GRID

The **DataGrid** Web server control is a multi-column, data-bound grid. Columns can be made that displays and edit data. Multi-columns include Edit, Update, Cancel, Select buttons, Custom Buttons, and Template Columns. Therefore Template Columns can be laid further in Template-Editing Mode.

The DataGrid control displays the fields of a data source as columns in a table. Each row in the control represents a record in the data source. The control supports selection, editing, deleting, paging, and sorting.

Like the Repeater and DataList controls, it enables to format and display records from a database table. However, it has several advanced features, such as support for sorting and paging through records, which makes it unique.

Records can be displayed in a DataGrid without using templates. A data source can be simply bound to the DataGrid, and it automatically displays the records.

The following example, displays all the records from the Employees database table in a DataGrid.

```

<%@ Import Namespace="System.Data.SqlClient" %>
<Script Runat="Server">
Sub Page_Load
Dim conNorthwind As SqlConnection
Dim cmdSelect As SqlCommand
conNorthwind=New SqlConnection(
"Server=localhost;UID=sa;PWD=secret;Database=Northwind" )
cmdSelect = New SqlCommand( "Select * From Employees", conNorthwind )
conNorthwind.Open()
dgrdEmployees.DataSource = cmdSelect.ExecuteReader()
dgrdEmployees.DataBind()
conNorthwind.Close()
End Sub
</Script>
<html>
<head><title>ExpertDataGrid.aspx</title></head>
<body>
<asp:DataGrid
ID="dgrdEmployees"
Runat="Server" />
</body>
</html>

```

The output of above example is shown below:

EmployeeID	LastName	FirstName	Title	TitleOfCourtesy	BirthDate	HireDate	Address
1	Davolio	Nancy	Sales Representative	Ms.	12/8/1948 12:00:00 AM	5/1/1992 12:00:00 AM	507 - 20th Ave. E. Apt. 2A

By default, a DataGrid displays gridlines around its items. Modification of the Grid line appearance can be done by setting the GridLines property. The possible values are Both, Horizontal, None, or Vertical.

For example, to completely disable GridLines, the DataGrid would look like this: -

```
<asp:DataGrid
  GridLines="None"
  Runat="Server" />
```

The cell spacing and cell padding of the cells in a DataGrid can be controlled by modifying the DataGrid control's CellSpacing and CellPadding properties like this:

```
<asp:DataGrid
  CellSpacing="10"
  CellPadding="10"
  Runat="Server">
```

A background image can be specified for a DataGrid by assigning the name of an image to the BackImageUrl property.

For example, the following DataGrid displays an image named Bricks.Gif in the background:

```
<asp:DataGrid
BackImageUrl="expert.Gif"
Runat="Server" />
```

Finally, headers and footers can be displayed and hidden for the columns in a DataGrid by enabling or disabling the ShowHeader and ShowFooter properties. By default the ShowHeader property has the value True, and the ShowFooter property has the value False. To prevent column headers from being displayed, a DataGrid would be like this:

```
<asp:DataGrid
ShowHeader="False"
Runat="Server" />
```

3.6.1 CREATING COLUMNS IN A DATAGRID CONTROL

The DataGrid control displays the columns in a variety of ways. By default, the columns are generated automatically based on fields in the data source. However, in order to control the content and layout of columns more precisely, the following types of columns can be defined:

Type of Column	Description
Bound column	Allows specifying which data source field to display and specifies the format of that field, using a .NET formatting expression.
Hyperlink column	Displays information as hyperlinks.
Button column	Allows adding a button for each item in the grid and defining custom functionality for that button.

Edit, Update, Cancel column	Allows creating in-place editing. For more details, see "Editing Items" below.
Template column	Allows creating combinations of HTML text and server controls to design a custom layout for a column.

3.6.2 EVENTS

The DataGrid control supports several events. One of them, the ItemCreated event, gives you a way to customize the item-creation process. The ItemDataBound event also gives you the ability to customize the DataGrid items, but after the data is available for inspection. For example, if you were using the DataGrid control to display a to-do list, you could display overdue items in red text, completed items in black text, and other tasks in green text.

The remaining events are raised in response to button or LinkButton clicked in grid items. They are designed to implement common data manipulation tasks. Four events of this type are supported:

- **EditCommand**
- **DeleteCommand**
- **UpdateCommand**
- **CancelCommand**

When the user clicks one of the buttons (labeled by default Edit, Delete, Update, or Cancel, respectively), the corresponding event is raised. The DataGrid control also supports the ItemCommand event that is raised when a user clicks a button that is not one of the predefined buttons above. This event can be used for custom functions by setting a button's CommandName property to a value needed, and then testing for it in the ItemCommand event handler.

3.7 KEY TERMS

Details View: DetailsView control displays a single database record as a table based layout

FormView Control: the FormView control uses a template to display a single database record at a time

Connection Object: **Connection** Object is used for connecting your application to data source or database. It carries required authentic information like username and password in the connection string and opens a connection.

Connection String: Connection String combines all the required authentic information that is used for connecting to a Data Source, like Server Name, Database Name, User Name, Password etc. It is just a single line string that is used by connection object to connect to the database.

OLEDB : OleDbConnection

SQLServer : SqlConnection

ODBC : OdbcConnection

Oracle : OracleConnection

DataSet: DataSet provides a disconnected representation of result sets from the Data Source.

DataProvider: The Data Provider classes are meant to work with different kinds of data sources. They are used to perform all data-management operations on specific databases.

DataReader: The DataReader Object is a stream-based , forward-only, read-only retrieval of query results from the Data Source, which do not update the data.

DataAdapter: DataAdapter Object populate a Dataset Object with results from a Data Source.

3.7.1 Questions: 2 Marks

1. List the events in DataGrid Control.

ItemCreated, ItemDataBound, EditCommand, DeleteCommand, UpdateCommand , CancelCommand

2. Distinguish between DetailsView and FormView Control.

The major difference between these controls is that the DetailsView control displays a single database record as a table based layout. In this layout, data recorded for each field appears as a row in the control and the FormView control uses a template to display a single database record at a time.

3. Differentiate between GridView and ListView Control.

The ListView control can exceed the capabilities of a Repeater or DataList control, but GridView still has the advantage of faster implementation and short markup code.

4. Write the steps for Connecting Database in ADO.Net.

There are 5 steps to connecting database.

1. Add Namespace: using System.Data.SqlClient;
2. Create Connection Object and Pass Connection String as Parameter.
3. Open Connection
4. Execute SQL Query
5. Close the Connection.

5. What is Connection Object?

Connection Object is used for connecting your application to data source or database. It carries required authentic information like username and password in the connection string and opens a connection.

6. What is Connection String?

Connection String combines all the required authentic information that is used for connecting to a Data Source, like Server Name, Database Name, User Name, Password etc. It is just a single line string that is used by connection object to connect to the database.

7. List some Data Connection Objects for different Data Providers.

OLEDB	– OleDbConnection
SQLServer	– SqlConnection
ODBC	– OdbcConnection
Oracle	– OracleConnection

8. Define DataSet.

DataSet provides a disconnected representation of result sets from the Data Source.

9. Define DataProvider.

The Data Provider classes are meant to work with different kinds of data sources. They are used to perform all data-management operations on specific databases.

10. Define DataReader.

The DataReader Object is a stream-based , forward-only, read-only retrieval of query results from the Data Source, which do not update the data.

11. Define DataAdapter.

DataAdapter Object populate a Dataset Object with results from a Data Source .

12. Distinguish between Connected and Disconnected DataBase in ADO.NET

Connected	Disconnected
It is connection oriented.	It is disconnection oriented.
Datareader	DataSet
Connected methods gives faster performance	Disconnected get low in speed and performance.
connected can hold the data of single table	disconnected can hold multiple tables of data
connected you need to use a read only forward only data reader	disconnected you cannot
Data Reader can't persist the data	Data Set can persist the data
It is Read only, we can't update the data.	We can update data

3.7.2 Questions: 5 Marks

1. Explain the events in DataGrid Control.
2. How to create columns in DataGrid Control.
3. List the features of FormView Control.
4. Discuss about Details View Control.
5. Explain in detail about List view Control and GridView Control.
6. Illustrate in detail about Add and Retrieve Connection String.
7. Explain DataProviders And DataSet.

3.7.3 Questions: 10 Marks

1. Explain DataGrid in detail.

2. How to display data on DataBound controls.
3. How to store Connection String in WebConfiguration file?
4. How to create connection in ADO.Net Object Model?
5. Write the steps for creating Application in ADO.NET.
6. Discuss in detail about creation of Database in ADO.NET
7. Explain the architecture of ADO.NET
8. Explain in detail about Connected and Disconnected Database in ADO.NET

UNIT – IV

Database Accessing on Web Applications

4.1 INTRODUCTION

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
- ASP.NET hides the complex processes of data access and provides much higher level of classes and objects through which data is accessed easily.
- These classes hide all complex coding for connection, data retrieving, data querying, and data manipulation.
- The .NET Framework is that it is very easy to encapsulate a database, allowing the rest of the program to work with data in a very generic way, without worrying about where it came from.

4.2 DATA BINDING CONCEPT WITH WEB

- 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 data source controls help the data-bound controls implement functionalities such as, sorting, paging, and editing data collections.

The BaseDataBoundControl is an abstract class, which is inherited by two more abstract classes:

- DataBoundControl
- HierarchicalDataBoundControl

The abstract class DataBoundControl is again inherited by two more abstract classes:

- ListControl
- CompositeDataBoundControl

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:

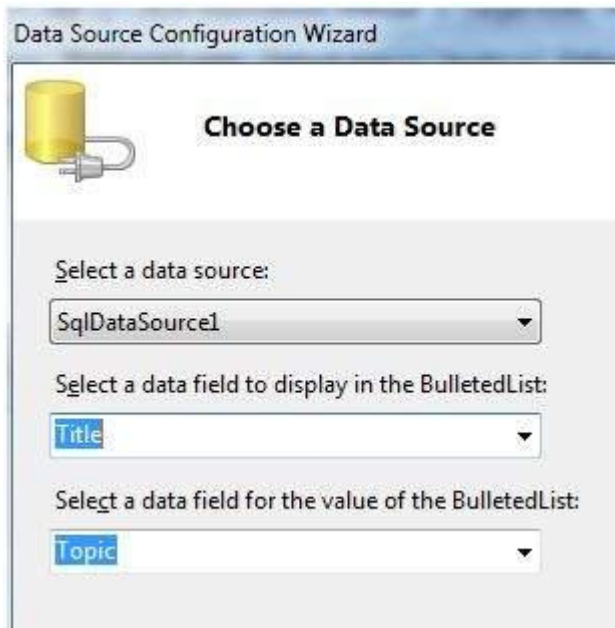
- `DetailsView`
- `FormView`
- `GridView`
- `RecordList`

Simple Data Binding

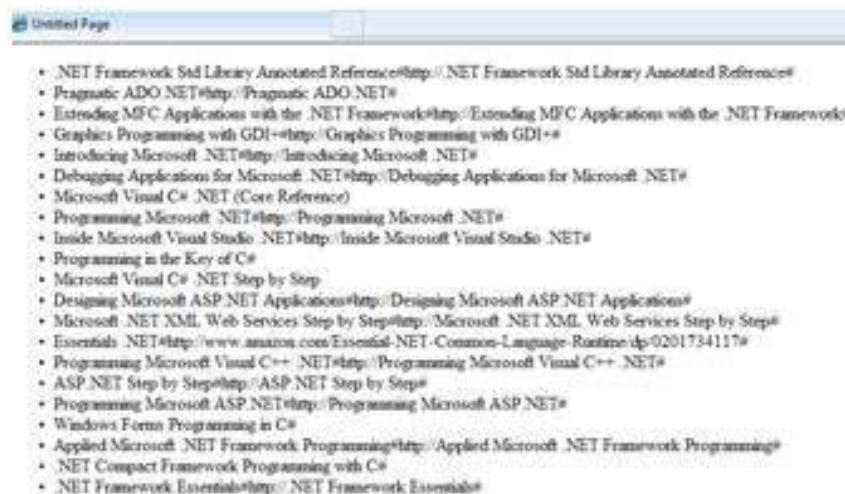
- Simple data binding involves the read-only selection lists. These controls can bind to an array list or fields from a database.
- Selection lists takes two values from the database or the data source; one value is displayed by the list and the other is considered as the value corresponding to the display.
- Let us take up a small example to understand the concept. Create a web site with a bulleted list and a `SqlDataSource` control on it.
- Configure the data source control to retrieve two values from your database

Choosing a data source for the bulleted list control involves:

- Selecting the data source control
- Selecting a field to display, which is called the data field
- Selecting a field for the value



When the application is executed, check that the entire title column is bound to the bulleted list and displayed.



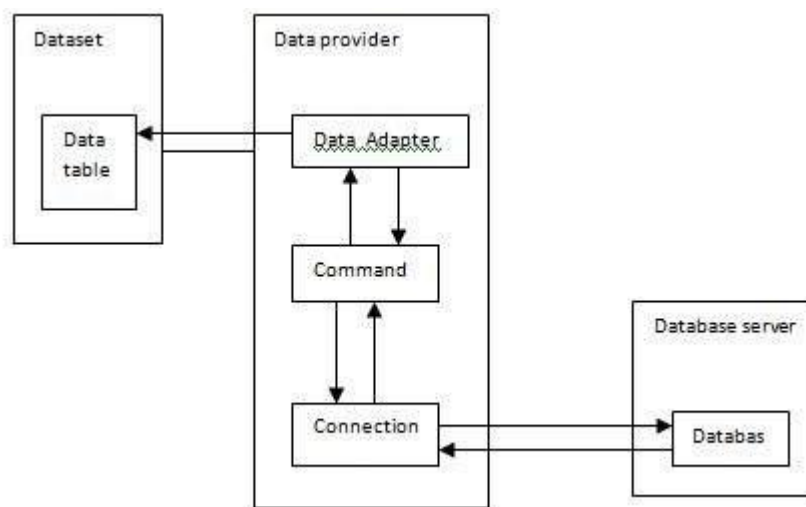
Declarative Data Binding

- The declarative data binding using the GridView control.
- The other composite data bound controls capable of displaying and manipulating data in a tabular manner are the DetailsView, FormView, and RecordList control.

However, the data binding involves the following objects:

- A dataset that stores the data retrieved from the database.
- The data provider, which retrieves data from the database by using a command over a connection.
- The data adapter that issues the select statement stored in the command object; it is also capable of update the data in a database by issuing Insert, Delete, and Update statements.

Relation between the data binding objects:



4.3 Data Grid

- .NET Framework provides DataGrid control to display data on the web page. It was introduced in .NET 1.0.
- DataGrid is used to display data in scrollable grid. It requires data source to populate data in the grid.
- It is a server side control and can be dragged from the toolbox to the web form.
- Data Source for the DataGrid can be either a DataTable or a database.
- Let's see an example, how can we create a DataGrid in our application.

- In this example application One is using the DataTable and second is using the database to display data into the DataGrid.

ASP.NET DataGrid Example with DataTable

This example uses DataTable to bind data to the DataGrid control.

// DataGridExample2.aspx

```

1.  <%@ Page Language="C#" AutoEventWireup="true" CodeBehind="DataGridExample2.aspx.cs" Inherits="DataGridExample.DataGridExample2" %>
2.  <!DOCTYPE html>
3.  <html xmlns="http://www.w3.org/1999/xhtml">
4.  <head runat="server">
5.      <title></title>
6.  </head>
7.  <body>
8.      <form id="form1" runat="server">
9.          <div>
10.             <p>This DataGrid contains DataTable records </p>
11.             <asp:DataGrid ID="DataGrid1" runat="server">
12.                 </asp:DataGrid>
13.             </div>
14.         </form>
15.     </body>
16. </html>

```

CodeBehind

// DataGridExample2.aspx.cs

```

1.  using System;
2.  using System.Data;
3.  namespace DataGridExample
4.  {

```



```

5.      public partial class DataGridExample2 : System.Web.UI.Page
6.      {
7.          protected void Page_Load(object sender, EventArgs e)
8.          {
9.              DataTable table = new DataTable();
10.             table.Columns.Add("ID");
11.             table.Columns.Add("Name");
12.             table.Columns.Add("Email");
13.             table.Rows.Add("101", "Deepak Kumar", "deepak@example.com");

14.             table.Rows.Add("102", "John", "john@example.com");
15.             table.Rows.Add("103", "Subramanium Swami", "subramanium@e
           xample.com");
16.             table.Rows.Add("104", "Abdul Khan", "abdul@example.com");
17.             DataGrid1.DataSource = table;
18.             DataGrid1.DataBind();
19.         }
20.     }
21. }

```

Output:

It produces the following output to the browser.

ID	NAME	EMAIL
101	Deepak Kumar	deepak@example.com
102	Jhon	jhon@example.com
103	Subramanium swami	subramanium@example.com
104	Abdhul Khan	abdhul@example.com

4.4 Binding Standard Web Server Controls

- Controls are small building blocks of the graphical user interface, which include text boxes, buttons, check boxes, list boxes, labels, and numerous other tools.
- Using these tools, the users can enter data, make selections and indicate their preferences.
- Controls are also used for structural jobs, like validation, data access, security, creating master pages, and data manipulation.

ASP.NET uses five types of web controls, which are:

- HTML controls
- HTML Server controls
- ASP.NET Server controls
- ASP.NET Ajax Server controls
- User controls and custom controls

ASP.NET server controls are the primary controls used in ASP.NET. These controls can be grouped into the following categories:

- **Validation controls** - These are used to validate user input and they work by running client-side script.
- **Data source controls** - These controls provides data binding to different data sources.
- **Data view controls** - These are various lists and tables, which can bind to data from data sources for displaying.
- **Personalization controls** - These are used for personalization of a page according to the user preferences, based on user information.
- **Login and security controls** - These controls provide user authentication.
- **Master pages** - These controls provide consistent layout and interface throughout the application.

- **Navigation controls** - These controls help in navigation. For example, menus, tree view etc.
- **Rich controls** - These controls implement special features. For example, AdRotator, FileUpload, and Calendar control.

The syntax for using server controls is:

```
<asp:controlType      ID  ="ControlID"  runat="server"  Property1=value1
[Property2=value2] />
```

In addition, visual studio has the following features, to help produce in error-free coding:

- Dragging and dropping of controls in design view
- IntelliSense feature that displays and auto-completes the properties
- The properties window to set the property values directly

Properties of the Server Controls

- ASP.NET server controls with a visual aspect are derived from the WebControl class and inherit all the properties, events, and methods of this class.
- The WebControl class itself and some other server controls that are not visually rendered are derived from the System.Web.UI.Control class. For example, Placeholder control or XML control.
- ASP.Net server controls inherit all properties, events, and methods of the WebControl and System.Web.UI.Control class.

The following table shows the inherited properties, common to all server controls:

Property	Description
AccessKey	Pressing this key with the Alt key moves focus to the control.
Attributes	It is the collection of arbitrary attributes (for rendering only) that do not correspond to properties on the control.
BackColor	Background color.
BindingContainer	The control that contains this control's data binding.
BorderColor	Border color.
BorderStyle	Border style.
BorderWidth	Border width.
CausesValidation	Indicates if it causes validation.
ChildControlCreated	It indicates whether the server control's child controls have been created.
ClientID	Control ID for HTML markup.
Context	The HttpContext object associated with the server control.
Controls	Collection of all controls contained within the control.
ControlStyle	The style of the Web server control.

CssClass	CSS class
FormItemContainer	Gets a reference to the naming container if the naming container implements IFormItemContainer.
DataKeysContainer	Gets a reference to the naming container if the naming container implements IDataKeysControl.
DesignMode	It indicates whether the control is being used on a design surface.
DisabledCssClass	Gets or sets the CSS class to apply to the rendered HTML element when the control is disabled.
Enabled	Indicates whether the control is grayed out.
EnableTheming	Indicates whether theming applies to the control.
EnableViewState	Indicates whether the view state of the control is maintained.
Events	Gets a list of event handler delegates for the control.
Font	Font.
ForeColor	Foreground color.
HasAttributes	Indicates whether the control has attributes set.
HasChildViewState	Indicates whether the current server control's child controls have any saved view-state settings.
Height	Height in pixels or %.

ID	Identifier for the control.
IsChildControlStateCleared	Indicates whether controls contained within this control have control state.
IsEnabled	Gets a value indicating whether the control is enabled.
IsTrackingViewState	It indicates whether the server control is saving changes to its view state.
IsViewStateEnabled	It indicates whether view state is enabled for this control.
LoadViewStateById	It indicates whether the control participates in loading its view state by ID instead of index.
Page	Page containing the control.
Parent	Parent control.
RenderingCompatibility	It specifies the ASP.NET version that the rendered HTML will be compatible with.
Site	The container that hosts the current control when rendered on a design surface.
SkinID	Gets or sets the skin to apply to the control.
Style	Gets a collection of text attributes that will be rendered as a style attribute on the outer tag of the Web server control.
TabIndex	Gets or sets the tab index of the Web server control.

TagKey	Gets the HtmlTextWriterTag value that corresponds to this Web server control.
TagName	Gets the name of the control tag.
TemplateControl	The template that contains this control.
TemplateSourceDirectory	Gets the virtual directory of the page or control containing this control.
ToolTip	Gets or sets the text displayed when the mouse pointer hovers over the web server control.
UniqueID	Unique identifier.
ViewState	Gets a dictionary of state information that saves and restores the view state of a server control across multiple requests for the same page.
ViewStateIgnoreCase	It indicates whether the StateBag object is case-insensitive.
ViewStateMode	Gets or sets the view-state mode of this control.
Visible	It indicates whether a server control is visible.
Width	Gets or sets the width of the Web server control.

Methods of the Server Controls

The following table provides the methods of the server controls:

Method	Description
AddAttributesToRender	Adds HTML attributes and styles that need to be rendered to the specified HtmlTextWriterTag.
AddedControl	Called after a child control is added to the Controls collection of the control object.
AddParsedSubObject	Notifies the server control that an element, either XML or HTML, was parsed, and adds the element to the server control's control collection.
ApplyStyleSheetSkin	Applies the style properties defined in the page style sheet to the control.
ClearCachedClientID	Infrastructure. Sets the cached ClientID value to null.
ClearChildControlState	Deletes the control-state information for the server control's child controls.
ClearChildState	Deletes the view-state and control-state information for all the server control's child controls.
ClearChildViewState	Deletes the view-state information for all the server control's child controls.
CreateChildControls	Used in creating child controls.
CreateControlCollection	Creates a new ControlCollection object to hold the child controls.

CreateControlStyle	Creates the style object that is used to implement all style related properties.
DataBind	Binds a data source to the server control and all its child controls.
DataBind(Boolean)	Binds a data source to the server control and all its child controls with an option to raise the DataBinding event.
DataBindChildren	Binds a data source to the server control's child controls.
Dispose	Enables a server control to perform final clean up before it is released from memory.
EnsureChildControls	Determines whether the server control contains child controls. If it does not, it creates child controls.
EnsureID	Creates an identifier for controls that do not have an identifier.
Equals(Object)	Determines whether the specified object is equal to the current object.
Finalize	Allows an object to attempt to free resources and perform other cleanup operations before the object is reclaimed by garbage collection.
FindControl(String)	Searches the current naming container for a server control with the specified id parameter.
FindControl(String, Int32)	Searches the current naming container for a server control with the specified id and an

	integer.
Focus	Sets input focus to a control.
GetDesignModeState	Gets design-time data for a control.
GetType	Gets the type of the current instance.
GetUniqueIDRelativeTo	Returns the prefixed portion of the UniqueID property of the specified control.
HasControls	Determines if the server control contains any child controls.
HasEvents	Indicates whether events are registered for the control or any child controls.
IsLiteralContent	Determines if the server control holds only literal content.
LoadControlState	Restores control-state information.
LoadViewState	Restores view-state information.
MapPathSecure	Retrieves the physical path that a virtual path, either absolute or relative, maps to.
MemberwiseClone	Creates a shallow copy of the current object.
MergeStyle	Copies any nonblank elements of the specified style to the web control, but does not overwrite any existing style elements of the control.

OnBubbleEvent	Determines whether the event for the server control is passed up the page's UI server control hierarchy.
OnDataBinding	Raises the data binding event.
OnInit	Raises the Init event.
OnLoad	Raises the Load event.
OnPreRender	Raises the PreRender event.
OnUnload	Raises the Unload event.
OpenFile	Gets a Stream used to read a file.
RemovedControl	Called after a child control is removed from the controls collection of the control object.
Render	Renders the control to the specified HTML writer.
RenderBeginTag	Renders the HTML opening tag of the control to the specified writer.
RenderChildren	Outputs the contents of a server control's children to a provided HtmlTextWriter object, which writes the contents to be rendered on the client.
RenderContents	Renders the contents of the control to the specified writer.

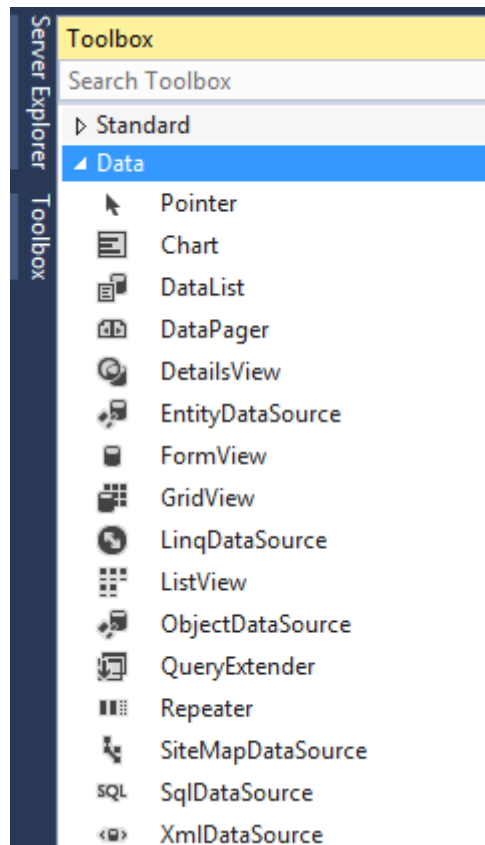
RenderControl(HtmlTextWriter)	Outputs server control content to a provided HtmlTextWriter object and stores tracing information about the control if tracing is enabled.
RenderEndTag	Renders the HTML closing tag of the control into the specified writer.
ResolveAdapter	Gets the control adapter responsible for rendering the specified control.
SaveControlState	Saves any server control state changes that have occurred since the time the page was posted back to the server.
SaveViewState	Saves any state that was modified after the TrackViewState method was invoked.
SetDesignModeState	Sets design-time data for a control.
ToString	Returns a string that represents the current object.
TrackViewState	Causes the control to track changes to its view state so that they can be stored in the object's view state property.

4.5 Display Data on Web Form using Data Bound Controls

Data Bound Controls

- ASP.NET provides a wide variety of rich controls that can be bound to data.

- Under the Data tab of the Visual Studio Toolbox, you can get several controls under the Data tab that could be used to display data from a data source, like a database or XML file.



The standard ASP.NET data presentation controls are:

- DataList
- DetailsView
- FormView
- GridView
- ListView
- Repeater

Data List Control

The DataList control was introduced with ASP.NET 1.0.

- DataList is the next step up from a Repeater; except you have very little control over the HTML that the control renders.
- DataList allows you to repeat columns horizontally or vertically.
- The DataList control renders data as a table and enables you to display data records in various layouts, such as ordering them in columns or rows.
- You can configure the DataList control to enable users to edit or delete a record in the table. We can use a DataList control where we need a single-column list.
- The DataList control works like the Repeater control, used to display the data in a repeating structure, such as a table.
- It displays data in a format that you can define using a template and styles. However, it arranges the data defined in the template within various HTML structures.
- This includes options for horizontal or vertical layout and it also allows you to set how the data should be repeated, as flow or table layout.
- The DataList control does not automatically use a data source control to edit data. Instead, it provides command events in which you can write your own code for these scenarios.
- You can configure the DataList control where the user can edit or delete a record in the table. The DataList control supports the following features:
 - Support for binding data source controls such as SqlDataSource, LinqDataSource and ObjectDataSource
 - Directional rendering
 - Good for columns
 - Item as cell

- Updatable
- Control over Alternate item
- Paging function needs handwriting.

After execution our ListView will look like this.

Student Details		
Student Id : 1 Name : Mark Last Name : Arton Email : markarton03@outlook.com	Student Id : 2 Name : Tony Last Name : Jacob Email : tonyjacob@yahoo.com	Student Id : 3 Name : Robert Last Name : Brown Email : robertbrown@yahoo.com
Student Id : 4 Name : Lucas Last Name : Biglia Email : lucasbiglia@gmail.com.com		

Details View control

The DetailsView control was introduced with ASP.NET 2.0.

- The DetailsView control uses a table-based layout where each field of the data record is displayed as a row in the control.
- Unlike the GridView control, the DetailsView control displays one row from a data source at a time by rendering an HTML table.
- The DetailsView supports both declarative and programmatic data binding.
- The DetailsView control is often used in master-detail scenarios where the selected record in a master control determines the record to display in the DetailsView control.
- It shows the details for the row in a separate space. We can customize the appearance of the DetailsView control using its style properties.

- Alternatively, we can also use Cascading Style Sheets (CSS) to provide styles to a DetailsView control.
- A Details View control appears as a form of recording and is provided by multiple records as well as insert, update and delete record functions.

The DetailsView control supports the following features:

- Tabular rendering
- Supports column layout, by default two columns at a time
- Optional support for paging and navigation.
- Built-in support for data grouping
- Built-in support for edit, insert and delete capabilities

Id	1
First Name	Mark
Last Name	Arton
Email	markarton03@outlook.com
Delete	
New	
Edit	
1 2 3 4	

FormView control

The FormView was introduced with ASP.NET 2.0.

- The FormView control renders a single data item at a time from a data source, even if its data source exposes a multiple records data item from a data source.
- It allows for a more flexible layout when displaying a single record.
- The FormView control renders all fields of a single record in a single table row.

- In contrast, the FormView control does not specify a pre-defined layout for displaying a record.
- Instead, you create templates that contain controls to display individual fields from the record.
- The template contains the formatting, controls and binding expressions used to lay out the form.
- When using templates, we can place any control such as a dropdown list, checkbox and we can even place tables and rich controls like a GridView and so on.
- A FormView is a databound control used to insert, display, edit, update and delete data in ASP.NET that renders a single record at a time.
- A FormView control is similar to a DetailView in ASP.NET but the only difference is that a DetailsView has a built-in tabular rendering whereas a FormView requires a user-defined template to insert, display, edit, update and delete data.
- The FormView control supports the following features:
 - Template driven
 - Supports column layout
 - Built-in support for paging and grouping
 - Built-in support for insert, edit and delete capabilities

Grid View Control

ASP.NET provides a number of tools for showing tabular data in a grid, including the GridView control. It was introduced with ASP.NET 2.0.

- The GridView control is used to display the values of a data source in a table.

- Each column represents a field where each row represents a record. It can also display empty data.
- The GridView control provides many built-in capabilities that allow the user to sort, update, delete, select and page through items in the control.
- The GridView control can be bound to a data source control, in order to bind a data source control, set the DataSourceID property of the GridView control to the ID value of the data source control.
- It's considered a replacement for the DataGrid control from .NET 1.1. Therefore, it is also known as a super DataGrid.
- The GridView control offers improvements such as the ability to define multiple primary key fields, improved user interface customization using bound fields and templates and a new model for handling or canceling events.

Performance is slow compared to DataGrid and ListView.

The GridView control supports the following features:

- Improved data source binding capabilities
- Tabular rendering – displays data as a table
- Item as row
- Built-in sorting capability
- Built-in select, edit and delete capabilities
- Built-in paging capability
- Built-in row selection capability
- Multiple key fields
- Programmatic access to the GridView object model to dynamically set properties, handle events and so on
- Richer design-time capabilities
- Control over Alternate item, Header, Footer, Colors, font, borders, and so on.

- Slow performance as compared to Repeater and DataList control

Action	<u>Id</u>	<u>Name</u>	<u>LastName</u>	<u>Email</u>
Edit Delete Select	1	Mark	Arton	markarton03@outlook.com
Edit Delete Select	2	Tony	Jacob	tonyjacob@yahoo.com
Edit Delete Select	3	Robert	Brown	robertbrown@yahoo.com
Edit Delete Select	4	Lucas	Biglia	lucasbiglia@gmail.com

List View Control

The ListView control was introduced with ASP.NET 3.5.

- The ListView control resembles the GridView control. The only difference between them is that the ListView control displays data using user-defined templates instead of row fields.
- Creating your own templates gives you more flexibility in controlling how the data is displayed.
- It enables you to bind to data items that are returned from a data source and display them.
- The data can be displayed in pages where you can display items individually, or you can group them.
- The template contains the formatting, controls and binding expressions that are used to lay out the data.
- The ListView control is useful for data in any repeating structure, similar to the DataList and Repeater controls.
- It implicitly supports the ability to edit, insert and delete operations, as well as sorting and paging functionality. You can define individual templates for each of these scenarios.

The ListView control supports the following features:

- Binding to data source controls Customizable appearance through user-defined templates and styles.
- Built-in sorting and grouping capabilities
- Built-in insert, edit and delete capabilities
- Support for paging capabilities using a DataPager control.
- Built-in item selection capabilities
- Multiple key fields
- Programmatic access to the ListView object model to dynamically set properties, handle events and so on
- Fast performance as compared to GridView

Id: 1 Name: Mark LastName: Arton Email: markarton03@outlook.com <input type="button" value="Delete"/> <input type="button" value="Edit"/>	Id: 2 Name: Tony LastName: Jacob Email: tonyjacob@yahoo.com <input type="button" value="Delete"/> <input type="button" value="Edit"/>	Id: 3 Name: Robert LastName: Brown Email: robertbrown@yahoo.com <input type="button" value="Delete"/> <input type="button" value="Edit"/>
Id: 4 Name: Lucas LastName: Biglia Email: lucasbiglia@gmail.com <input type="button" value="Delete"/> <input type="button" value="Edit"/>	Name: <input type="text"/> LastName: <input type="text"/> Email: <input type="text"/> <input type="button" value="Insert"/> <input type="button" value="Clear"/>	
<input type="button" value="First"/> <input type="button" value="Previous"/> <input type="button" value="Next"/> <input type="button" value="Last"/>		

Repeater Control

The Repeater control was introduced with ASP.NET 1.0. The ASP.NET Repeater is a basic container control that allows you to create custom lists from any data available to the page.

- It provides a highly customized interface. It renders a read-only template; in other words, it supports only the ItemTemplate to define custom binding.
- The Repeater control is a Data Bind Control, also known as container controls.

- The Repeater control is used to display a repeated list of items that are bound to the control.
- This control may be bound to a database table, an XML file, or another list of items.
- It has no built-in layout or styles, so you must explicitly declare all layout, formatting and style tags within the controls templates.
- You would require writing an explicit code to do paging using this control.
- The Repeater repeats a layout of HTML you write, it has the least functionality of the rest of the three controls.

The Repeater control supports the following features:

- List format
- No default output
- More control and complexity
- Item as row
- Paging, Sorting and Grouping requires custom code writing
- only Web control that allows you to split markup tags across the templates
- no built-in selection capabilities
- no built-in support for edit, insert and delete capabilities
- no built-in support for paging, sorting and grouping capabilities
- no built-in layout or styles, need to declare all layout, formatting and style tags explicitly within the control's templates
- Strictly emits the markup specified in its templates, nothing more and nothing less.

Using the data we stored in table Student of our sample database, the Repeater control will look like this.

Id: 1 Name: Mark LastName: Arton Email: markarton03@outlook.com
Id: 2 Name: Tony LastName: Jacob Email: tonyjacob@yahoo..com
Id: 3 Name: Robert LastName: Brown Email: robertbrown@yahoo.com

4.6.1 2 MARK QUESTIONS

1. What is ASP.Net?

It is a framework developed by Microsoft on which we can develop new generation web sites using web forms(aspx), MVC, HTML, Javascript, CSS etc. Its successor of Microsoft Active Server Pages(ASP). Currently there is ASP.NET 4.0, which is used to develop web sites. There are various page extensions provided by Microsoft that are being used for web site development. Eg: aspx, asmx, ascx, ashx, cs, vb, html, XML etc.

2. What's the use of Response.Output.Write ()?

We can write formatted output using Response.Output.Write ().

3. In which event of page cycle is the ViewState available?

After the Init() and before the Page_Load().

4. What is the difference between Server.Transfer and Response.Redirect?

In Server.Transfer page processing transfers from one page to the other page without making a round-trip back to the client's browser. This provides a faster response with a little less overhead on the server. The

clients url history list or current url Server does not update in case of Server.Transfer.

Response.Redirect is used to redirect the user's browser to another page or site. It performs trip back to the client where the client's browser is redirected to the new page. The user's browser history list is updated to reflect the new address.

5. From which base class all Web Forms are inherited?

Page class.

6. What are the different validators in ASP.NET?

Required field Validator

Range Validator

Compare Validator

Custom Validator

Regular expression Validator

Summary Validator

7. Which validator control you use if you need to make sure the values in two different controls matched?

Compare Validator control.

8. What is ViewState?

ViewState is used to retain the state of server-side objects between page post backs.

9. Where the viewstate is stored after the page postback?

ViewState is stored in a hidden field on the page at client side. ViewState is transported to the client and back to the server, and is not stored on the server or any other external source.

10. How long the items in ViewState exists?

They exist for the life of the current page.

11. What are the different Session state management options available in ASP.NET?

In-Process

Out-of-Process.

In-Process stores the session in memory on the web server.

Out-of-Process Session state management stores data in an external server. The external server may be either a SQL Server or a State Server. All objects stored in session are required to be serializable for Out-of-Process state management.

12. How you can add an event handler?

Using the Attributes property of server side control.

e.g. `btnSubmit.Attributes.Add("onMouseOver","JavascriptCode();")`

13. What is caching?

Caching is a technique used to increase performance by keeping frequently accessed data or files in memory. The request for a cached file/data will be accessed from cache instead of actual location of that file.

14. What are the different types of caching?ASP.NET has 3 kinds of caching

Output Caching,

Fragment Caching,

Data Caching.

15. Which type of caching will be used if we want to cache the portion of a page instead of whole page?

Fragment Caching: It caches the portion of the page generated by the request. For that, we can create user controls with the below code:


```
<%@ OutputCache Duration="120"
VaryByParam="CategoryID;SelectedID"%>
```

16. List the events in page life cycle.

- 1) Page_PreInit
- 2) Page_Init
- 3) Page_InitComplete
- 4) Page_PreLoad
- 5) Page_Load
- 6) Page_LoadComplete
- 7) Page_PreRender
- 8) Render

17. Can we have a web application running without web.Config file?

Yes

18. Is it possible to create web application with both webforms and mvc?

Yes. We have to include below mvc assembly references in the web forms application to create hybrid application.

System.Web.Mvc

System.Web.Razor

System.ComponentModel.DataAnnotations

19. Can we add code files of different languages in App_Code folder?

No. The code files must be in same language to be kept in App_code folder.

20. What is Protected Configuration?

It is a feature used to secure connection string information.

4.6.2 5 MARK QUESTIONS

1. What are the basics of data binding concept? Summarize.
2. Draw a neat diagram of data grid and explain in detail.
3. Explain about the Data Bound Controls.

4. Write about Web Server Controls. Explain.
5. Briefly notes on Database Accessing.

4.6.3 10 MARK QUESTIONS

1. Explain in detail about the Database Accessing in Web applications.
2. Explain about data bound controls in detail.
3. Describe the advantages and disadvantages of Data Grid.
4. Explain Binding standard web server controls in detail

UNIT -V

XML

5.1 WRITING DATASET CONTENTS AS XML DATA

In ASP.NET you can write an XML representation of a DataSet, with or without its schema. If schema information is included inline with the XML, it is written using the XML Schema definition language (XSD).

- The schema contains the table definitions of the DataSet as well as the relation and constraint definitions.
- When a DataSet is written as XML data, the rows in the DataSet are written in their current versions. The DataSet can also be written as a DiffGram so that both the current and the original values of the rows will be included.
- The XML representation of the DataSet can be written to a file, a stream, an **XmlWriter**, or a string. These choices provide great flexibility for how you transport the XML representation of the DataSet. To obtain the XML representation of the DataSet as a string, use the **GetXml** method as shown in the following

Example

```
string xmlDS = custDS.GetXml();
```

- **GetXml** returns the XML representation of the DataSet without schema information. To write the schema information from the DataSet (as XML Schema) to a string, use **GetXmlSchema**.

- To write a [DataSet](#) to a file, stream, or **XmlWriter**, use the **WriteXml** method. The first parameter you pass to **WriteXml** is the destination of the XML output.
- For example, pass a string containing a file name, a **System.IO.TextWriter** object, and so on. You can pass an optional second parameter of an **XmlWriteMode** to specify how the XML output is to be written.

The following table shows the options for **XmlWriteMode**.

XmlWriteMode option	DESCRIPTION
IgnoreSchema	Writes the current contents of the DataSet as XML data, without an XML Schema. This is the default.
WriteSchema	Writes the current contents of the DataSet as XML data with the relational structure as inline XML Schema.
DiffGram	Writes the entire DataSet as a DiffGram, including original and current values. For more information, see DiffGrams .

- When writing an XML representation of a [DataSet](#) that contains **DataRelation** objects, you will most likely want the resulting XML to have the child rows of each relation nested within their related parent elements.
- To accomplish this, set the **Nested** property of the **DataRelation** to **true** when you add the **DataRelation** to the [DataSet](#). For more information, see [Nesting DataRelations](#).
- The following are two examples of how to write the XML representation of a [DataSet](#) to a file. The first example passes the file name for the

resulting XML as a string to **WriteXml**. The second example passes a **System.IO.StreamWriter** object.

Program

```
custDS.WriteXml("Customers.xml", XmlWriteMode.WriteSchema);
```

```
C#Copy
```

```
System.IO.StreamWriter xmlSW = new
```

```
System.IO.StreamWriter("Customers.xml");
```

```
custDS.WriteXml(xmlSW, XmlWriteMode.WriteSchema);
```

```
xmlSW.Close();
```

MAPPING COLUMNS TO XML ELEMENTS, ATTRIBUTES, AND TEXT

You can specify how a column of a table is represented in XML using the **ColumnMapping** property of the **DataColumn** object. The following table shows the different **MappingType** values for the **ColumnMapping** property of a table column, and the resulting XML.

MappingType value	Description
Element	This is the default. The column is written as an XML element where the ColumnName is the name of the element and the contents of the column are written as the text of the element. For example: <ColumnName>Column Contents</ColumnName>
Attribute	The column is written as an XML attribute of the XML element for the current row where the ColumnName is the name of the attribute and the contents of the column are written as the value of the attribute. For example: <RowElement ColumnName="Column Contents" />

MappingType value	Description
SimpleContent	The contents of the column are written as text in the XML element for the current row. For example: <RowElement>Column Contents</RowElement>
Hidden	The column is not written in the XML output.

Create one empty data dataset with database name, so that this name will reflect to your converted xml document.

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

```
ds.DataSetName = "ds";
```

Add your data (data tables) to this dataset with name

```
data.TableName = "tb";
```

```
ds.Tables.Add(data);
```

Create one XmlDocument & load data string into it

```
using (MemoryStream memoryStream = new MemoryStream())
{
    using (TextWriter streamWriter = new StreamWriter(memoryStream))
    {
        XmlSerializer xmlSerializer = new XmlSerializer(typeof(DataSet));
        xmlSerializer.Serialize(streamWriter, ds);
        result = Encoding.UTF8.GetString(memoryStream.ToArray());
    }
}
```

```
XmlDocument _doc = new XmlDocument();
```

```
_doc.LoadXml(result);
```

5.2 READ XML DATA INTO A DATASET

- ASP.NET provides simple methods for working with XML data. In this walkthrough, you create a Windows application that loads XML data into a dataset.
- The dataset is then displayed in a [DataGridView](#) control. Finally, an XML schema based on the contents of the XML file is displayed in a text box.

Create a new project

- Create a new **Windows Forms App** project for either C# or Visual Basic. Name the project **ReadingXML**.
- Generate the XML file to be read into the dataset
- Because this walkthrough focuses on reading XML data into a dataset, the contents of an XML file is provided.
- On the **Project** menu, select **Add New Item**.
- Select **XML File**, name the file **authors.xml**, and then select **Add**.
- The XML file loads into the designer and is ready for edit.
- Paste the following XML data into the editor below the XML declaration:

XMLCopy

```
<Authors_Table>
```

```
<authors>
```

```
<au_id>172-32-1176</au_id>
```

```
<au_lname>White</au_lname>
```

```
<au_fname>Johnson</au_fname>
```

```
<phone>408 496-7223</phone>
```

```
<address>10932 Bigge Rd.</address>
```

```
<city>Menlo Park</city>
```

```
<state>CA</state>

<zip>94025</zip>

<contract>true</contract>

</authors>

<authors>

  <au_id>213-46-8915</au_id>

  <au_lname>Green</au_lname>

  <au_fname>Margie</au_fname>

  <phone>415 986-7020</phone>

  <address>309 63rd St. #411</address>

  <city>Oakland</city>

  <state>CA</state>

  <zip>94618</zip>

  <contract>true</contract>

</authors>

<authors>

  <au_id>238-95-7766</au_id>

  <au_lname>Carson</au_lname>

  <au_fname>Cheryl</au_fname>

  <phone>415 548-7723</phone>

  <address>589 Darwin Ln.</address>

  <city>Berkeley</city>

  <state>CA</state>

  <zip>94705</zip>
```

```
<contract>true</contract>

</authors>

<authors>

  <au_id>267-41-2394</au_id>

  <au_lname>Hunter</au_lname>

  <au_fname>Anne</au_fname>

  <phone>408 286-2428</phone>

  <address>22 Cleveland Av. #14</address>

  <city>San Jose</city>

  <state>CA</state>

  <zip>95128</zip>

  <contract>true</contract>

</authors>

<authors>

  <au_id>274-80-9391</au_id>

  <au_lname>Straight</au_lname>

  <au_fname>Dean</au_fname>

  <phone>415 834-2919</phone>

  <address>5420 College Av.</address>

  <city>Oakland</city>

  <state>CA</state>

  <zip>94609</zip>

  <contract>true</contract>

</authors>
```


</Authors_Table>

On the **File** menu, select **Save authors.xml**.

Create the user interface:

- The user interface for this application consists of the following:
- A [DataGridView](#) control that displays the contents of the XML file as data.
- A [TextBox](#) control that displays the XML schema for the XML file.
- Two [Button](#) controls.
- One button reads the XML file into the dataset and displays it in the [DataGridView](#) control.
- A second button extracts the schema from the dataset, and through a [StringWriter](#) displays it in the [TextBox](#) control.
- To add controls to the form
- Open Form1 in design view.
- From the **Toolbox**, drag the following controls onto the form:
- One [DataGridView](#) control
- One [TextBox](#) control
- Two [Button](#) controls

Set the following properties:

Control	Property	Setting
TextBox1	Multiline	True
	ScrollBars	Vertical
Button1	Name	ReadXmlButton
	Text	Read XML

Control	Property	Setting
Button2	Name	ShowSchemaButton
	Text	Show Schema

Create the dataset that receives the XML data

- In this step, you create a new dataset named authors. For more information about datasets.
- In **Solution Explorer**, select the source file for **Form1**, and then select the **View Designer** button on the **Solution Explorer** toolbar.
- From the [Toolbox, Data tab](#), drag a **DataSet** onto **Form1**.
- In the **Add Dataset** dialog box, select **Untyped dataset**, and then select **OK**.
- **DataSet1** is added to the component tray.
- In the **Properties** window, set the **Name** and [DataSetName](#) properties forAuthorsDataSet.
- Create the event handler to read the XML file into the dataset
- The **Read XML** button reads the XML file into the dataset. It then sets properties on the [DataGridView](#) control that bind it to the dataset.
- In **Solution Explorer**, select **Form1**, and then select the **View Designer** button on the **Solution Explorer** toolbar.
- Select the **Read XML** button.
- The **Code Editor** opens at the ReadXmlButton_Click event handler.
- Type the following code into the ReadXmlButton_Click event handler:

EXAMPLE:

```
private void ReadXmlButton_Click(object sender, EventArgs e)
{
    string filePath = "Complete path where you saved the XML file";
```

```

AuthorsDataSet.ReadXml(filePath);
dataGridView1.DataSource = AuthorsDataSet;
dataGridView1.DataMember = "authors";
}

```

In the ReadXMLButton_Click event handler code, change the filepath = entry to the correct path. Create the event handler to display the schema in the textbox. The **Show Schema** button creates a [StringWriter](#) object that's filled with the schema and is displayed in the [TextBox](#) control.

In **Solution Explorer**, select **Form1**, and then select the **View Designer** button. Select the **Show Schema** button. The **Code Editor** opens at the ShowSchemaButton_Click event handler. Paste the following code into the ShowSchemaButton_Click event handler.

C#Copy

```

private void ShowSchemaButton_Click(object sender, EventArgs e)
{
    System.IO.StringWriter swXML = new System.IO.StringWriter();
    AuthorsDataSet.WriteXmlSchema(swXML);
    textBox1.Text = swXML.ToString();
}

```

Test the Form:

- ❖ You can now test the form to make sure it behaves as expected.
- ❖ Select **F5** to run the application.
- ❖ Select the **Read XML** button.
- ❖ The DataGridView displays the contents of the XML file.
- ❖ Select the **Show Schema** button.
- ❖ The text box displays the XML schema for the XML file.

Next steps

This walkthrough teaches you the basics of reading an XML file into a dataset, as well as creating a schema based on the contents of the XML file. Here are some tasks that you might do next: Edit the data in the dataset and

write it back out as XML. For more information, see [WriteXml](#). Edit the data in the dataset and write it out to a database.

5.3 REMOTE METHOD CALL USING XML SOAP

What is SOAP?

- ❖ SOAP is an XML-based protocol for accessing web services over HTTP. It has some specification which could be used across all applications.
- ❖ SOAP is known as the Simple Object Access Protocol, but in later times was just shortened to SOAP v1.2. SOAP is a protocol or in other words is a definition of how web services talk to each other or talk to client applications that invoke them.
- ❖ SOAP was developed as an intermediate language so that applications built on various programming languages could talk easily to each other and avoid the extreme development effort.
- To Exchanging data between applications is crucial in today's networked world. But data exchange between these heterogeneous applications would be complex. So will be the complexity of the code to accomplish this data exchange.
- One of the methods used to combat this complexity is to use XML (Extensible Markup Language) as the intermediate language for exchanging data between applications.
- Every programming language can understand the XML markup language. Hence, XML was used as the underlying medium for data exchange.
- But there are no standard specifications on use of XML across all programming languages for data exchange. That is where SOAP comes in.
- SOAP was designed to work with XML over HTTP and have some sort of specification which could be used across all applications. We will look into further details on the SOAP protocol in the subsequent chapters.

5.3.1 ADVANTAGES OF SOAP:

- ❖ SOAP is the protocol used for data interchange between applications. Below are some of the reasons as to why SOAP is used.
- ❖ When developing Web services, you need to have some of language which can be used for web services to talk with client applications. SOAP is the perfect medium which was developed in order to achieve this purpose.
- ❖ This protocol is also recommended by the W3C consortium which is the governing body for all web standards.
- ❖ SOAP is a light-weight protocol that is used for data interchange between applications. Note the keyword '**light**.'
- ❖ Since SOAP is based on the XML language, which itself is a light weight data interchange language, hence SOAP as a protocol that also falls in the same category.
- ❖ SOAP is designed to be platform independent and is also designed to be operating system independent. So the SOAP protocol can work any programming language based applications on both Windows and [Linux](#) platform.
- ❖ It works on the HTTP protocol –SOAP works on the HTTP protocol, which is the default protocol used by all web applications. Hence, there is no sort of customization which is required to run the web services built on the SOAP protocol to work on the World Wide Web.

5.3.2 SOAP Building blocks

The SOAP specification defines something known as a "**SOAP message**" which is what is sent to the web service and the client application.

The diagram below shows the various building blocks of a SOAP Message.

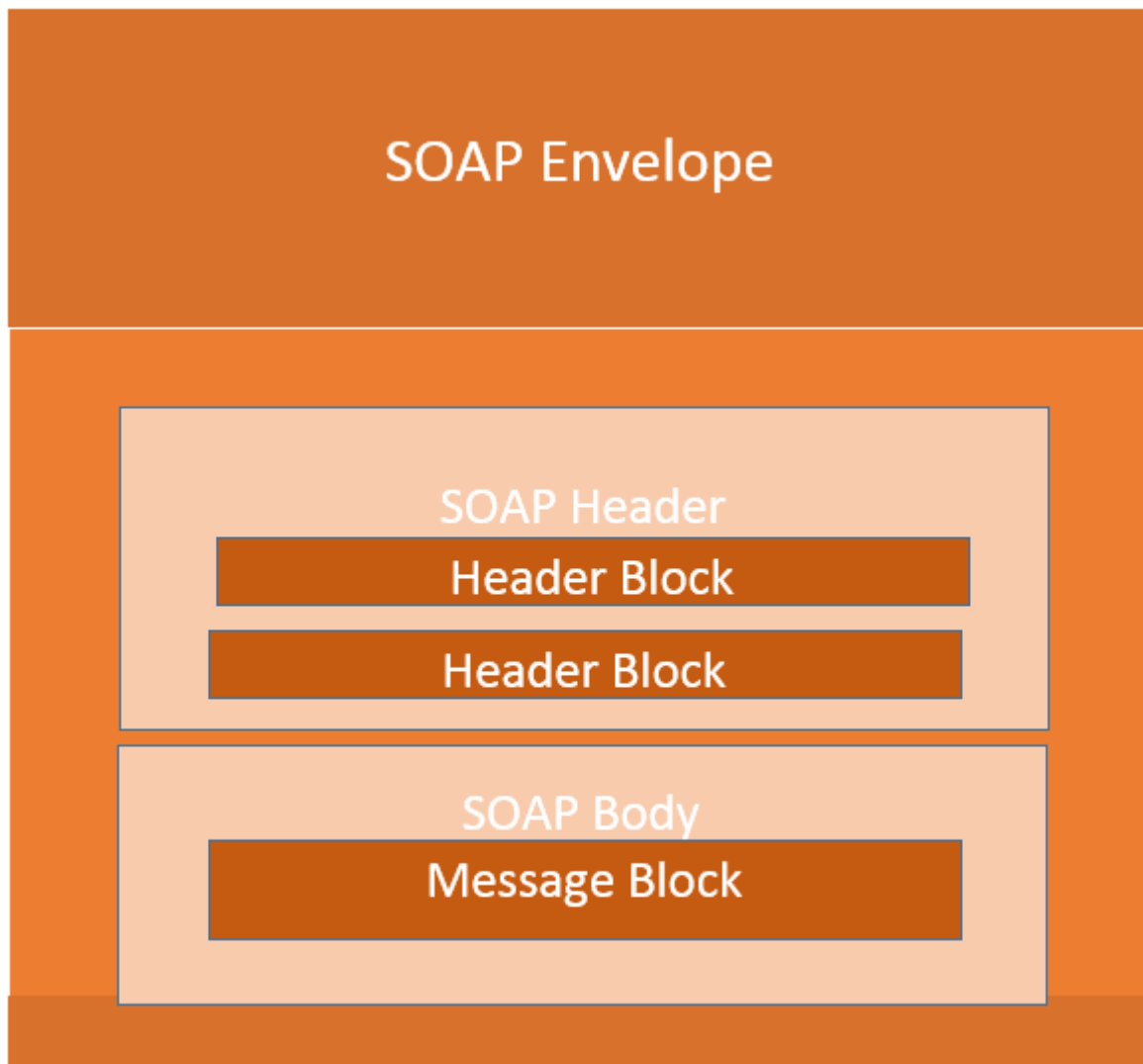


FIG 5.3 SOAP BUILDING BLOCK

- The SOAP message is nothing but a mere XML document which has the below components. An Envelope element that identifies the XML document as a SOAP message – This is the containing part of the SOAP message and is used to encapsulate all the details in the SOAP message. This is the root element in the SOAP message.
- A Header element that contains header information – The header element can contain information such as authentication credentials which can be used by the calling application. It can also contain the definition of complex types which could be used in the SOAP message.
- By default, the SOAP message can contain parameters which could be of simple types such as strings and numbers, but can also be a

complex object type. A simple example of a complex type is shown below.

- Suppose we wanted to send a structured data type which had a combination of a "Tutorial Name" and a "Tutorial Description," then we would define the complex type as shown below.
- The complex type is defined by the element tag <xsd:complexType>. All of the required elements of the structure along with their respective data types are then defined in the complex type collection.

Example:

```
<xsd:complexType>
```

```
<xsd:sequence>
```

```
<xsd:element name="Tutorial Name" type="string"/>
```

```
<xsd:element name="Tutorial Description" type="string"/>
```

```
</xsd:sequence>
```

```
</xsd:complexType>
```

- A Body element that contains call and response information – This element is what contains the actual data which needs to be sent between the web service and the calling application.
- Below is an example of the SOAP body which actually works on the complex type defined in the header section.
- Here is the response of the Tutorial Name and Tutorial Description that is sent to the calling application which calls this web service.

```
<soap:Body>
```

```
<GetTutorialInfo>
```

```
<TutorialName>Web Services</TutorialName>
```

```
<TutorialDescription>All about web  
services</TutorialDescription>
```

```
</GetTutorialInfo>
```

</soap:Body>

5.3.4 SOAP Message Structure

SOAP messages are normally auto-generated by the web service when it is called. Whenever a client application calls a method in the web service, the web service will automatically generate a SOAP message which will have the necessary details of the data which will be sent from the web service to the client application.

5.4 WEB SERVICE DESCRIPTION LANGUAGE:

WSDL breaks down web services into three specific, identifiable elements that can be combined or reused once defined. The three major elements of WSDL that can be defined separately are:

- Types
- Operations
- Binding

A WSDL document has various elements, but they are contained within these three main elements, which can be developed as separate documents and then they can be combined or reused to form complete WSDL files.

Definition:

It is the root element of all WSDL documents. It defines the name of the web service, declares multiple namespaces used throughout the remainder of the document, and contains all the service elements described here.

Data types: The data types to be used in the messages are in the form of XML schemas.

Message: It is an abstract definition of the data, in the form of a message presented either as an entire document or as arguments to be mapped to a method invocation.

Operation: It is the abstract definition of the operation for a message, such as naming a method, message queue, or business process, that will accept and process the message.

Port type: It is an abstract set of operations mapped to one or more end-points, defining the collection of operations for a binding. The collection of operations, as it is abstract, can be mapped to multiple transports through various bindings.

Binding: It is the concrete protocol and data formats for the operations and messages defined for a particular port type.

Port: It is a combination of a binding and a network address, providing the target address of the service communication.

Service: It is a collection of related end-points encompassing the service definitions in the file. The services map the binding to the port and include any extensibility definitions. In addition to these major elements, the WSDL specification also defines the following utility elements:

Documentation: This element is used to provide human-readable documentation and can be included inside any other WSDL element.

Import: This element is used to import other WSDL documents or XML Schemas. WSL parts are usually generated automatically using web services-aware tools.

The WSDL Document Structure The main structure of a WSDL document looks like this:

```
<definitions>
```

```
<types>
```

```
    definition of types.....
```

```
</types>
```

```
<message>
```

```
    definition of a message....
```

```
</message>
```

```
<portType>
```

<operation>

definition of a operation.....

</operation>

</portType>

<binding>

definition of a binding....

</binding>

<service>

definition of a service....

</service>

</definitions>

A WSDL document can also contain other elements, like extension elements and a service element that makes it possible to group together the definitions of several web services in one single WSDL document. Proceed further to analyze an example of WSDL Document.

5.4.1 FEATURES OF WSDL :

- ❖ WSDL is an XML-based protocol for information exchange in decentralized and distributed environments.
- ❖ WSDL definitions describe how to access a web service and what operations it will perform.
- ❖ WSDL is a language for describing how to interface with XML-based services.
- ❖ WSDL is an integral part of Universal Description, Discovery, and Integration (UDDI), an XML-based worldwide business registry.
- ❖ WSDL is the language that UDDI uses.
- ❖ WSDL is pronounced as 'wiz-dull' and spelled out as 'W-S-D-L'

Example Contents of HelloService.wsdl file:

```

<definitions name="HelloService"

  targetNamespace="http://www.examples.com/wsdl/HelloService.wsdl"

  xmlns="http://schemas.xmlsoap.org/wsdl/"

  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"

  xmlns:tns="http://www.examples.com/wsdl/HelloService.wsdl"

  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <message name="SayHelloRequest">

    <part name="firstName" type="xsd:string"/>

  </message>

  <message name="SayHelloResponse">

    <part name="greeting" type="xsd:string"/>

  </message>

  <portType name="Hello_PortType">

    <operation name="sayHello">

      <input message="tns:SayHelloRequest"/>

      <output message="tns:SayHelloResponse"/>

    </operation>

  </portType>

  <binding name="Hello_Binding" type="tns:Hello_PortType">

    <soap:binding style="rpc"

      transport="http://schemas.xmlsoap.org/soap/http"/>

    <operation name="sayHello">

      <soap:operation soapAction="sayHello"/>

```

Binding:

Direction to use the SOAP HTTP transport protocol.

Service: Service available at <http://www.examples.com/SayHello/>

Port: Associates the binding with the URI <http://www.examples.com/SayHello/> where the running service can be accessed.

WSDL – Definitions element

The <definitions> element must be the root element of all WSDL documents. It defines the name of the web service. Here is the piece of code from the last chapter that uses the definitions element.

```
<definitions name="HelloService"
  targetNamespace="http://www.examples.com/wsdl/HelloService.wsdl"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.examples.com/wsdl/HelloService.wsdl"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
</definitions>
```

From the above example, we can conclude that definitions:

- ❖ It is a container of all the other elements.
- ❖ It specifies that this document is called HelloService.
- ❖ It specifies a targetNamespace attribute. The targetNamespace is a convention of XML Schema that enables the WSDL document to refer to itself. In this example, we have specified a targetNamespace of <http://www.examples.com/wsdl/HelloService.wsdl>.
- ❖ It specifies a default namespace:
xmlns=http://schemas.xmlsoap.org/wsdl/. All elements without a

namespace prefix, such as message or portType, are therefore assumed to be a part of the default WSDL namespace.

- ❖ Which specifies numerous namespaces that are used throughout the remainder of the document.

A web service needs to define its inputs and outputs and how they are mapped into and out of the service. WSDL <types> element takes care of defining the data types that are used by the web service. Types are XML documents or document parts.

- The types element describes all the data types used between the client and the server.
- WSDL is not tied exclusively to a specific typing system.
- WSDL uses the W3C XML Schema specification as its default choice to define data types.
- If the service uses only XML Schema built-in simple types, such as strings and integers, then types element is not required.
- WSDL allows the types to be defined in separate elements so that the types are reusable with multiple web services.

Here is a piece of code taken from W3C specification. This code depicts how a types element can be used within a WSDL.

<types>

```
<schema targetNamespace="http://example.com/stockquote.xsd"
```

```
  xmlns="http://www.w3.org/2000/10/XMLSchema">
```

```
  <element name="TradePriceRequest">
```

```
    <complexType>
```

```
      <all>
```

```
        <element name="tickerSymbol" type="string"/>
```

```
      </all>
```

```
    </complexType>
```

```

</element>

<element name="TradePrice">

  <complexType>

    <all>

      <element name="price" type="float"/>

    </all>

  </complexType>

</element>

</schema>

</types>

```

Data types address the problem of identifying the data types and the formats you intend to use with your web services. Type information is shared between the sender and the receiver. The recipients of messages therefore need access to the information you used to encode your data and must understand how to decode the data.

5.4.2 MESSAGE ELEMENT

The `<message>` element describes the data being exchanged between the web service providers and the consumers.

- Each Web Service has two messages: input and output.
- The input describes the parameters for the web service and the output describes the return data from the web service.
- Each message contains zero or more `<part>` parameters, one for each parameter of the web service function.
- Each `<part>` parameter associates with a concrete type defined in the `<types>` container element.

Let us take a piece of code from the WSDL Example :

```
<message name="SayHelloRequest">
```

```

    <part name="firstName" type="xsd:string"/>

</message>

<message name="SayHelloResponse">

    <part name="greeting" type="xsd:string"/>

</message>

```

Here, two message elements are defined. The first represents a request message SayHelloRequest, and the second represents a response message SayHelloResponse. Each of these messages contains a single part element. For the request, the part specifies the function parameters

We specify a single firstName parameter. For the response, the part specifies the function return values; in this case, we specify a single greeting return value.

5.4.3 PORTTYPE ELEMENT

The <portType> element combines multiple message elements to form a complete one-way or round-trip operation. For example, a <portType> can combine one request and one response message into a single request/response operation.

This is most commonly used in SOAP services. A portType can define multiple operations. Let us take a piece of code from the WSDL Example:

```

<portType name="Hello_PortType">

    <operation name="sayHello">

        <input message="tns:SayHelloRequest"/>

        <output message="tns:SayHelloResponse"/>

    </operation>

</portType>

```

The portType element defines a single operation, called sayHello. The operation consists of a single input message SayHelloRequest and an output message SayHelloResponse.

Patterns of Operation WSDL supports four basic patterns of operation:

One-way The service receives a message. The operation therefore has a single input element. The grammar for a one-way operation is:

```
<wsdl:definitions .... > <wsdl:portType .... > *
    <wsdl:operation name="nmtoken">
        <wsdl:input name="nmtoken"? message="qname"/>
    </wsdl:operation>
</wsdl:portType >
</wsdl:definitions>
```

Request-response the service receives a message and sends a response. The operation therefore has one input element followed by one output element. To encapsulate errors, an optional fault element can also be specified. The grammar for a request-response operation is:

```
<wsdl:definitions .... >
    <wsdl:portType .... > *
        <wsdl:operation name="nmtoken" parameterOrder="nmtokens">
            <wsdl:input name="nmtoken"? message="qname"/>
            <wsdl:output name="nmtoken"? message="qname"/>
            <wsdl:fault name="nmtoken" message="qname"/>*
        </wsdl:operation>
    </wsdl:portType >
</wsdl:definitions>
```


Solicit-response The service sends a message and receives a response. The operation therefore has one output element followed by one input element. To encapsulate errors, an optional fault element can also be specified. The grammar for a solicit-response operation is:

```
<wsdl:definitions .... >

  <wsdl:portType .... > *

    <wsdl:operation name="nmtoken" parameterOrder="nmtokens">

      <wsdl:output name="nmtoken"? message="qname"/>

      <wsdl:input name="nmtoken"? message="qname"/>

      <wsdl:fault name="nmtoken" message="qname"/>*

    </wsdl:operation>

  </wsdl:portType >

</wsdl:definitions>
```

Notification The service sends a message. The operation therefore has a single output element. Following is the grammar for a notification operation:

```
<wsdl:definitions .... >

  <wsdl:portType .... > *

    <wsdl:operation name="nmtoken">

      <wsdl:output name="nmtoken"? message="qname"/>

    </wsdl:operation>

  </wsdl:portType >

</wsdl:definitions>
```

5.4.4 BINDING ELEMENT

The `<binding>` element provides specific details on how a portType operation will actually be transmitted over the wire.

- The bindings can be made available via multiple transports including HTTP GET, HTTP POST, or SOAP.
- The bindings provide concrete information on what protocol is being used to transfer portType operations.
- The bindings provide information where the service is located.
- For SOAP protocol, the binding is <soap:binding>, and the transport is SOAP messages on top of HTTP protocol.
- You can specify multiple bindings for a single portType.

The binding element has two attributes:

- Name
- Type.

```
<binding name="Hello_Binding" type="tns:Hello_PortType">
```

The name attribute defines the name of the binding, and the type attribute points to the port for the binding, in this case the "tns:Hello_PortType" port.

SOAP Binding WSDL 1.1 includes built-in extensions for SOAP 1.1. It allows you to specify SOAP-specific details including SOAP headers, SOAP encoding styles, and the SOAPAction HTTP header. The SOAP extension elements include the following:

- soap:binding
- soap:operation
- soap:body

Soap:Binding

This element indicates that the binding will be made available via SOAP. The style attribute indicates the overall style of the SOAP message format. A style value of rpc specifies an RPC format. The transport attribute indicates the transport of the SOAP messages. The value `http://schemas.xmlsoap.org/soap/http` indicates the SOAP HTTP transport, whereas `http://schemas.xmlsoap.org/soap/smtp` indicates the SOAP SMTP transport.

Soap:Operation

This element indicates the binding of a specific operation to a specific SOAP implementation. The soapAction attribute specifies that the SOAPAction HTTP header be used for identifying the service.

Soap:Body

This element enables you to specify the details of the input and output messages. In the case of HelloWorld, the body element specifies the SOAP encoding style and the namespace URN associated with the specified service.

Example

```
<binding name="Hello_Binding" type="tns:Hello_PortType">
  <soap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="sayHello">
    <soap:operation soapAction="sayHello"/>
    <input>
      <soap:body
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="urn:examples:helloservice"
        use="encoded"/>
    </input>
    <output>
      <soap:body
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="urn:examples:helloservice"
```

```

        use="encoded"/>

    </output>

</operation>

</binding>

```

5.5 BUILDING AND CONSUMING A WEB SERVICE

Web services give developers the ability to utilize four open Web standards:

HTTP – Hypertext Transfer Protocol

The standard protocol used over Port 80, which traverses firewalls, and is responsible for requesting and transmitting data over the Internet.

SOAP – Simple Object Access Protocol

An XML-inherent protocol that encloses a set of rules for data description and process. As a standard, this is the center-piece that complements the other three standards mentioned here.

XML – Extensible Markup Language

The most common markup language in which all this information is written.

WSDL – Web Services Description Language

An XML-based method used to identify Web Services and their access at runtime. .NET provides a tool called WSDL.exe, which essentially makes it quite easy to generate an XML Web service as an XML file.

This contains all the methods and instructions the Web Service has, and typically uses SOAP as its default. you create and consume a data-driven .NET XML Web service in 5 quick and easy steps.

.NET data access, Web server controls, such a datagrid, and some object-oriented programming concepts. If not, don't worry too much. If you complete the examples, and view the results, you should have no difficulty in keeping up and observing the causes and effects of what this tutorial entails.

There were other alternatives that could be used to access a Web service, such as Microsoft's MSXML component, which enabled you to communicate with the given Web Service over HTTP POST. However, this process, while acceptable, is just not .NET.

Step 1: Create the Web Service

First we'll create the Web service function or method that'll you'll call (or "expose") over the Internet as you would any object-oriented class. The difference is that we'll have to incorporate and import all the necessary Web Services namespaces, syntax and attributes, as well as our data namespaces, in this case.

So go ahead and copy the code below to a file called suppliers.asmx. Save it to your Inetpub/wwwroot folder, and then run it in your browser using <http://localhost/suppliers>.

By clicking the exposed method link, you'll be presented with the three main protocols that are available for your use. Just so you know, the file with the .asmx extension is the actual Web Service file that enables the ASP.NET runtime to return all pertinent exposed Web service methods and information.

```
<%@ WebService Language="C#" Class="GetInfo" %>

using System;

using System.Data;

using System.Data.SqlClient;

using System.Web.Services;

[WebService(Description="My Suppliers List Web Service")]

public class GetInfo : WebService
{
    [WebMethod(BufferResponse=true)]

    public DataSet ShowSuppliers (string str)
```

```

{
    SqlConnection dbConnection = new SqlConnection("server=(local);
    uid=sa;pwd=;database=Northwind;");

    SqlDataAdapter objCommand = new SqlDataAdapter("select
        ContactName, CompanyName, City, Phone from Suppliers
        where Country = '" + str + "' order by ContactName
        asc", dbConnection);

    DataSet DS = new DataSet();
    objCommand.Fill(DS);
    return DS;
    dbConnection.Close();
    dbConnection = null;
}
}

```

The <%@ WebService Language="C#" Class="GetInfo" %> directive sets up the file as a Web Service, gives it a name and specifies the language it uses. Aside from your typical data namespace import, you also add the Web Services namespace:

```
using System.Web.Services
```

In VB this would be:

```
Imports System.Web.Services
```

Here I've added the [WebService(Description="My Suppliers List Web Service")], which gives a custom description. Next we create our class, which, in order that it be exposed, and able to inherit the Webservice base class, must be a public class.

```
public class GetInfo : WebService
```

```
{
```

The method that's to be exposed via the WebMethod attribute. Every class that's enclosed within a WebMethod will be exposed as a service. Also, I boost performance by setting the BufferResponse to true.

```
[WebMethod (BufferResponse=true)]
```

If you prefer to have the description right below the exposed WebMethod link, you can achieve it like this in C#:

```
[WebMethod (Description="My Suppliers List Web  
Service",BufferResponse=true)]
```

The VB version would incorporate the description and BufferResponse in one statement, namely WebMethod, that used angle brackets. This is placed after the class creation and Web service inheriting line, and on the same line with, but before the function, like so:

```
<WebMethod(Description:="My Suppliers List Web  
Service",Buffer Response:=True)>
```

```
Public Function
```

```
ShowSuppliers (ByVal str As String) As DataSet
```

If you're really savvy you can pass this and other parameters with the help of C# structs (scaled down classes) to enumerate your data value types, and provide you with better memory allocation. But for now, we'll stick to the basics:

```
public DataSet ShowSuppliers (string str)
```

The dataset method to return us exactly that. Lastly, we perform typical data connection and access and return our results, and we're done! So far so good? After viewing this in your browser, the above code should to start to make sense. Let's go to Step 2.

STEP 2 : CONSUME THE WEB SERVICE SOURCE FILE

Next, append ?WSDL to the Web services URI (Uniform Resource Identifier) like so:

`http://localhost/suppliers.asmx?WSDL`

This is the WSDL document that the client will use to access this service. Even so, you don't have to know much about this unreadable code to produce results, which is where the WSDL.exe command-line tool comes into play. Due to the open protocol nature of Web services, this tool enables you to consume non-.NET Web Services as well.

You can bypass WSDL and test the XML results instantly through HTTP GET protocol, by typing the following into your browser:

`http://localhost/suppliers.asmx/ShowSuppliers?str=USA`

This passes USA as a parameter to the `ShowSuppliers` class method. Note that if you're using .NET SDK Beta 1.1 (v.1.1.4322), it seems to prefer HTTP POST protocol, so invoke the Web Service through its exposed method link. The XML results? A little crazy, huh?

So, to create our proxy class sourcefile, make a batch file named `makeWS.bat` and type in:

[C#]

```
wsdl.exe /l:CS /n:WService /out:bin/GetSuppliers.cs
http://localhost/suppliers.asmx?WSDL
```

[VB]

```
wsdl.exe /l:VB /n:WService /out:bin/GetSuppliers.vb
http://localhost/suppliers.asmx?WSDL
```

Now locate your new batch file, and double-click on it to run it. Once you've done this, you will have created, rather consumed, the proxy class or source file `GetSuppliers.cs` right from the `.asmx` file. Have a look in your bin folder.

STEP 3 : BUILD OUR OBJECT

To invoke the .NET C# compiler (csc.exe) or VB compiler (vbc.exe) to convert your source file into an assembly or DLL. Create a new .bat file named makelib.bat and type in:

[C#]

```
csc /t:library /out:binGetSuppliers.dll binGetSuppliers.cs
/reference:System.dll,System.Data.dll,System.Web.dll,
System.Web.Services.dll,System.XML.dll /optimize
```

[VB]

```
vbc /t:library /out:binGetSuppliers.dll binGetSuppliers.vb
/reference:System.dll,System.Data.dll,System.Web.dll,
System.Web.Services.dll,System.XML.dll /optimize
```

All this does is compile the source file from the bin folder to a DLL of set name, to the bin folder.

The `/t:library` instructs the compiler to create a dll (dynamic link library), rather than an exe (executable) file, with `/reference:` importing the necessary dll's libraries that will be used in our Web service. Finally, we `/optimize` our dll to produce smaller, faster, and more efficient output.

STEP 4 : PUT IT ALL TOGETHER

Now copy and paste the code below to an .aspx .NET page and name it. This code is for the page that will access the assembly/DLL in your bin folder, and with all the Web server controls, pass the appropriate parameters to the Web Service. Go ahead and run it (<http://localhost/Websrvce.aspx>).

```
<%@ Page Language="C#" Explicit="true" Strict="true" Buffer="true"%>
<%@ Import Namespace="System.Data" %>
```

```

<%@ Import Namespace="System.Data.SqlClient" %>

<%@ Import Namespace="WService" %>

<html>

<script language="C#" runat="server">

void Page_Load(Object sender, EventArgs e)

{

    Response.Flush();

}

void SubmitBtn_Click (object src, EventArgs e)

{

int RcdCount;

string Catg = DropDown1.SelectedItem.Text;

    //Instantiate DLL

    GetInfo supInfo = new GetInfo();

        //Pass parameter into DLL function

    DataSet MyData = supInfo.ShowSuppliers(Catg);

    MyDataGrid.DataSource = MyData.Tables[0].DefaultView;

    MyDataGrid.DataBind();

    RcdCount = MyData.Tables[0].Rows.Count;

    if (RcdCount <= 0)

    {

        Message.InnerHtml = "<b>No results were found for <FONT
Color=Red><i>" + Catg + "</i></font>";

        MyDataGrid.Visible = false; //Hide Results until needed

```

```

    }

    else

    {

        Message.InnerHtml = "<b><FONT Color=Red><i>" + Catg +
            "</i></font> has " + RcdCount + " local suppliers</b>";

        MyDataGrid.Visible = true;

    }

}

</script>

<body style="font: 10pt verdana">

<h4>Accessing Data with Web Services</h4>

<form runat="server">

<asp:DropDownList id=DropDown1 runat="server">

<asp:ListItem>Australia</asp:ListItem>

<asp:ListItem>Brazil</asp:ListItem>

<asp:ListItem>Canada</asp:ListItem>

<asp:ListItem>Denmark</asp:ListItem>

<asp:ListItem>Finland</asp:ListItem>

<asp:ListItem>France</asp:ListItem>

<asp:ListItem>Germany</asp:ListItem>

<asp:ListItem>Italy</asp:ListItem>

<asp:ListItem>Japan</asp:ListItem>

<asp:ListItem>Netherlands</asp:ListItem>

<asp:ListItem>Norway</asp:ListItem>

```

```

<asp:ListItem>Singapore</asp:ListItem>

<asp:ListItem>Spain</asp:ListItem>

<asp:ListItem>Sweden</asp:ListItem>

<asp:ListItem>UK</asp:ListItem>

<asp:ListItem>USA</asp:ListItem>

</asp:DropDownList>

<asp:button text="Submit" OnClick="SubmitBtn_Click" runat=server/>

<p>

<span id="Message" runat="server"/>

<p>

<ASP:DataGrid id="MyDataGrid" runat="server"

AutoGenerateColumns="True"

Width="100%"

BackColor="White"

Border="1"

BorderWidth="1"

CellPadding="1"

CellSpacing="1"

Font-Size="10pt"

HeaderStyle-BackColor="White"

HeaderStyle-ForeColor="Blue"

AlternatingItemStyle-BackColor="White"

AlternatingItemStyle-ForeColor="Black"

ShowFooter="false"

```

```

/>

</form>

</body>

<%@ Import Namespace="WService" %>

```

We then instantiate it or reference our object:

[C#]

```
GetInfo supInfo = new GetInfo();
```

[VB]

```
Dim supInfo As New WService.GetInfo()
```

This is our Web Service class. We then retrieve our dropdown list parameter, and pass it to our ShowSuppliers constructor method like so:

[C#]

```
string Catg = DropDown1.SelectedItem.Text; DataSet MyData =
supInfo.ShowSuppliers(Catg);
```

[VB]

```
Dim Catg As String = DropDown1.SelectedItem.
```

5.6 WEB APPLICATION 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

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

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.

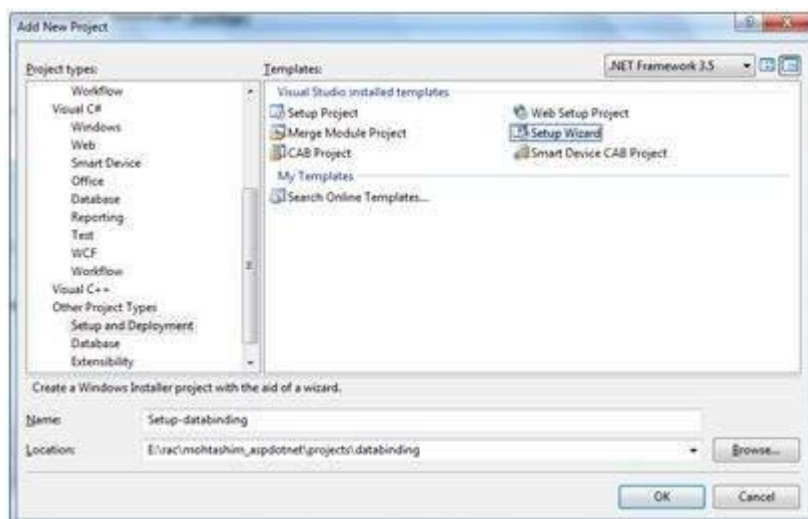
Creating a Setup Project

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.

Open the project and take the following steps:

Step (1) : Select File -> Add -> New Project with the website root directory highlighted in the Solution Explorer.

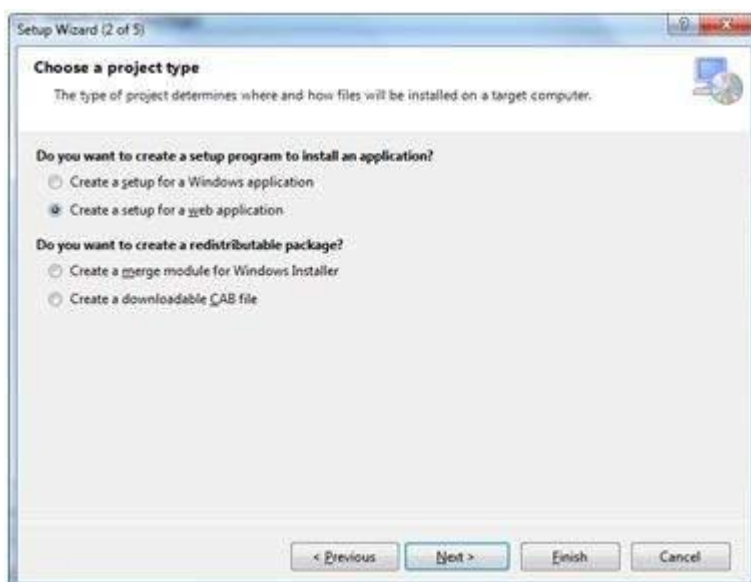
Step (2) : Select Setup and Deployment, under Other Project Types. Select Setup Wizard.



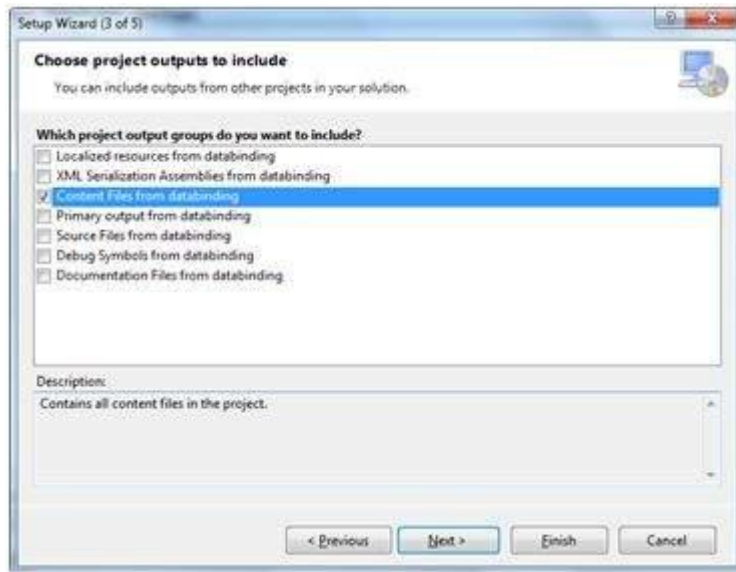
Step (3) : Choosing the default location ensures that the set up project will be located in its own folder under the root directory of the site. Click on okay to get the first splash screen of the wizard.



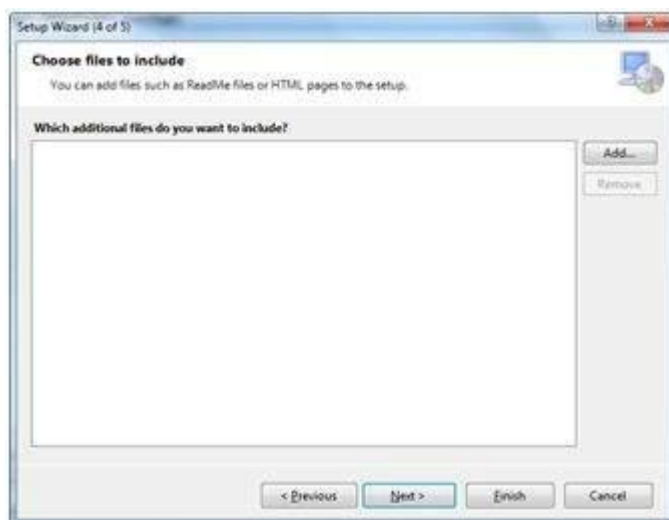
Step (4) : Choose a project type. Select 'Create a setup for a web application'.



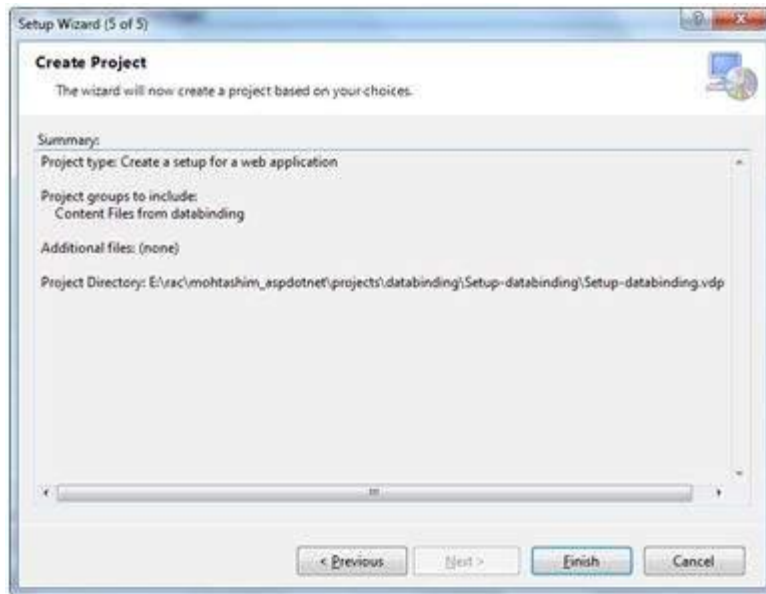
Step (5) : Next, the third screen asks to choose project outputs from all the projects in the solution. Check the check box next to 'Content Files from...'



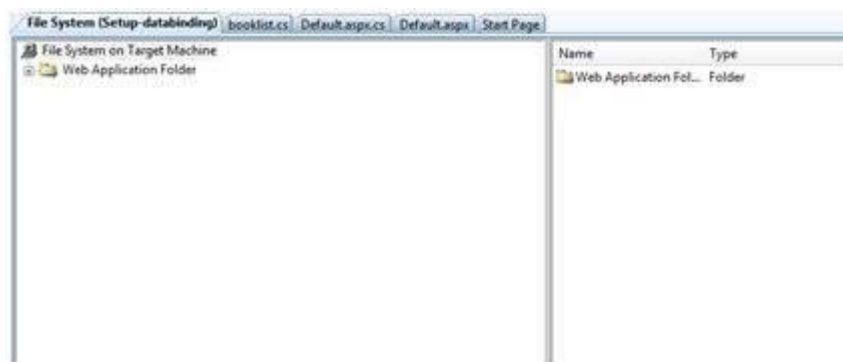
Step (6) : The fourth screen allows including other files like ReadMe. However, in our case there is no such file. Click on finish.



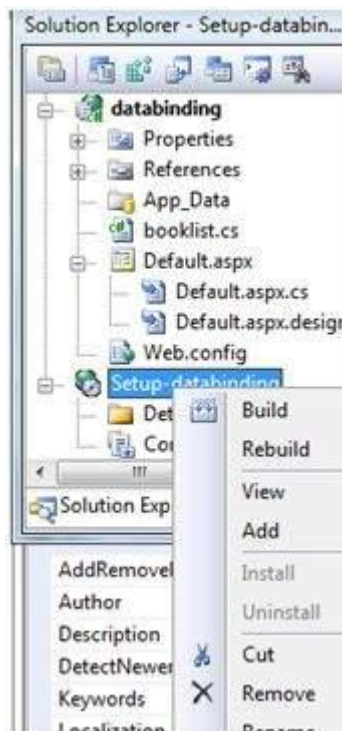
Step (7) : The final screen displays a summary of settings for the set up project.



Step (8) : The Set up project is added to the Solution Explorer and the main design window shows a file system editor.



Step (9) : Next step is to build the setup project. Right click on the project name in the Solution Explorer and select Build.



Step (10) : When build is completed, you get the following message in the Output window:

```
Packaging file 'Web.config'...
Packaging file 'Default.aspx'...
===== Build: 2 succeeded or up-to-date, 0 failed, 0 skipped =====
```

Two files are created by the build process:

- Setup.exe
- Setup-databinding.msi

You need to copy these files to the server. Double-click the setup file to install the content of the .msi file on the local machine.

5.7 KEY TERMS

XmlWriter The XML representation of the DataSet can be written to a file, a stream, an XmlWriter .

WriteSchema Writes the current contents of the DataSet as XML data with the relational structure as inline XML Schema.

DiffGram Writes the entire [DataSet](#) as a DiffGram, including original and current values. For more information, see [DiffGrams](#)

Hidden The column is not written in the XML output.

SOAP SOAP is an XML-based protocol for accessing web services over HTTP. It has some specification which could be used across all applications.

WSDL web service description language WSDL breaks down web services into three specific, identifiable elements that can be combined or reused once defined.

Data types: The data types to be used in the messages are in the form of XML schemas.

Message: It is an abstract definition of the data, in the form of a message presented either as an entire document or as arguments to be mapped to a method invocation.

Port type: It is an abstract set of operations mapped to one or more endpoints, defining the collection of operations for a binding.

Binding: It is the concrete protocol and data formats for the operations and messages defined for a particular port type.

Port: It is a combination of a binding and a network address, providing the target address of the service communication.

Documentation: This element is used to provide human-readable documentation and can be included inside any other WSDL element.

Import: This element is used to import other WSDL documents or XML Schemas. WSL parts are usually generated automatically using web services-aware tools.

Soap:Binding This element indicates that the binding will be made available via SOAP. The style attribute indicates the overall style of the SOAP message format.

Soap:Operation This element indicates the binding of a specific operation to a specific SOAP implementation.

Soap:Body This element enables you to specify the details of the input and output messages.

HTTP – Hypertext Transfer Protocol The standard protocol used over Port 80, which traverses firewalls, and is responsible for requesting and transmitting data over the Internet.

SOAP – Simple Object Access Protocol.

XML – Extensible Markup Language. The most common markup language in which all this information is written.

XCOPY Deployment XCOPY deployment means making recursive copies of all the files to the target folder on the target machine.

5.7.1 2 MARKS QUESTION AND ANSWER

1. Define XML

An XML representation of a DataSet, with or without its schema. If schema information is included inline with the XML, it is written using the XML Schema definition language (XSD).

2. What is meant by Dataset?

DataSet is written as XML data, the rows in the DataSet are written in their current versions. The DataSet can also be written as a DiffGram so that both the current and the original values of the rows will be included.

3. Define XML Element.

This is the default. The column is written as an XML element where the ColumnName is the name of the element and the contents of the column are written as the text of the element.

4. Define Attributes

The column is written as an XML attribute of the XML element for the current row where the ColumnName is the name of the attribute and the contents of the column are written as the value of the attribute.

5. How to create the User Interface

- The user interface for this application consists of the following:
- A [DataGridView](#) control that displays the contents of the XML file as data.
- A [TextBox](#) control that displays the XML schema for the XML file.
- Two [Button](#) controls.

6. What is SOAP?

SOAP is an XML-based protocol for accessing web services over HTTP. It has some specification which could be used across all applications. SOAP was developed as an intermediate language so that applications built on various programming languages could talk easily to each other and avoid the extreme development effort.

7. Define SOAP Message Structure

SOAP messages are normally auto-generated by the web service when it is called. Whenever a client application calls a method in the web service, the web service will automatically generate a SOAP message which will have the necessary details of the data which will be sent from the web service to the client application.

8. What is web service language?

WSDL breaks down web services into three specific, identifiable elements that can be combined or reused once defined. The three major elements of WSDL that can be defined separately are Types, Operations and Binding.

9. Define Message Element

The <message> element describes the data being exchanged between the web service providers and the consumers each Web Service has two messages input and output. The input describes the parameters for the web service and the output describes the return data from the web service.

10.What is PortType element?

The <portType> element combines multiple message elements to form a complete one-way or round-trip operation.

11. What is Binding Element?

The <binding> element provides specific details on how a portType operation will actually be transmitted over the wire. The bindings can be made available via multiple transports including HTTP GET, HTTP POST, or SOAP. The binding element has two attributes: Name and Type.

12. Define Soap Operation

This element indicates the binding of a specific operation to a specific SOAP implementation. The soapAction attribute specifies that the SOAPAction HTTP header be used for identifying the service.

13. Define HTTP

Hypertext Transfer Protocol The standard protocol used over Port 80, which traverses firewalls, and is responsible for requesting and transmitting data over the Internet.

14. Define WSDL

Web Services Description Language(WSDL) An XML-based method used to identify Web Services and their access at runtime. .NET provides a tool called WSDL.exe, which essentially makes it quite easy to generate an XML Web service as an XML file.

15. What is Local deployment?

The entire application is contained within a virtual directory and all the contents and assemblies are contained within it and available to the application.

5.7.2 5 MARKS QUESTIONS

1. How to write Dataset content as XML data?
2. Explain how to create the XML document and load the data string.

3. Write the steps how to read XML data into dataset?
4. Explain the Remote the Method call using XML.
5. Explain Building Block of SOAP .Write some advantage of SOAP ?

5.7.3 10 MARK QUESTIONS

1. Explain WSDL and write some features of WSDL.
2. List out the port type and messaging element in WSDL.
3. Describe Web Application Deployment in detail.
4. Explain the consuming a web service in detail.
5. How to create the web service in WSDL?

Multiple Choice Question With Answer

1. What does XML stand for?
 - A. eXtra Modern Link
 - B. eXtensible Markup Language
 - C. Example Markup Language
 - D. X-Markup Language

Ans: B

2. What is the correct syntax of the declaration which defines the XML version?:

- A. <xml version="A.0" />
- B. <?xml version="A.0"?>
- C. <?xml version="A.0" />
- D. None of the above

Ans: B

3. Which statement is true?
 - A. All the statements are true
 - B. All XML elements must have a closing tag
 - C. All XML elements must be lower case
 - D. All XML documents must have a DTD

Ans: B

4. Is it easier to process XML than HTML?

- A. Yes
- B. No
- C. Sometimes
- D. Cant say

Ans: A

5. Which of the following programs support XML or XML applications?:

- A. Internet Explorer 5.5
- B. Netscape D.7
- C. RealPlayer.
- D. both A and B

Ans: D

6. Kind of Parsers are

- A. well-formed
- B. well-documented
- C. non-validating and validating
- D. none of the above

Ans: C

7. Well formed XML document means

- A. it contains a root element
- B. it contain an element
- C. it contains one or more elements
- D. must contain one or more elements and root element must contain all other elements

Ans: D

8. Comment in XML document is given by

- A. <?-- -->
- B. <!-- --!>
- C. <!-- -->

D. </-- -- >

Ans: C

9. When processing an output XML, "new line" symbols

A. are copied into output "as is", i.e. "CR+LF" for Windows, CR for Macintosh, LF for Unix.

B. are converted to single LF symbol

C. are converted to single CR symbol

D. are discarded

Ans: B

10. Which of the following strings are a correct XML name?

A. _myElement

B. my Element

C. #myElement

D. None of the above

Ans: A

11. Which of the following strings are a correct XML name?

A. xmlExtension

B. xslNewElement

C. XMLElement#123

D. All

Ans: B

12. Which of the following XML fragments are well-formed?

A. <?xml?>

B. <?xml version="A.0"?>

C. <?xml encoding="JIS"?>

D. <?xml encoding="JIS" version="A.0"?>

Ans: B

13. What are the predefined attributes

- A. xml:lang
- B. xml:space
- C. both
- D. none.

Ans: C

14. Kind of Parsers are

- A. well-formed
- B. validating
- C. non-validating
- D. Both B & C

Ans: D

15. Valid XML document means (most appropriate)

- A. the document has root element
- B. the document contains atleast one or more root element
- C. the XML document has DTD associated with it & it complies with that DTD
- D. Each element must nest inside any enclosing element property

Ans: C

16. XML uses the features of

- A. HTML
- B. XHTML
- C. VML
- D. SGML

Ans: D

17. XML document can be viewed in

- A. IE C.0
- B. IE B.0

C. IE 6.0

D. IE X.0

Ans: C

18. There is a way of describing XML data, how?

A. XML uses a DTD to describe the data

B. XML uses XSL to describe data

C. XML uses a description node to describe data

D. Both A and C

Ans: D

19. What does DTD stand for?

A. Direct Type Definition

B. Document Type Definition

C. Do The Dance

D. Dynamic Type Definition

Ans: B

20. DTD includes the specifications about the markup that can be used within the document, the specifications consists of all EXCEPT

A. the browser name

B. the size of element name

C. entity declarations

D. element declarations

Ans: A

21. Which of the following XML documents are well-formed?

A. <firstElement>some text goes here

<secondElement>another text goes here</secondElement>

</firstElement>

B. <firstElement>some text goes here</firstElement>

<secondElement> another text goes here</secondElement>

C. <firstElement>some text goes here

<secondElement> another text goes here</firstElement>
 </secondElement>
 D. </firstElement>some text goes here
 </secondElement>another text goes here
 <firstElement>

Ans: B

22. Which of the following XML fragments are well-formed?

- A. <myElement myAttribute="someValue"/>
- B. <myElement myAttribute=someValue/>
- C. <myElement myAttribute='someValue'>
- D. <myElement myAttribute="someValue'"/>

Ans: A

23. How can we make attributes have multiple values:

- A. <myElement myAttribute="value1 value2"/>
- B. <myElement myAttribute="value1" myAttribute="value2"/>
- C. <myElement myAttribute="value1, value2"/>
- D. attributes cannot have multiple values

Ans: D

24. Which of the following XML fragments are well-formed?

- A. <myElement myAttribute="value1 <= value2"/>
- B. <myElement myAttribute="value1 & value2"/>
- C. <myElement myAttribute="value1 > value2"/>
- D. None of the above

Ans: C

25. The use of a DTD in XML development is:

- A. required when validating XML documents
- B. no longer necessary after the XML editor has been customized
- C. used to direct conversion using an XSLT processor
- D. a good guide to populating a templates to be filled in when generating an

XML document automatically

Ans: A

26. Parameter entities can appear in

- A. xml file
- B. dtd file
- C. xsl file
- D. Both 1 and 2

Ans: B

27. Attribute standalone="no" should be included in XML declaration if a document:

- A. is linked to an external XSL stylesheet
- B. has external general references
- C. has processing instructions
- D. has an external DTD

Ans: D

28. In XML

- A. the internal DTD subset is read before the external DTD
- B. the external DTD subset is read before the internal DTD
- C. there is no external type of DTD
- D. there is no internal type of DTD

Ans: A

29. Disadvantages of DTD are

- (i) DTDs are not extensible
- (ii) DTDs are not in to support for namespaces
- (iii) there is no provision for inheritance from one DTDs to another

- A. (i) is correct
- B. (i),(ii) are correct
- C. (ii),(iii) are correct

D. (i),(ii),(iii) are correct

Ans: D

30. To use the external DTD we have the syntax

A. `<?xml version="A.0" standalone="no"?>`

`<! DOCTYPE DOCUMENT SYSTEM "order.dtd"?>`

B. `<?xml version="A.0" standalone="yes"?>`

`<! DOCTYPE DOCUMENT SYSTEM "order.dtd"?>`

(3) `<?xml version="A.0" standalone="no"?>`

`<! DOCTYPE DOCUMENT "order.dtd"?>`

D. `<?xml version="A.0" standalone="yes"?>`

`<! DOCTYPE DOCUMENT SYSTEM "order.dtd"?>`

Ans: A

31. To add the attribute named Type to the `<customer>` tag the syntax will be

A. `<customer attribute Type="exelent">`

B. `<customer Type attribute ="exelent">`

C. `<customer Type attribute_type="exelent">`

D. `<customer Type=" exelent" >`

Ans: D

32. The syntax for parameter entity is

A. `<! ENTITY % NAME DEFINITION>`

B. `< ENTITY % NAME DEFINITION>`

C. `<! ENTITY $ NAME DEFINITION>`

D. `< ENTITY % NAME DEFINITION>`

Ans: A

33. You can name the schema using the name attribute like

A. `<schema attribute="schema1">`

B. `<schema nameattribute="schema1">`

C. `<schema nameattri="schema1">`

D. <schema name="schema1">

Ans: D

34. The default model for complex type, in XML schemas for element is

A. textOnly

B. elementOnly

C. no default type

D. both 1 & 2

Ans: B

35. Microsoft XML Schema Data types for Hexadecimal digits representating octates

A. UID

B. UXID

C. UUID

D. XXID

Ans: C

36. A schema describes

(i) grammer

(ii) vocabulary

(iii) structure

(iv) datatype of XML document

A. (i) & (ii) are correct

B. (i),(iii) ,(iv) are correct

C. (i),(ii),(iv) are correct

D. (i),(ii),(iii),(iv) are correct

Ans: D

37. Microsoft XML Schema Data Type “ boolean” has values

A. True ,False

B. True ,False or 1,0

- C. 1,0
 - D. any number other than zero and zero
- Ans: C

38. Simple type Built into Schema “ data’ represent a data in
- A. MM-DD-YY
 - B. Dd-MM-YY
 - C. YY-MM-DD
 - D. YYYY-MM-DD
- Ans: D

39. In simple Type Built into XML schema Boolean type holds
- A. True, False
 - B. 1,0
 - C. both A. & B.
 - D. True/False and any number except 0
- Ans: C

40. In simple type built into XML schema type float has single precision of _____ floating point
- A. 16 bit
 - B. 32 bit
 - C. 8 bit
 - D. 4 bit
- Ans: C

41. The XML DOM object is
- A. Entity
 - B. Entity Reference
 - C. Comment Reference
 - D. Comment Data
- Ans: B

42. Attribute of the document interface in DOM is/are

- (i)doctype
- (ii)implementation
- (iii)documentElement

which are read only attributes

- A. (i) only
- B. (ii) only
- C. (ii),(iii) only
- D. all

Ans: D

43. The default model for complex type, in XML schemas for element is

- A. textOnly
- B. elementOnly
- C. no default type
- D. both a & b

Ans: B

44. To create a choice in XML schemas, we use the

- A. <xsd:select> element
- B. <xsd:multi> element
- C. <xsd:choise> element
- D. <xsd:single> element

Ans: C

45. The XML DOM object is

- A. Entity
- B. Entity Reference
- C. Comment Reference
- D. Comment Data

Ans: B

46. To create a data island we use the _____HTML element

- A. <XML>
- B. <dataisland>
- C. <Island>
- D. <XMLIsland>

Ans: A

47. To Bind the HTML elements with DSO we use _____ attribute

- A. DATASOURCE
- B. DATAFIELD
- C. DATASRC
- D. DATAFLD

Ans: A,C

48. To bind the HTML element <INPUT> Type in text with the datasource “dsoCustomer” we use

- A. <INPUT TYPE=”TEXT” DATAFIELD=”#dsoCustomer”>
- B. <INPUT TYPE=”TEXT” DATASRC=” dsoCustomer”>
- C. <INPUT TYPE=”TEXT” DATASRC=” #dsoCustomer” >
- D. <INPUT TYPE=”TEXT” DATAFLD=” #dsoCustomer”>

Ans: C

49. XML DSOs has the property for the number of pages of data the recordset contains

- A. count
- B. number
- C. pageCount
- D. pageNumber

Ans: C

50. Whats so great about XML?

- A. Easy data exchange

- B. High speed on network
- C. Only B.is correct
- D. Both A. & B.

Ans: D

51. Which of the following object is not an ASP component?

- A. LinkCounter
- B. Counter
- C. AdRotator
- D. File Access

Ans: LinkCounter

52. The first event triggers in an aspx page is.

- A. Page_Init()
- B. Page_Load()
- C. Page_click()

Ans: Page_Init()

53. Difference between Response.Write() andResponse.Output.Write().

- A. Response.Output.Write() allows you to buffer output
- B. Response.Output.Write() allows you to write formatted output
- C. Response.Output.Write() allows you to flush output
- D. Response.Output.Write() allows you to stream output

Ans: Response.Output.Write() allows you to write formatted output

53. Which of the following method must be overridden in a custom control?

- A. The Paint() method
- B. The Control_Build() method
- C. The default constructor
- D. The Render() method

Ans: The Render() method

54. How do we create a FileSystemObject?

- A. Server.CreateObject("Scripting.FileSystemObject")
- B. Create("FileSystemObject")
- C. Create Object:"Scripting.FileSystemObject"
- D. Server.CreateObject("FileSystemObject")

Ans: Server.CreateObject("Scripting.FileSystemObject")

55. Which of the following tool is used to manage the GAC?

- A. RegSvr.exe
- B. GacUtil.exe
- C. GacSvr32.exe
- D. GacMgr.exe

Ans: GacUtil.exe

56. What class does the ASP.NET Web Form class inherit from by default?

- A. System.Web.UI.Page
- B. System.Web.UI.Form
- C. System.Web.GUI.Page
- D. System.Web.Form

Ans: System.Web.UI.Page

57. Can we use view state in MVC ?

- A) Yes
- B) No
- C) Both A & B
- D) None

Ans: B

58. Which web server is developed by Microsoft?

- a) Apache Tomcat
- b) Caudium
- c) Internet Information Services

d) WEBrick

Ans: C

59. Which Command is used to specify setting of an .aspx file?

a) Class

b) Directives

c) Events

d) Validation

Ans: B

60. Which of the following is the common property of webserver controls that assign a small piece of text when a mouse pointer is held over the control for a short period of time?

a) Access key

b) Tooltip

c) SkinID

d) TabIndex

Ans: B

61. The method used to change styles of the element in ASP.NET Webpage is called

a) Master Page

b) Child Page

c) cascading style sheet

d) UTF-8

Ans: C

62. Which webserver control is used to display advertisement in ASP.Net Web page?

- a)Image
- b) Image Map
- c) Panel
- d) AdRotator

Ans: D

63. Which of the following control shows data in tabular format and allow sorting, Paging, edit, delete each record?

- a)Listbox
- b)Gridview
- c) Repeater
- d) None of these

Ans: b

64. Which of the following web server controls used as a container for other server controls in ASP .Net Web page?

- a) Placeholder
- b) Table
- c) Panel
- d) ImageMap

Ans:C

65. By using which of the following attribute, HTML elements are transformed to HTML Server Control?

- a) runat="client"
- b) runat="server"

c)runat="browser"

d) runat="host"

Ans: B

66. Which of the following validation control is used to ensure that an user does not skip a form entity field?

a) CompareValidator

b) RegularExpressionValidator

c)RangeValidator

d) RequiredFieldValidator

Ans: D

67. ASP.NET is built on the _____ which allows the programmers to execute its code using any .NET language

A) Common Language Runtime.

B) Custom Language Runtime.

C) Custom Language Repository.

D) Common Language Repository.

Ans: A

68. _____is the process of removing unwanted resources when they are no longer required.

A) Common Language Runtime.

B) Common Type System.

C) Exception Handler.

D) Garbage Collection.

Ans: D

69. What is used to validate complex string patterns like an e-mail address?

A) Extended expressions

- B) Basic expressions
- C) Regular expressions
- D) Irregular expressions

Ans: C

70. File extension used for ASP.NET files.

- A) .Web
- B) .ASP
- C) .ASPX
- D) None of the above

Ans: C

71. A project cannot contain which one of the following content files:

- A) Page file (.aspx)
- B) User control (.ascx)
- C) Web service (.asmx)
- D) All the above

Ans : D

72. _____ is the directive used to specify the default programming language for a page

- A) Import directive
- B) Page directive
- C) Code declaration block
- D) Code Render block

Ans: B

73. Which of the following is not ASP.NET Page Life cycle Event?

- A) PreInt
- B) Int
- C) Load
- D) DisLoad

Ans : D

74. Validator that evaluates the value of an input control against another input control on the basis of specified operator

- A) Range Validator
- B) Compare Validator
- C) RequireField Validator
- D) RegularExpression Validator

Ans: B

75. Validator that is used to make a control required

- A) Range Validator
- B) Compare Validator
- C) RequireField Validator
- D) RegularExpression Validator

Ans: C