Kavayitri Bahinabai Chaudhari
North Maharashtra University

SEM. IV ▪ BCA 404 (A)

BCA

Bachelor in Computer Application

Web Development Technology - II

▪ Harshal V. Patil
▪ Rajashri P. Deshmukh

Prashant

As per U.G.C. Guidelines and also on the basis of revised syllabus of
Kavayitri Bahinabai Chaudhari North Maharashtra University
with effect from Academic Year 2023-24. Also useful for all Universities.

SEM IV ▪ BCA-404 (A)

Bachelor in Computer Application

WEB DEVELOPMENT TECHNOLOGY- II

Text Book Development Committee

- C O O R D I N A T O R -

Sanjay E. Pate
Department of Computer Science,
Nanasaheb Y. N. Chavan Arts, Science and Commerce College, Chalisgaon.

- A U T H O R S -

Harshal V. Patil
Department of Computer Management,
Bhusawal Arts, Commerce and P.O. Nahta Science College, Bhusawal.

Rajashri P. Deshmukh
Department of Computer Management,
Nanasaheb Y. N. Chavan Arts, Science and Commerce College, Chalisgaon.

Prashant
Publications

WEB DEVELOPMENT TECHNOLOGY - II

SEM IV ▪ BCA-404 (A)

Publisher | Printer
Prashant Publications
3, Pratap Nagar, Sant Dnyaneshwar Mandir Rd,
Near Nutan Maratha College, Jalgaon 425001.

Phone | Website | E-mail
0257-2235520, 0257-2232800, 9665626717
www.prashantpublications.com
prashantpublication.jal@gmail.com

Distributor
Prashant Book House
17, Stadium Shopping Centre, Opp. SBI, Jalgaon.
Ph. 0257-2227231

Available at all leading booksellers
Jalgaon | Dhule | Nadurbar

First Edition, January, 2024

ISBN 978-81-19120-68-0

Price ₹ 120/-

PREFACE

WEB DEVELOPMENT TECHNOLOGY-II is a Simple version for S.Y.B.C.A. students, published by Prashant Publication.

This text is in accordance with the new syllabus CBCS-2023 recommended by the Kavayitri Bahainabai Chaudhari North Maharashtra University, Jalgaon, which has been serving the need of S.Y.B.C.A. Computer Science students from various colleges. This text is also useful for the student of Engineering, B.Sc. (Information Technology and Computer Science), M.Sc, M.C.A. B.B.M., M.B.M. other different Computer courses.

We are extremely grateful to Prof. Dr. S. R. Kolhe, Professor, Director, School of Computer Sciences and Chairman, Board of Studies, Kavayitri Bahinabai Chaudhari North Maharashtra University, Jalgaon for his valuable guidance.

We are obligated to Principals Dr. S. R. Jadhav, Nanasaheb Y. N. Chavan Arts, Science and Commerce College, Chalisgaon, Dr. B. H. Barhate, Vice-principal, P. O. Nahta College, Bhusawal and Librarians and staff of respective colleges for their encouragement.

We are very much thankful to Shri. Rangrao Patil, Shri. Pradeep Patil of Prashant Publications, who has shown extreme co-operation during the preparation of this book, for getting the book published in time and providing an opportunity to be a part of this book.

We welcome any suggestions aimed at enhancing the content of this book, and we look forward to constructive feedback from our readers.

- Authors

S Y L L A B U S

Bachelor in Computer Applications (BCA)

Lab on Web Development Technologies - II | Sem. IV | BCA-404 (A)

w.e.f. Academic Year 2023-24 (Semester System 60 + 40 Pattern)

Unit 1 : NET Framework **(10 L, 15 M)**

Introduction to .NET framework,Origin of .Net Technology, Features of .NET framework: CTS, CLS, CLR, MSIL, IL, JIT.

Unit 2 : Introduction to C#.Net **(10 L, 15 M)**

Introduction to C #, Advantages & Disadvantages of C#, Programming Structure of C#, Basic Constructs – Variables, Data types, Operators, arrays, functions, Control Statements (if statement, if....else statement, nesting of if....else statement, the else if ladder, switch statement), Looping Construct(while statement, do statement, for statement)

Unit 3 : Introduction to ASP.NET web **(10 L, 20 M)**

History of Asp.Net, Introduction to Asp.Net, Features of Asp.Net, Structure of Asp.Net Page, ASP.NET Web Pages Model (Single Page Model, Two Page Model).

Unit 4 : ASP.NET Controls **(10 L, 20 M)**

Working with Web Server Controls (Label, Textbox, Button, Checkbox, Radio button, ListBox, Dropdown etc, Navigation controls (Tree view, Menu navigation),ASP.Net Rich Controls(Adrotator, Calender), Validation Controls (Required Field Validator, Range Validator, Compare Validator),

Unit 5 : ASP.Net Intrinsic Objects and State Management **(10 L, 20 M)**

HTTP Request Object, HTTP Responce Object, HTTP Server, HTTP Application State Object, HTTP SessionState Object, State Management (View State, Session, Application, Cookies) Global Application Class (global.asax),WebConfig File.

Unit 6 : Master Page & ADO .NET **(15 L, 20 M)**

Master pages and content Pages Basic, Architecture of ADO.NET, Create and retrieve data from Database Connections, SqlDataSource Controls, ASP.NET Data-Bound Controls (GridView, Repeater, DataList, Details View, Form View).

CONTENTS

1. Chapter

NET Framework

S y l l a b u s :

1Introduction to .NET framework,Origin of .Net Technology, Features of .NET framework: CTS, CLS, CLR, MSIL, IL, JIT.

(10 L, 15 M)

Introduction to .NET framework :

An important advantage of the .NET framework is its support for various programming languages. This means that developers can choose a language that fits their needs and expertise, yet are able to use the same set of libraries and tools provided by the framework.

Another advantage of the .NET framework is its support for different application types. The framework includes libraries and tools for building desktop, web, mobile, and gaming applications, making it a versatile choice for developers working on a variety of projects.

The .Net Framework is a technology that supports the creation and running of Windows apps and web services.

- Provide a code-implementation environment that :
  - o Reduces software deployment and version conflict.
  - o Promotes secure implementation of code with code generated by unknown or semi-reliable third party.
  - o Eliminates performance problems in scripted or defined environments.
- Streamline the developer experience across a wide variety of apps, such as Windows-based apps and web-based apps.
- Create all communications on industry standards to ensure that code based on the .NET framework integrates with any other code.

The .NET framework includes common language runtime (CLR) and .NET framework class libraries. Common language runtime is the base of the .NET framework. Think of runtime as an agent who manages code during implementation, provides basic services such as memory management, thread management, and remoteting, as well as implementing rigorous types of security and other forms of code accuracy that promote security and robustness.

The main advantage of the .NET framework is interoperability in different languages. Since all Microsoft .NET languages share the same runtime language, they all work well together.

What is Framework :

The framework is a software. Or you could say that the framework is a collection of many small technologies put together to develop applications that can be implemented anywhere.

What is .Net Framework :

The .NET framework is a pure object oriented, which is similar to the Java language. But there is no separate platform like Java. So, its applications only run on the Windows platform.

The main purpose of this framework is to develop applications that can run on the Windows platform. The current version of the .NET framework is 4.8.

Origin of .Net Technology :

Microsoft has been renaming the .NET implementation, but I think they've finally become something better. In this blog post, we'll try to clear up some of the confusion between naming and version.

The first .NET Framework version, launched in 2002, was born. From approximately 2002-2016, the .NET Framework was the Microsoft framework of choice for all types of development. The .NET framework versions range from 1.0 to 4.8, and 4.8 will be the last version so far. The .NET framework is limited to the Windows operating system, and while it's not under active development, there are still many business-critical applications that use it today. Since 2016, Microsoft has decided to move to open-source and cross-platforms, called .NET Core.

The split between the .NET framework and .net core will be a major problem for code sharing and recycling. This is where the .NET standard fits in. The .NET standard is a specification for a set of .NET APIs that are available for both the .NET framework and .NET core. The .NET standard Nuget package is a popular choice for authors as it can be referenced by a wide range of .NET implementations. It is a powerful tool for migrating from the .NET framework to the .NET core.

Evolution of .Net :

Microsoft released further versions of the .NET framework, each bringing improvements and upgrades to empower developers.

In 2002, the .NET Framework came to the 1.0 scene and set the stage for future growth. Common Language Runtime (CLR), which provided a manageable implementation environment for running code and provided a strong foundation with features such as a framework class library (FCL), reusable classes, and a vast collection of components.

With the release of .NET Framework 2.0 in 2005, creation of ASP.NET 2.0, a powerful web application framework.

Rise of .NET Framework and Web Development :

Released in 2006, .NET Framework 3.0 made several important additions, including the Windows Presentation Foundation (WPF), technology for creating rich and interactive user interfaces, and the Windows Communication Foundation (WCF), a framework for building distributed applications.

In 2008, .NET Framework 3.5 expanded these capabilities, introducing language integrated query (LINQ), a powerful query language that enabled developers to investigate a variety of data sources using coherent syntax.

The .NET Framework 4.0, released in 2010, focused on enhancing efficiency, scalability, and parallel programming. Features such as task parallel library (TPL) and improved support for dynamic language were introduced.

.NET Core and the Cross-Platform Revolution :

In 2016, Microsoft underwent a game-changing change in its strategy with the announcement of .NET Core. This cross-platform and open-source framework was designed to address the growing demand for building applications that can run on Windows, macOS, and Linux.

Released the same year, .NET Core 1.0 offered a modular and lightweight framework, optimized for high-performance conditions. This provided developers with the flexibility to select and incorporate only the necessary components for their applications, resulting in a leaner and more efficient deployment.

Then with the release of .NET Core, Microsoft continued to refine and expand its capabilities. .Net Core 2.0 brought improvements to performance and security, while .Net Core 3.0 introduced support for desktop application development and introduced the much-hyped Windows Presentation Foundation (WPF) and Windows Forms Framework in the cross-platform world.

Features of .NET framework :

1. CTS
2. CLS
3. CLR
4. MSIL
5. IL
6. JIT

1. **CTS** : It specifies a standard that shows what type of data and value can be defined and managed in computer memory at runtime. The CTS ensures that defined programming data in different languages communicate with each other to share information. For

example, in C# we define the data type as INT, while in VB.NET we define the integer data type.

2. **CLS** : It is responsible for transforming various .NET programming language syntactic rules and regulations into CLR understandable formats. Basically, it provides language interoperability. Language interoperability means providing implementation support to other programming languages in the .NET framework as well.
3. **CLR** : CLR is the heart of the .NET framework. It interface between the framework and the operating system. Is a key component within the .NET Framework that is responsible for converting the MSIL (Microsoft Intermediate Language) code into the original code and then provides a runtime environment for implementing the code. This means common language runtime (CLR) is an implementation engine that handles running applications. These include exceptional handling, memory management, and waste collection. Moreover, it provides security, type-security, interoperability, and portability. Basically, it is responsible for managing the implementation of .NET programs regardless of any .NET programming language.
4. **MSIL** : MSIL, or simply IL, is a instruction set in which all .NET programs are compiled. It is similar to the assembly language and contains instructions for loading, storage, starting, and calling methods. When we compiled a C# program or any program written in CLS compliance language, the source code is compiled into MSIL.

   To build programs, the.NET framework comes with individual compilers for various programming languages, such as C#.

   After compiling the source code, the each.NET compiler creates a central code, and each environment uses this intermediate code known as the Microsoft Intermediate Language (MSIL).

   The MSIL machine is agnostic and can be quickly converted to the original code. CLR includes a Just-in-Time (JIT) compiler, which converts the MSIL code to the original machine code. As a result, MSIL needs to be translated by the JIT compiler before it can be implemented on the CPU. For each supported machine architecture, a JIT compiler is accessible. Any other supported machine will run the same MSIL.

   This is usually a set of platform-independent instructions generated from source code by a language-specific compiler.
5. **IL :** Intermediate Language (IL) is an object-oriented programming language designed to be used by collection for the .NET framework before static or dynamic compilation in machine code. IL is used as the output of the compilation of source code written in any .NET programming language to create machine-independent code through the .NET framework.

6. **JIT :** JIT compiles the IL code into machine code before implementation and then saves the transaction to memory. Compilers are platforms used to translate source code into machine-understandable language. In the .NET framework, Common Language Runtime provides a compiler that converts the source code to the MSIL- Microsoft Intermediate Language.

   However, MSIL cannot be used in the machine. It must then be translated into machine-understandable code at runtime. This is done by .NET's JIT compiler which converts to CPU-specific code.

   JIT streamlines the process of built-in collection. The first step is platform-dependent as it converts the source code to MSIL. However, the second step is simple because each platform has a JIT machine that translates MSIL into the original code.

   JIT has to manage the implementation of various programming languages that come under .NET technology. Input files for JIT collectors contain .dll or .exe that are MSIL files. Once these files are implemented, the JIT compiler will begin its implementation.

: QUESTIONS :

1. What is .NET?
2. What is .NET Framework and what are the main components of it?
3. What is an IL?
4. Features of .Net framework?
5. What is MSIL?

***

2. Chapter

Introduction to C#.Net

S y l l a b u s :

Introduction to C #, Advantages & Disadvantages of C#, Programming Structure of C#, Basic Constructs – Variables, Data types, Operators, arrays, functions, Control Statements (if statement, if....else statement, nesting of if....else statement, the else if ladder, switch statement), Looping Construct(while statement, do statement, for statement)t)

(10 L, 15 M)

Introduction to C# :

C# is pronounced "C-sharp". C# is an object-oriented programming language provided by Microsoft that runs on the .NET framework.

Anders Heiselberg is the founder of the C# language.

It is based on C++ and Java, but there are many additional extensions used to perform a component-oriented programming approach.

C# has evolved a lot since their first release in 2002. It was introduced with the .NET framework 1.0, and the current version of C# is 5.0.

Advantages & Disadvantages of C# :

Advantages of C# :

- As mentioned, C#, since its inception, was intended to be an object-oriented programming language.
- In OOP, programmers can easily define the structure of data and classify them into objects, which are generated from classes.
- This group of data makes it easier to develop applications, maintain applications, and recycle code. This makes the code easier to correct and reduces the chances of errors.
- C# is cross-platform programming language is.
- If you build an application in C#, it can run on any operating system or platform, including Apple, iOS, Windows, Android, or in the cloud.

- C# is making it easy to write and maintain complex software.
- With features like inheritance and polymorphism, you can create efficient and beautiful code that's easy to read and modify.
- Collection and implementation time is very fast
- Scale is automatic and it is up to date
- It is safer than C and C++ as pointer types are not allowed
- Provides type security
- It's ideal for development in Windows

Disadvantages of C# :

- C# can be a challenging language to learn, especially for developers who are new to programming.
- With its advanced features and syntax, it can take time to master C#.
- This can make C# less suitable for the creation of applications that need to be run on older or less powerful devices.
- C++ is not so flexible
- It may have an intense learning curve
- Some errors can be complex in nature, so adequate knowledge and experience are required
- The application performance does not give the best performance in terms of benchmarks

Programming Structure of C# :

```
using System;
namespace MyFirstProgram
  {
    class Problem
      {
        static void Main(String[ ] args)
      {
            /* My first Program */
            Console.WriteLine("Welcome to World");
            Console.ReadKey( );
      }
     }
  }
```

A C# Program consists of the following part-

i. **Importing Namespace Section** : This section contains BCL (base class libraries) as well as import statements used to import user-defined namespaces if necessary. This is similar to the statements included in the C programming language. Let's say if you want to use some classes and interfaces in your code, you'll need to include the namespace(S) from where these classes and interfaces are defined.

The keyword used to include system namespace in the program is used. The program typically contains multiple usable statements.

For example, if you're going to use a console class in your code, you'll need to include a system namespace because the console class belongs to the system namespace.

Syntax :

```
using NamesapceName;
```

Example :

```
using System;
```

ii. **Namespace declaration** : Here, the user-defined namespace is declared. All project-related classes and interfaces or any kind of announcement in .NET applications should be made in some namespace. Normally we put all the relevant classes under the same namespace, and in a project, we can create multiple namespaces.

The next line has a namespace declaration. Namespace is a collection of classes.

Syntax :

```
namespace NamespaceName
{
. . . . .
}
```

Example:

```
namespace MyFirstProgram
{
. . . . .
}
```

iii. **Class Declaration Section** : For every desktop application in NET, we need a initiate class file. For example, for every .NET desktop application, such as Console and Windows, there should be a start-up class that should have the main method from where program implementation is going to begin. When we create a console application using Visual Studio, by default, Visual Studio will create a start-up class file called a program.cs which will have a class of classes called programs that contain the main method. A start-up class is a class that consists of a main () system from which program implementation is to begin.

The next line contains class announcements, class Helloworld contains definitions of the data and methods your program uses. Classes typically have several methods. Methods define class behavior. However, there is only one method main in the Helloworld class.

Syntax :

```
class ClassName
{
. . . . .
}
```

Example :

```
class Problem
{
. . . . .
}
```

iv. **Main Method Section** : The main () method is the entry point or starting point to start the implementation of the application. When the application starts to implement, the main method will be the first block of the application to be implemented. The main method consists of the main logic of the application.

It tells us what the class does when the main method is implemented. Within the class, you can have multiple methods and variable names can be announced. Constant/zero is a return value, which will be explained in a short time.

Syntax :

```
static void Main(String[ ] args)
{
 . . . . .
}
```

Example :

```
static void Main(String[ ] args)
{
 . . . . .
}
```

v. **Statements and Expressions** :

The Console.WriteLine () is a predefined method used to display the value of any string or variable in a C# program.

Console.ReadKey is another predefined method used by a program to wait for any key press.

Syntax

Console.WriteLine("Write a message");
Console.ReadKey( );

Example :

Console.WriteLine("Welcome to World");
Console.ReadKey( );

vi. **Comments** :

The next line is ignored by the /*. . . . .*/ compiler and is added to the program to add comments.

Syntax

/* My first Program */

Example:

/* My first Program */

Basic Constructs :

Variables :

Variable is the name of a memory location. It is used to store data. Its value can be changed and it can be reused several times.

This is a way to show memory location through a symbol so that it can be easily recognized.

Variable is the name given to the storage area that our programs can handle. Each variable in C# has a special type, which determines the size and layout of the variable's memory, the range of values that can be stored in that memory, and the set of operations that can be request to the variable.

Classification of basic value types provided in C# :

Variable Type	Example
Integral Type	Int, Char, Byte, Short, Long
Boolean Type	True or False
Decimal Type	Decimal
Floating Point Type	Float and Double
Nullable Type	Nullable Data Type

Syntax :

<Data Type> <Variable list>;

Example :

```
int A,B,C;
char ST;
float Percentage;
double pi;
```

Here, A, B, C, ST, Percentage, pi are variables and int, char, float, double are data types.

We can also provide values when declaring variables as follows :

```
int A=2, B=4, C=6; //Declaring 3 variables of Integer type
char ST='String';
float Percentage=70.5;
double pi=3.1415;
```

Data types :

Let us now understand what are the different data types available in .NET and under what circumstances which data type is appropriate in C#. Why I'm going to focus on this is that most of the time .NET use a limited data type. Look, you'll see that most of the time as a .NET developer, we're familiar with using INT, bull, double, string, and datetime data types. These five data types are primarily used by .NET. Due to the limited use of data types, we lose out in terms of optimization and efficiency.

1. Value Data Type
2. Reference Data Type
3. Pointer Data Type

1. **Value Data Type :**

   The data type that stores value directly in memory is called the value data type in C#.

   The Value data type are integer based, floating based and Boolean etc.

   The value data types in C# are again classified into two types which are as follows:

   i. Predefined Data Type- Integer, Float, Boolean etc.

   ii. User-defined Data Type- Structure, Enumerations etc.

2. **Reference Data Type :**

   Reference data types do not contain real data stored in variables, but they do refer to variables.

   If the data is replaced by one variable, the other variable automatically reflects this change in value.

   The Reference data types in C# are again classified into two types which are as follows :

   i. Predefined Data Type- String and Object

   ii. User-defined Data Type- Interface and Class

3. **Pointer Data Type** :

A pointer in the C# is a variable, also known as a locator or indicator that points to the address of the value.

A C# pointer is a variable that holds another type of memory address. But the C# pointer can only be declared to keep the memory address of the value types and arrays.

The Pointer data types in C# are again classified into two types which are as follows.

i. &(ampersand sign) Address Operator means determine the address of a variable

ii. *(asterisk sign) Indirection Operator means access the value of an address

Operators :

Operator is just a symbol that is used to perform an operation. There can be several types of operations such as arithmetic, logic, bitwise, etc.

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Assignment Operators
5. Unary Operators

1. **Arithmetic Operator :**

Arithmetic operators in C# are used to perform arithmetic/mathematical operations such as addition, subtraction, multiplication, division, etc. on the operand.

The following operators fall into this category :

Operator	**Operator Name**
+	Addition Operator
-	Subtraction Operator
*	Multiplication Operator
/	Division Operator
%	Modulo Operator

Example:

```
using System;
namespace ArithmeticOperatorDemo
{
  class Arithmeticprogram
  {
   Static void Main(String[ ] args)
   {
    int Performance ;
    int No1=40, No2=20;
    //Addition Operator
     Performance=(No1 + No2);
   Console.WriteLine($"Addition Operator: {Performance}");
   // Subtraction Operator
   Performance = (No1- No2);
   Console.WriteLine($"Subtraction Operator: {Performance}");
   // Multiplication Operator
   Performance = (No1* No2);
   Console.WriteLine($"Multiplication Operator: {Performance}");
   // Division Operator
   Performance = (No1/ No2);
   Console.WriteLine($"Division Operator: {Performance}");
   // Modulo Operator
   Performance = (No1% No2);
   Console.WriteLine($"Modulo Operator: {Performance}");
   Console.ReadKey();
   }
  }
}
```

Output:

```
Addition Operator: 60
Subtraction Operator: 20
Multiplication Operator: 80
Division Operator: 2
Modulo Operator: 0
```

2. **Relational Operators :**

   Relational operators in C# are also a called as comparison operators. This determines the relationship between the two operands and returns the Boolean effect after comparison, i.e., true or false. The different types of relational operators suggested by C# are as follows.

The following operators fall into this category :

Operator	**Operator Name**
==	Equal To
<	Less Than
>	Greater Than
<=	Less Than Or Equal To
>=	Greater Than Or Equal To
!=	Not Equal To

Example :

```
using System;
namespace RelationalOperatorDemo
{
 class RelationalProgram
 {
 static void Main(String[ ] args)
 {
 bool Performance;
 int No1=40, No2=20;
 //Equal to Operator
 Performance=(No1==No2);
 Console.WriteLine("Equal (=) to Operator: " + Performance);
 // Greater than Operator
 Performance = (No1> No2);
 Console.WriteLine("Greater (<) than Operator: " + Performance);
 // Less than Operator
 Performance = (No1< No2);
 Console.WriteLine("Less than (>) Operator: " + Performance);
 // Greater than Equal to Operator
 Performance = (No1>= No2);
 Console.WriteLine("Greater than or Equal to (>=) Operator: " + Performance);
 // Less than Equal to Operator
 Performance = (No1<= No2);
 Console.WriteLine("Lesser than or Equal to (<=) Operator: " + Performance);
 // Not Equal To Operator
 Performance = (No1!= No2);
 Console.WriteLine("Not Equal to (!=) Operator: " + Performance);
 Console.ReadKey();
 }
}
}
```

Output:

```
Equal (=) to Operator: False
Greater (<) than Operator: True
Less than (>) Operator: False
Greater than or Equal to (>=) Operator: True
Lesser than or Equal to (<=) Operator: False
Not Equal to (!=) Operator: True
```

3. **Logical Operators :**

Logical operators are mainly used in conditional statements and loops to evaluate a situation. These operators will work with Boolean Expression.

The following operators fall into this category:

Operator	**Operator Name**
||	OR
&&	AND
!	NOT

Example:

```
using System;
namespace LogicalOperatorDemo
{
 class LogicalProgram
 {
 static void Main(String[ ] args)
 {
 bool P=true, Q=false, R;
 //Logical AND operator
 R = P && Q;
 Console.WriteLine("Logical AND Operator (&&) : " + R);
 //Logical OR operator
 R = P || Q;
 Console.WriteLine("Logical OR Operator (||) : " + R);
 //Logical NOT operator
 R = !P;
 Console.WriteLine("Logical NOT Operator (!) : " + R);
 Console.ReadKey();
 }
 }
}
```

Output :

```
Logical AND Operator (&&) : false
Logical OR Operator (||) : true
Logical NOT Operator (!) : false
```

4. **Assignment Operators :**

Assignment operators in C# are used to provide value to the variable. The left-sided operand of the assignment operator is a variable, and the right-sided operand of the assignment operator may be a value or expression that must return some value and that value will be assigned to the left-sided variable.

The most important point that you need to keep in mind is that the value on the right side must be of the same data type as the left side variable otherwise you will get a complier time error.

The following operators fall into this category :

Operator	**Operator Name**
=	Simple Assignment
+=	Addition Assignment
-=	Subtraction Assignment
*=	Multiplication Assignment
/=	Division Assignment
%=	Modulus Assignment

Example:

```
using System;
namespace AssignmentOperatorsDemo
{
class AssignmentProgram
{
static void Main(string[] args)
{
// Initialize variable p using Simple Assignment Operator "="
int p = 15;
p += 10; //It means p = p + 10 i.e. 15 + 10 = 25
Console.WriteLine($"Add Assignment Operator: {x}");
// initialize variable p again
p = 20;
p -= 5; //It means p = p - 5 i.e. 20 - 5 = 15
Console.WriteLine($"Subtract Assignment Operator: {x}");
// initialize variable p again
p = 15;
p *= 5; //It means p = p * 5 i.e. 15 * 5 = 75
Console.WriteLine($"Multiply Assignment Operator: {p}");
// initialize variable p again
p = 25;
p /= 5; //It means p = p / 5 i.e. 25 / 5 = 5
Console.WriteLine($"Division Assignment Operator: {p}");
// initialize variable p again
p = 25;
p %= 5; //It means p = p % 5 i.e. 25 % 5 = 0
Console.WriteLine($"Modulo Assignment Operator: {p}");
Console.ReadKey();
}
}
}
```

Output:

```
Add Assignment Operator: 25
Subtract Assignment Operator: 15
Multiply Assignment Operator: 75
Division Assignment Operator: 5
Modulo Assignment Operator: 0
```

5. **Unary Operators :**

The unitary operators in C# only need one operator. They are used to Increment (increase) or Decrement (decrease) value.

The following operators fall into this category:

Operator	**Operator Name**
+	Unary Plus
-	Unary Minus
++	Increment
--	Decrement
!	Logical Negative (Not)

Example:

```
using System;
namespace UnaryOperatorsDemo
{
class UnaryProgram
{
static void Main(string[] args)
{
int num = 5, result;
bool flag = true;
result = +num;
Console.WriteLine("+num = " + result);
result = -num;
Console.WriteLine("-num = " + result);
result = ++num;
Console.WriteLine("++num = " + result);
result = --numb;
Console.WriteLine("--num = " + result);
Console.WriteLine("!flag = " + (!flag));
}
}
}
```

Output:

```
+num = 5
-num = -5
++num = 6
--num = 5
!flag = false
```

Array :

An array is a type of sequential data structure, used to store collections of the same type of objects.

An array is defined as a collection of similar data elements. If you have some sets of integers and some sets of floats, you can classify them by a name as an array. So, in simple terms, we can define an array as a collection of similar types of values that are stored in consecutive memory locations under the same name.

We have already learned that variables are used to store values. But the variable can hold the same value of a particular data type at any time.

Array Representation :

Element : Every item stored in the array is called an element.

Index : There is a numerical index at each point in the array, which is used to identify the element.

The array can be declared differently in different languages.

Syntax :

Datatype[ ] arrayName;

As in the example above, the following important points should be considered:

- The index starts with 0.
- The array length is 10, which means it can store 10 elements.
- Each component can be accessed through its index.
- Datatypes are used to specify the type of elements in the array.
- [ ] Specifies the status of the array. Rank specifies the size of the array.
- The name of the array specifies the name of the array.

Types of Array :

There are two types of Array :

1. Single Dimensional Array
2. Multi Dimensional Array

1. Single Dimensional Array :

It represents each element by a single subscript. These components are stored in successive memory locations.

Syntax :

Datatype[ ] arrayName;

Example :

```
using System.Text;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int[] arr=new int[3];
            arr[0]=10;
            arr[1]=20;
            arr[2]=30;
            Console.WriteLine("Length of an Array:" +arr.Length);
            Console.WriteLine("Zero Index value:" +arr[0]);
            Console.WriteLine("First Index value:"+arr[1]);
            Console.WriteLine("Second Index value:" +arr[2]);
            Console.ReadLine();

        }
    }
}
```

Output :

```
Length of an Array:3
Zero Index value:10
First Index value:20
Second Index value:30
```

2. **Multi Dimensional Array :**
Multi Dimensional Array also know as Matrix array. Data store in tabular form (row * column). A multi-dimensional array can be declared by adding a comma to a square bracket [,].

Syntax :

Datatype[,] arrayName;

Example :

```
using System.Text;
using System.Threading.Tasks;
namespace MultiArray
{
    class MultiArrProgram
    {
        static void Main(string[] args)
        {
        int[,] arr=new int[3,3];
        arr[0,1]=50;
```

```
            arr[1,2]=60;
            arr[2,0]=70;
            for (int i = 0; i < 3; i++)
            {
                for (int j = 0; j < 3; j++)
                {
                    Console.Write(arr[i, j] + " ");
                }
                Console.WriteLine();
                   }
            Console.ReadLine();
            }
        }
    }
```

Output :

```
0   50 0
0   0   60
70 0   0
```

Function :

A function allows you to include a piece of code and make calls from other parts of your code. You can soon run into situations where you need to repeat a piece of code from multiple locations, and that's where the functions come in.

Syntax :

```
<Access specifier> <return type> <function name>(<Parameters>)
{
//Function Code
//Return Statement
}
```

Name of function : This is a unique name that is used to call a function.

Return Type : This is used to specify the data type of function return value.

Body : This is a block that contains functional statements.

Access Specifier : It is used to specify function availability in the application.

Parameters : This is a list of arguments that we can give to the function during the call.

Types of Function :

1. Bulit In Function
2. User Defined Function

1. **Bulit In Function :** The most important reason you use library functions or built-in functions is because they work. These Bulit In functions, or pre-defined functions, have already gone through multiple test stages and are easy to use. Built-in functions are promoted for performance. So, you'll get superior performance with built-in functions. Since functions are "standard library" functions, a dedicated group of developers is continually working on them to make them superior. This saves development time. Common functions such as printing on a screen, calculating a square root, and more have already been written. You shouldn't have to worry about rebuilding them. You require to use them and save your time.
2. **User Defined Function :** User-defined functions that have been produce by the programmer so that he/she can use them several times. This reduces the complexity of large programs and optimizes the code. C# allows you to define tasks according to your needs. A function whose body is executed by a developer or user is called a user-defined function. Depending on the needs of the client or project, the tasks we are developing are called user-defined functions. Always user-defined functions are customer-specific functions or project-specific functions. As programmers, we have full control of user-defined tasks. As a programmer, since coding parts are available, it is possible to change or modify the behaviour of a function defined by any user if necessary.

   Example :

```
using System;
namespace MultiArray
{
  class MultiArrProgram
  {
    public void clg(int a, string c)
    {
      Console.WriteLine("Student Roll : "+a);
      Console.WriteLine("Student Name : " + c);
    }
    static void Main(string[] args)
    {
      MultiArrProgram sd=new MultiArrProgram();
      sd.clg(101,"Ram");

    Console.ReadLine();
    }
  }
}
```

Output :

```
Student Roll : 101
Student Name : Ram
```

Type of Control Statements

1) if statement
2) if....else statement
3) nesting of if....else statement
4) the else if ladder
5) switch statement

1) if statement

If a condition is true, use an IF statement to specify a block of code to be implemented.

Syntax

```
if (condition)
{
  //If the condition is true, the block of code to be implemented
}
```

Example

```
using System;
namespace IfStatment
{
    class IfStatProg
    {

        static void Main(string[] args)
        {
       int number = 10;
            if (number % 2 == 0)
            {
                Console.WriteLine("It is even number");
            }
        }

    }
}
```

Output

It is even number

2) **if....else statement**

The statement also tests the status. If the condition is true, it implements the block otherwise else block is executed.

Syntax

```
if (condition)
{
  //If the condition is true, the block of code to be implemented
}
Else
{
//If the condition is false
}
```

Example

```
using System;
namespace IfElseStatment
{
    class IfElseStatProg
    {
        static void Main(string[] args)
        {
            Console.Write("Enter a Number : ");
            int c = Convert.ToInt32(Console.ReadLine());
            if (c >= 0)
            {
                Console.WriteLine("Positive Number");
            }
            else
            {
                Console.WriteLine("Negative Number");
            }
            Console.ReadLine();
        }

    }
}
```

Output:

```
Enter a Number : 12
Positive Number
```

3) nesting of if....else statement

If the statement inside a statement is known as a nested IF. If the statement in this case is the target of another, then the statement otherwise. When more than one condition must be true, and one of them is a sub-condition of the parent condition, it can be used.

Syntax

```
if (Condition1)
        {
             // code to be implemented
             // if Condition2 is true
             if (Condition2)
             {
                 // code to be implemented
                 // if Condition2 is true
             }
        }
```

Example

```
using System;
namespace NestedStatment
{
class NestStatProg
 {
    public static void Main(String[] args)
    {
        int i = 20;
        if (i == 20)
       {
           // Nested - if statement
            // Will only be implemented if statement
            // above it is true
            if (i < 25)
                Console.WriteLine("i is smaller than 25 too");
            else
                Console.WriteLine("i is greater than 30");
        }
    }
}
```

Output

```
i is smaller than 12 too
```

4) the else if ladder

The if-else-if ladder statement implements one of the multiple statements.

Syntax

```
if(Condition1)
{
//code to be implemented if condition1 is true
}
else if(ondition2)
{
//code to be implemented if condition2 is true
}
else if(condition3)
{
//code to be implemented if condition3 is true
}
...

Else
{
//code to be implemented if all the conditions are false
}
```

Example

```
using System;

namespace LadderStat
{
    class LadderProgram
    {

        static void Main(string[] args)
        {
            Console.Write("Enter a Marks : ");
            int c = Convert.ToInt32(Console.ReadLine());
        if (c >= 80 && c<=100)
        {
            Console.WriteLine("A Grade");
        }
        else if(c>=60 && c<=79)
        {
            Console.WriteLine("B Grade");
        }
        else if (c >= 35 && c <= 59)
        {
            Console.WriteLine("C Grade");
        }
        else
        {
            Console.WriteLine("Fail");
        }
        Console.ReadLine();
    }

  }
}
```

Output:

```
Enter a Marks : 89
A Grade
```

6) switch statement

The switch statement is an alternative to the ladder if else if long. The expression is tested for different cases and a match is implemented. A brake statement is used to exit the switch. If the brake is not used, the control will flow in all cases below it until the brake is obtained or the switch ends. The default case is (optional) at the end of the switch, if none of the cases match, the default case is executed.

Syntax

```
switch (expression)
 {
Case 1: // statement sequence
  break;
Case 2: // statement sequence
 break;
.
.
.
Case N: // statement sequence
             break;
default: // default statement sequence
}
```

Example

```
using System;
 namespace SwitchStat

class SwitchProg
{
    public static void Main(String[] args)
    {
        int number = 25;
        switch(number)
        {
        case 5: Console.WriteLine("case 5");
                 break;
        case 15: Console.WriteLine("case 15");
                 break;
        case 25: Console.WriteLine("case 25");
                 break;
        default: Console.WriteLine("None matches");
                 break;
        }
    }
}
```

Output

```
case 25
```

Looping Construct

1) **while statement**
2) **do statement**
3) **for statement**

1) **while statement**

The while loop loops through a block of code as long as a specified state is True.

Syntax

```
while(condition)
{
//code to be implemented
}
```

Example

```
using System;
namespace WhileStat
{
   class WhileStatProg
   {

      static void Main(string[] args)
      {
         char i = 'A';
         while (i <= 'Z')
         {
            Console.Write( i+"\t");
            i++;
         }

         Console.ReadLine();
      }

   }
}
```

Output:

```
A    B    C    D    E    F    G    H    I    J    K    L    M    N    O
     P    Q    R    S    T    U    V    W    X    Y    Z
```

2) **do statement**

The do/while loop is a alternative of the while loop. This loop will implement the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax

```
Do
{
//code to be implemented
}
while(condition);
```

Example

```
using System;
namespace DoWhileStat
{
   class SoWhileProg
   {

      static void Main(string[] args)
      {
         int j = 5;
         do
         {
            Console.WriteLine(j);
            j--;
         } while (j >= 1);

         Console.ReadLine();
      }

   }
}
```

Output

```
5
4
3
2
1
```

3) for statement

The for loop is used to create a part of the program several times.

Syntax

```
    for(Initialization; Condition; Increment/Decrement)
    {
    //code to be Implemented
    }
    Example
using System;
namespace ForLoopStat
{
    class ForLoopProg
    {
        static void Main(string[] args)
        {
            Console.Write("Enter a Number  : ");
            int j = Convert.ToInt32(Console.ReadLine());
            for (int i = 1; i <= 10; i++)
            {
                Console.WriteLine(i*j);
            }
            Console.ReadLine();
        }

    }
}
```

Output

```
Enter a Number : 2
2
4
6
8
10
12
14
16
18
20
```

: QUESTIONS :

1. What is C# ?
2. What is array ?
3. Advantages of C# ?
4. Programming Structure of C# ?
5. Explain Data Types in C#?

***

3. Chapter

Introduction to ASP.NET web

S y l l a b u s :

History of Asp.Net, Introduction to Asp.Net, Features of Asp.Net, Structure of Asp.Net Page, ASP.NET Web Pages Model (Single Page Model, Two Page Model).)

(10 L, 20 M)

History of ASP .NET :

In the beginning state of the internet boom, the internet was more or less collection of static linked web pages. However, with the growing popularity of the internet, trying to manually keep all the sides up-to-date soon proved to be unmanageable. Microsoft addressed this need by introducing Active Server Pages (ASP). This simple but powerful scripting language for Microsoft's Internal Information Server (IIS) allowed the creation of dynamic web pages.

Like any other technology, ASP too had to meet a number of challenges. To list a few :

- Coding Overhead is too much. Everything requires writing code.
- Maintaining page state requires more code.
- Reusability is less as HTML and scripting code is merged.
- Does not support many type of browser. For example, certain controls do not work with Netscape. Marquee feature also does not work in Netscape.
- Session state scalability and availability is less.
- Limited support for caching, tracing, debugging and similar issues.

To overcome the drawback of ASP, Microsoft finally introduced ASP.NET as a web application framework enabling programmer to build dynamic web sites, web application and web services.

ASP.NET (Active Server Pages .NET) is a web development framework developed by Microsoft. It has undergone several significant changes and updates over the years. Here's a brief history of ASP.NET :

1. **ASP.NET 1.0 (January 2002)** : The first version of ASP.NET was released as part of the Microsoft .NET Framework. It introduced the concept of code-behind files, separating the HTML markup from the server-side code. It also included Web Forms, a powerful event-driven programming model for building web applications.
2. **ASP.NET 1.1 (April 2003)** : This release included various improvements and bug fixes. It continued to build on the foundation laid by ASP.NET 1.0.
3. **ASP.NET 2.0 (November 2005)** : A major release with significant enhancements, including master pages, themes, and a new membership system for authentication and authorization. ASP.NET 2.0 also introduced a new set of controls, such as the GridView and DetailsView, which simplified data binding.
4. **ASP.NET 3.5 (November 2007)** : This version expanded on ASP.NET 2.0 by introducing the ASP.NET AJAX framework, making it easier to build dynamic and responsive web applications. It also included new controls and features like the ListView control.
5. **ASP.NET 4.0 (April 2010)** : ASP.NET 4.0 brought improvements in web forms, enhancements to the ASP.NET MVC framework (introduced in the previous version), and better support for parallel programming.
6. **ASP.NET 4.5 (August 2012)** : This version came with several enhancements, including support for asynchronous programming using the 'async' and 'await' keywords. It also introduced the One ASP.NET project template, which aimed to unify the various ASP.NET project types.
7. **ASP.NET 5.0 (now ASP.NET Core 1.0 - June 2016)** : This was a significant departure from the previous versions, introducing a cross-platform, open-source framework known as ASP.NET Core. ASP.NET Core allowed developers to build and run applications on Windows, Linux, and macOS. It was modular, lightweight, and designed for modern development practices. It also embraced modern web standards and was decoupled from the traditional System. Web infrastructure.
8. **ASP.NET Core 2.0 (August 2017)** : This release built upon the foundation of ASP.NET Core 1.0, introducing new features, improvements, and performance enhancements.
9. **ASP.NET Core 3.0 (September 2019)** : More features and improvements were introduced, including endpoint routing, gRPC support, and improved JSON APIs.
10. **ASP.NET Core 5.0 (November 2020)** : This version continued to build on ASP.NET Core 3.1, unifying the naming and versions of ASP.NET Core, Entity Framework Core, and Azure SDK. It also brought performance improvements and new features.
11. **ASP.NET 6 (November 2021)** : ASP.NET 6 is the latest major version, introducing a unified framework for building web, cloud, desktop, mobile, gaming, IoT, and AI

applications. It includes features like minimal APIs, Blazor improvements, and enhanced support for cloud-native applications.

Introduction to ASP.NET :

ASP.NET is a server side technology for web development. It is provided for Microsoft. ASP.NET was first released in the year 2002. The full form of ASP is Active Server Page and .NET is Network enable technology. ASP.NET is a technology or application framework from Microsoft for developing web applications to produced dynamic web page using Microsoft .NET Framework.

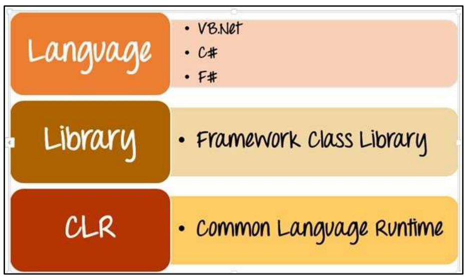
ASP.NET is not a programming language, it is a web technology or server side technology means all ASP.NET controls are executed on server side. By using ASP.NET user can design a websites or web pages. ASP.Net applications can also be written in a variety of .Net languages. These include C#, VB.Net, and J# etc. ASP.NET provides services to allow the creation, development and execution of web application and web services.

Few terms in ASP.NET are as follows :

1. **Web Page** : Information towards the end user or client is nothing but web page.
2. **Websites** : It is a collection of web pages or other resources.
3. **Web Browser** : To open any web page or website compulsory one software is required in our system that is called web browser. It is software which manages different types of web pages.
4. **Web Server** : Web server is a server which is use to provide web responds when we send web request to the server. The default web server in .NET is ASP.NET web development server (Webdev). IIS is not a default web server because we have to installed explicitly this server. When you installed .NET framework or Visual Studio .NET automatically Webdev server is installed.

ASP.NET Architecture and its Components :

ASP.Net is a framework which is used to develop a Web-based application. The basic architecture of the ASP.Net framework is as shown below.



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

1. **Language** : A variety of languages exists for .net framework. They are VB.net and C#. These can be used to develop web applications.
2. **Library** : The .NET Framework includes a set of standard class libraries. The most common library used for web applications in .net is the Web library. The web library has all the necessary components used to develop.Net web-based applications.
3. **Common Language Runtime** : The Common Language Infrastructure or CLI is a platform. .Net programs are executed on this platform. The CLR is used for performing key activities. Activities include Exception handling and Garbage collection.

Features of ASP.NET :

The features of ASP.NET are as follows :

1. **Integrated Development Environment (IDE)** : ASP.NET is tightly integrated with Visual Studio, providing a powerful IDE for designing, coding, testing, and debugging web applications.
2. **Code Behind Model** : This is the separation of code from the HTML documents and therefore from the look-and-feel HTML language. This makes it much easier for teams of programmers and designers to collaborate efficiently. This is the concept of separation of design and code. By making this separation, it becomes easier to maintain the ASP.Net application. The general file type of an ASP.Net file is aspx. Assume we have a web page called MyPage.aspx. There will be another file called MyPage.aspx.cs which would denote the code part of the page.
3. **Compiler Assemblies** : ASP.NET is a compiler base technology. The code is compile by JIT compiler into native machine code and does not need to interpreted every time a web page is requested.
4. **Choice of Languages** : With ASP.NET, you can write programs in any supported .Net Programming language like C#, VB.NET, J#, etc.
5. **Event-driven programming Model** : ASP.NET Web Forms uses an event-driven programming model, where server-side events are triggered in response to user actions on the client side. This simplifies the handling of user input and interaction.
6. **Web Forms** : The Framework Class Library includes the Sysem.Web.UI namespace, which comes with a powerful set of prewritten functionality, such as calendar and time controls. And unlike the old ASP, these new Web forms are browser-independent.
7. **State management** : ASP.NET provides various options for managing state, including view state, session state, and application state. These mechanisms help in preserving data between page requests.

8. **Server Controls** : ASP.NET includes a rich set of server controls, such as buttons, textboxes, and grids, that abstract complex HTML and client-side scripting, making it easier to build interactive web applications.
9. **Web API** : ASP.NET Web API allows developers to build HTTP services that can be consumed by clients, including web browsers and mobile devices. It is well-suited for building RESTful APIs.
10. **Authentication and Authorization** : ASP.NET provides built-in mechanisms for authentication and authorization, making it easier to secure applications. It supports various authentication providers and authorization policies.
11. **Data Binding** : ASP.NET supports powerful data-binding features that simplify the process of connecting UI elements to data sources, such as databases or XML. This includes controls like the GridView and FormView.
12. **Master Pages and Themes** : Master Pages allow developers to define a consistent layout for pages in a web application. Themes provide a way to define a consistent look and feel across multiple pages.
13. **Caching** : ASP.NET includes caching mechanisms to improve application performance. Developers can use output caching, data caching, and fragment caching to store and retrieve frequently used data.
14. **Dependency Injection (ASP.NET Core)** : ASP.NET Core includes built-in support for dependency injection, making it easier to manage and organize components within the application. This promotes a more modular and testable codebase.
15. **Cross-Platform (ASP.NET Core)** : ASP.NET Core is designed to be cross-platform, allowing developers to build and run applications on Windows, Linux, and macOS. This is particularly beneficial for modern cloud-based and containerized applications.

ASP.NET File Extension :

- Page File : Design File - .aspx and code file - .aspx.cs or .aspx.vb
- User Control : .ascx
- Web Service : .asmx
- Master File : .master
- Site Map : .sitemap
- Configuration File : .config

Structure of ASP.NET Web Page :

ASP.NET pages are simply a text file that have the .aspx file name extension and can be place on any web server. The main components of ASP.NET page are as follows :

1. Directives

2. Code declaration block
3. ASP.NET controls
4. Code render block
5. Server side comments
6. Server side include directive
7. Literal text and HTML Tags

1. **Directives :** The directives control is used to how the page is compile and it is marked by the tags, <%@ ……… %>. It can appear anywhere in the page but by default it is placed at the top of the page.

   Page and Import are the two types of the directives.

   - **Page Directives :** A page directives is used to specify default programming language for a page. A page directive is also be used to trace and debugging for the page.

     Example :

     ```
     <%@Page language=”C#”%>
                  or
     <%@ language=”C#”%>
                  The keyword Page in page directives is optional.

     <%@Page trace=”true”%>
                  or
     <%@ debug=”true”%>
     ```

   - **Import Directives :** By default only certain namespace are automatically imported into an ASP.NET web page. If anyone class that is not member of a default namespace then you must explicitly imported the namespace of the class or you must use fully qualified name of the class.

     Example :

     ```
         <%@Imoprt namespace=”System.data.Sqlclient”%>
     ```

2. **Code Declaration Block :** A code declaration block contains all the application logic for a page. It also includes declarations of global variables, and functions. It must be written within the <script runat= "server"> tag. User can written subroutines and functions only within the code declaration block.

   The <script> tag has two optional parameters:

   - **Language** : You can specify the programming language to be used within the <script> tag. Otherwise, the language specified in the Page directive is used.
   - **SRC** : You can specify an external file that contains the code block.

Example :

```
<script runat="server">
        protected void Page_load(object sender, EventArgs e)
        {
                // Statements;
        }
        protected void btnAdd_Click(object sender, EventArgs e)
        {
                // Statements;
        }
</script>
```

3. **ASP.NET Controls :** ASP.NET controls can be mixed with text and static HTML in a page. All controls must appear within a <form runat= "server"> tag. Some controls such as <span runat= "server"> and the Label control can appear outside this tag. You can have only one form per page in ASP.NET.

4. **Code Render Blocks :** If you wish to execute code within HTML, you can include the code within code render blocks. There are two types of code render blocks:

- Inline Code: It executes a statement or series of statements. It is marked by the characters <% and %>.
- Inline Expressions: They display the value of a variable or method. They can be considered as shorthand notation for the Response.Write method. They are marked by the characters <%= and %>.

Example :

```
<html>
        <head> <title> Code Render Blocks </title> </head>
        <body>
        <form runat= "server">
                <h1> Code Render Blocks</h1>
                <%
                        String name;
                        name = "Purva Harshal Patil."
                %>
                <b>My Name is: <%= name %> </b>
                <p>
                        <% name = "Bhakti Harshal Patil." %>
                        My Name is:     <%= name %>
                </p>
            </form>
        </body>
</html>
```

5. **Server-Side Comments :** You can add comments in server-side code using the characters <%-- and --%>. The main use of these comment blocks is to add documentation to a page.
6. **Server-Side Include Directives :** You can include a file in an ASP.NET page by using a server-side include directive. It is executed before any of the code in the page. If the file is in the same directory or in a sub-directory of the page including the file, this directive is written as: <!-- #INCLUDE file="includedfile.aspx" -->

   Note : It is recommended that you avoid using server-side include directives. It is better to use user controls.
7. **HTML Tags and Literal Text :** You can build the static part of an ASP.NET page using HTML tags and literal text. The HTML content of your page is also compiled along with the rest of the contents. The literal text has been made bold and converted to uppercase before being rendered in the browser.

Example:

```
<script RunAt="Server">
        protected void Page_Load(Object sender, EventArgs e)
   {
                foreach (LiteralControl lcControl in Page.Controls)
                {
                lcControl.Text = "<b>" + (lcControl.Text.ToUpper());
                }
        }
</script>
<html>
        <head> <title> LiteralControl Class </title> </head>
        <body>
                <p> This is some literal text</p>
        </body>
</html>
```

ASP.NET Web Pages Model :

A Web page is consisting of following two parts: The Visual portion - Which includes markup, server controls and static text. The programming logic – Which includes event handler and other code.

ASP.NET provides two model for managing the visual elements and code, Single Page Model and Code behind Page Model (Two page model). In the given two models the functionality are same and user can use same controls and code for both model.

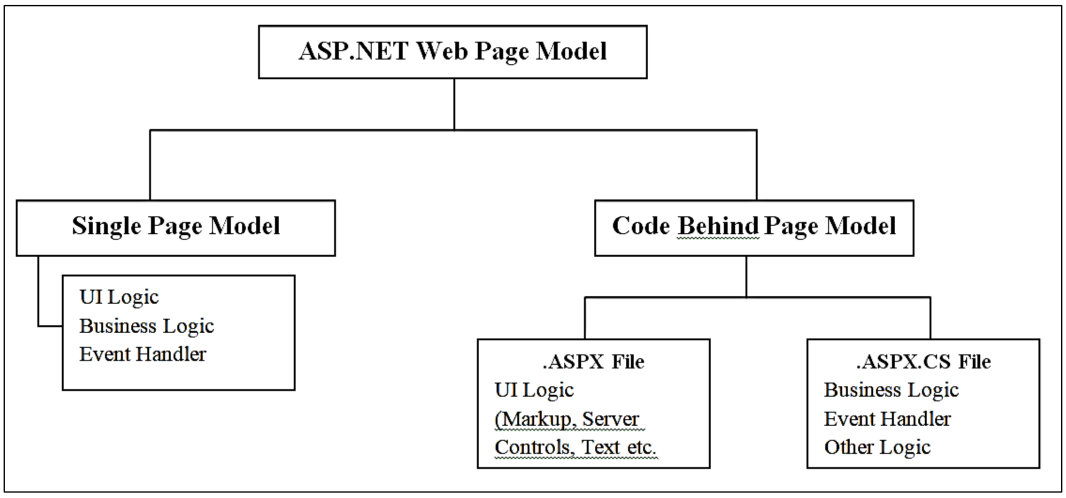


Fig. 3.1 : ASP.NET Web Page Model.

1. **Single File Pages Model :**

In the single-file page model, HTML markup of the page and its programming code are in the same physical .aspx file. The programming code is contained in a <script> block that specifies the attribute runat="server" to mark it as code that ASP.NET should process and execute on the server side.

Example : default.aspx

```
<%@ Page Language="C#" %>
<script runat="server">
   void submit_Button_Click(object sender, EventArgs e)
   {
      result_Label.Text = "Welcome to " + name_TextBox.Text;
   }
</script>
```

```
<html >
<head runat="server">
  <title>Single File Page Model</title>
</head>
<body>
 <form id="form1" runat="server">
  <div>
   <asp:Label ID="Label1" runat="server" Text="Enter the Name:"> </asp:Label>
    <asp:TextBox ID="name_TextBox" runat="server"></asp:TextBox>
    <br />
    <asp:Button ID="submit_Button" runat="server" Text="Submit"
                OnClick="submit_Button_Click" />
    <br />
    <asp:Label ID="result_Label" runat="server" Text="Result"></asp:Label>
  </div>
  </form>
</body>
</html>
```

Above example shows a single-file page default.aspx containing a Button control and a Label control. The Server side code shows the click event handler for the Button control inside a script block.

Advantages of single-file page model :

- If the page contains less code, single page code model makes it easy to read and maintain.
- Pages are easier to deploy.
- Managing files in source code control system is easier, because the page is self-contained in single file.
- There is no dependency between file, a single file page is easier to rename.

2. **Code Behind Pages Model (Two Page Model):**

In the code-behind model, HTML markup is kept in one file (with extension .aspx) and the programming code in another (with extension .cs or .vb). The code file contains a partial class, which indicates that the class contains only some of the total code that makes up the full class for the page. The partial class inherits from a base class (either System.Web.UI.PAge or a class derived from System.Web.UI.Page).

There are two differences in the .aspx page between the single-file and the code-behind models:

1. In the cod-behind model, there is no <script> block with the runat="server" attribute.
2. In the code-behind model, the @PAge directive contains attributes that refernce an external file and class.

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

- The CodeFile - attribute specifies the cod-behind file with which the page is associated.
- The Inherits - attribute specifies a cod-behind class for the page to inherit. This class can be any class that is derived from the Page class.

Example: default.aspx File

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="_Default" %>
<html>
<head runat="server">
   <title>Code Behind Model</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <asp:Label ID="Label1" runat="server" Text="Enter the Name:"> </asp:Label>
      <asp:TextBox ID="name_TextBox" runat="server"></asp:TextBox>
      <br />
      <asp:Button ID="submit_Button" runat="server" Text="Submit"
                  onclick="submit_Button_Click" />
      <br />
      <asp:Label ID="result_Label" runat="server" Text="Result"></asp:Label>
   </div>
   </form>
</body>
</html>
```

The above code shows Default.aspx containing Button control and Label control. VS.NET adds three attributes to the @Page directive:

- Inherits – allows the .aspx page to derive from the code-behind class
- CodeFile – indicates the code behind file associated with the page.

- AutoEventWireup – The automatic binding of page events based on the method naming convention. If set to true ASP.NET performs automatic looking up and binding of events (onclick event).

default.aspx.cs File

The following source code indicates the code behind file for the default.aspx page in the above example. It includes click event handler for the Button control.

```
using System;
using System.Collections.Generic;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
public partial class _Default : System.Web.UI.Page
{
   protected void Page_Load(object sender, EventArgs e)
   {
     //Some Logic
   }
   protected void submit_Button_Click(object sender, EventArgs e)
   {
      result_Label.Text = name_TextBox.Text;
   }
}
```

Advantages of Code Behind Page Model :

- Clean separation of the presentation logic (UI) and code.
- User cant browse .aspx.cs file so code is not expose.
- Code can be reused for multiple pages.

ASP.NET Life Cycle :

When an ASP.NET page runs, the page goes through a life cycle in which it performs a series of processing steps. These include initialization, instantiating controls, restoring and maintaining state, running event handler code, and rendering. The Life Cycle of ASP.NET is divided into two categories: Application Life Cycle and Page Life Cycle.

1. Application Life Cycle :

When an ASP.Net application is launched, there are series of steps which are carried out. These series of steps make up the lifecycle of the application.

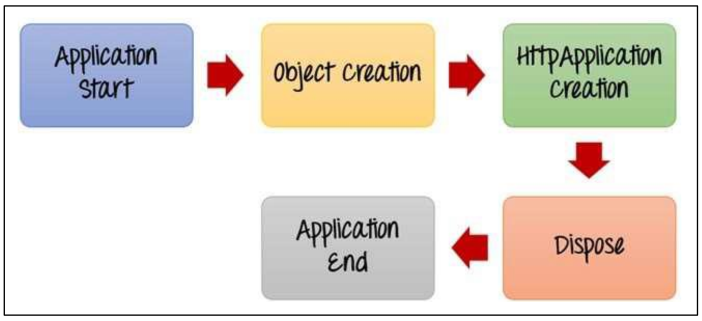


Fig. 3.2 : ASP.NET Application Life Cycle

i. **Application Start :** The life cycle of an ASP.NET application starts when a request is made by a user. This request is to the Web server for the ASP.Net Application. This happens when the first user normally goes to the home page for the application for the first time. During this time, there is a method called Application_start which is executed by the web server. Usually, in this method, all global variables are set to their default values.

ii. **Object Creation :** The next stage is the creation of the HttpContext, HttpRequest & HttpResponse by the web server. The HttpContext is just the container for the HttpRequest and HttpResponse objects. The HttpRequest object contains information about the current request, including cookies and browser information. The HttpResponse object contains the response that is sent to the client.

iii. **HttpApplication Creation :** This object is created by the web server. It is this object that is used to process each subsequent request sent to the application.
For example, let's assume we have two Web applications. One is a shopping cart application, and the other is a news website. For each application, we would have two HttpApplication objects created. Any further requests to each website would be processed by each HttpApplication respectively.

iv. **Dispose :** This event is called before the application instance is destroyed. During this time, one can use this method to manually release any unmanaged resources.

v. **Application End :** This is the final part of the application. In this part, the application is finally unloaded from memory.

2. ASP.NET Page Life Cycle :

When an ASP.Net page is called, it goes through a particular lifecycle. This is done before the response is sent to the user. When a page is requested, it is loaded into the server memory, processed, and sent to the browser. Then it is unloaded from the memory. At each of these steps, methods and events are available, which could be overridden according to the need of the application. In other words, you can write your own code to override the default code.

ASP.NET Page Life Cycle Stages :

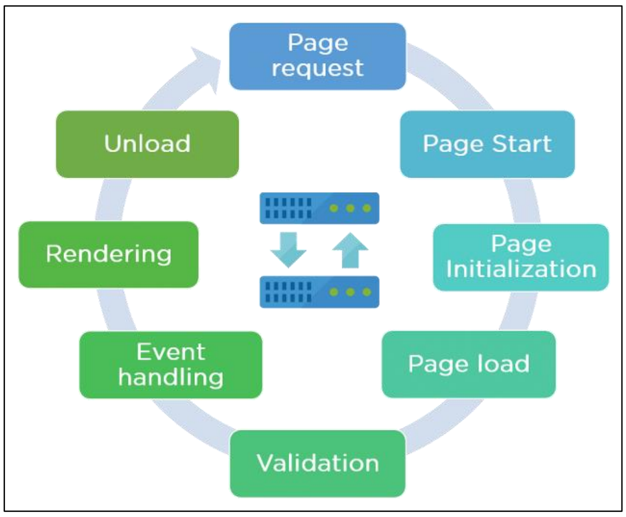


Fig. 3.3 : ASP.NET Page Life Cycle Stages

1. **Page Request :** The page request occurs before the page life cycle begins. This is when the page is first time requested from the server. When the page is requested, the server checks if it is requested for the first time. If so, then it needs to compile the page, parse the response and send it across to the user. If it is not the first time the page is requested, the cache is checked to see if the page output exists. If so, that response is sent to the user.
2. **Page Start :** During this time, two objects, known as the Request and Response object are created. The Request object is used to hold all the information which was sent when the page was requested. The Response object is used to hold the information which is sent back to the user.
3. **Page Initialization :** During this time, all the controls on a web page is initialized. So if you have any label, textbox or any other controls on the web form, they are all initialized.
4. **Page Load :** This is when the page is actually loaded with all the default values. So if a textbox is supposed to have a default value, that value is loaded during the page load time.

5. **Validation :** Sometimes there can be some validation set on the form. For example, there can be a validation which says that a list box should have a certain set of values. If the condition is false, then there should be an error in loading the page.
6. **Postback event handling :** This event is triggered if the same page is being loaded again. This happens in response to an earlier event. Sometimes there can be a situation that a user clicks on a submit button on the page. In this case, the same page is displayed again. In such a case, the Postback event handler is called.
7. **Page Rendering :** This happens just before all the response information is sent to the user. All the information on the form is saved, and the result is sent to the user as a complete web page.
8. **Unload :** Once the page output is sent to the user, there is no need to keep the ASP.net web form objects in memory. So the unloading process involves removing all unwanted objects from memory.

ASP.NET Page Life Cycle Events :

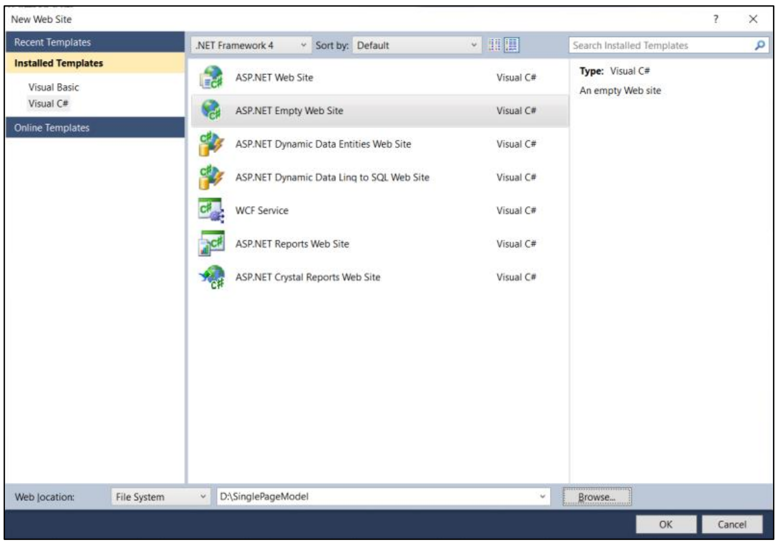
Within each stage of the life cycle of a page, the page raises events that you can handle to run your own code.

1. **PreInit**: This event occurs when the start stage is done but before the initialization stage.
2. **Init** : This event reads and initializes control properties and happens after all the controls are initialized.
3. **InitComplete** : This happens at the conclusion of the page's initialization stage. The event makes changes to the view state that we want to ensure are persisted at the end of the subsequent postback.
4. **PreLoad** : PreLoad occurs at the conclusion of the event-handling stage, and is used for tasks that need all other page controls to be loaded.
5. **Load** : Load is raised initially for the page, then recursively for the child controls.
6. **Control events** : This takes care of handling specific control events like button control clicks.
7. **LoadComplete** : This event is used for many tasks that need other controls found on the page to be loaded. LoadComplete occurs at the conclusion of the event-handling stage.
8. **PreRender** : This event kicks in after the page object has created the controls necessary to render the page.
9. **PreRenderComplete** : This event happens once the pre-render stage is complete.
10. **SaveStateComplete** : This event is raised when the view and control states are saved for both the page and all the controls.
11. **Render** : Render isn't actually an event; rather, it's a method that the Page object calls for each control.
12. **Unload** : This event is first raised for each control, then for the page itself.

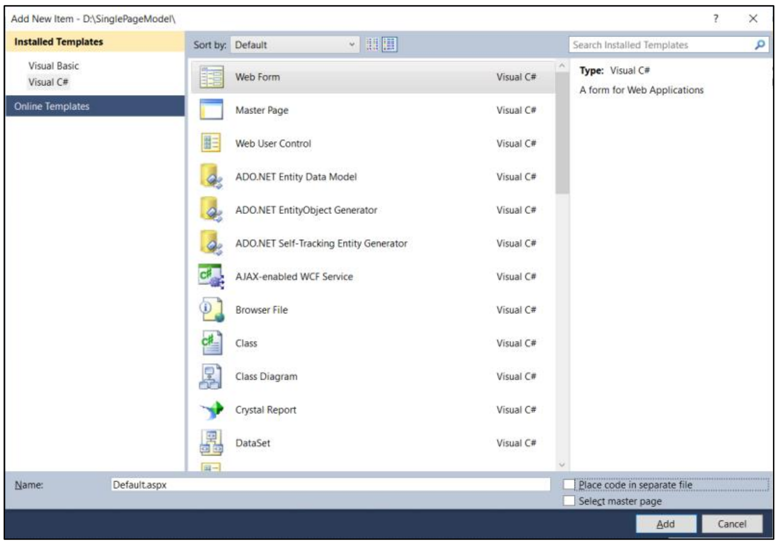
: Practice :

1. Create an ASP .NET web application to demonstrate ASP.NET Single File Page Model.

Step – 1 : First of all, create a new Blank Website in Visual Studio, and then add a Web page to it. But this time don't check the check box and if it's checked then Uncheck it and then click on "Add".


New Web Site
Recent Templates
Installed Templates
Visual Basic
Visual C#
Online Templates
.NET Framework 4
Sort by: Default
Search Installed Templates
ASP.NET Web Site Visual C#
ASP.NET Empty Web Site Visual C#
ASP.NET Dynamic Data Entities Web Site Visual C#
ASP.NET Dynamic Data Linq to SQL Web Site Visual C#
WCF Service Visual C#
ASP.NET Reports Web Site Visual C#
ASP.NET Crystal Reports Web Site Visual C#
Browse...
OK
Cancel

Type: Visual C#
An empty Web site
Web location: File System D:\SinglePageModel


Add New Item - D:\SinglePageModel\
Installed Templates
Visual Basic
Visual C#
Online Templates
Sort by: Default
Search Installed Templates
Web Form Visual C#
Master Page Visual C#
Web User Control Visual C#
ADO.NET Entity Data Model Visual C#
ADO.NET EntityObject Generator Visual C#
ADO.NET Self-Tracking Entity Generator Visual C#
AJAX-enabled WCF Service Visual C#
Browser File Visual C#
Class Visual C#
Class Diagram Visual C#
Crystal Report Visual C#
DataSet Visual C#
Add
Cancel

Type: Visual C#
A form for Web Applications
Name: Default.aspx
☐ Place code in separate file
☐ Select master page

Step – 2 : Add Controls into the files.

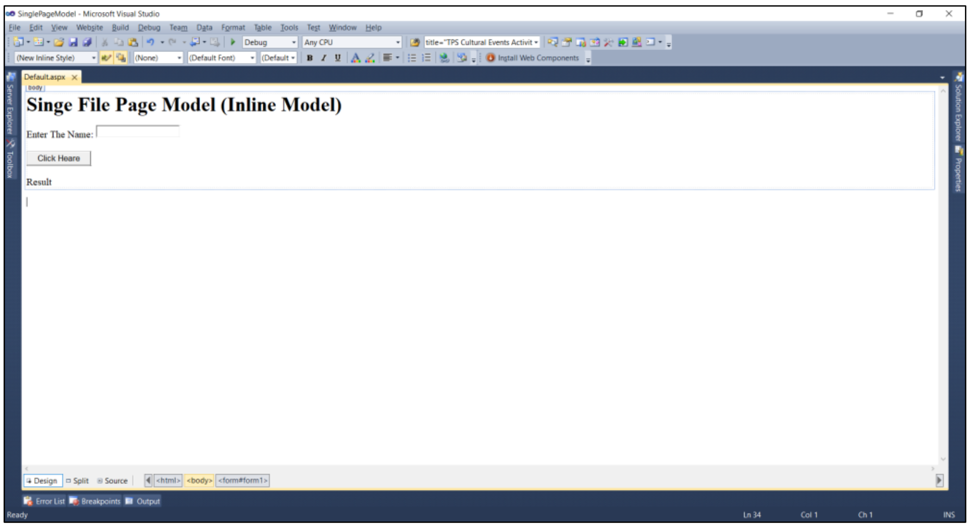
<%@ Page Language="C#" %>

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

<script runat="server">

    protected void Button1_Click(object sender, EventArgs e)
    {
        Label2.Text = Label2.Text + TextBox1.Text;
    }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Singe File Page Model</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
      <h1>Singe File Page Model (Inline Model)</h1>
        <asp:Label ID="Label1" runat="server" Text="Label">Enter The Name:
</asp:Label>
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        <br />
        <br />
        <asp:Button ID="Button1" runat="server" Text="Click Heare"
            onclick="Button1_Click" />
        <br />
        <br />
         <asp:Label ID="Label2" runat="server" Text="Label">Result </asp:Label>
    </div>
    </form>
</body>
</html>
```

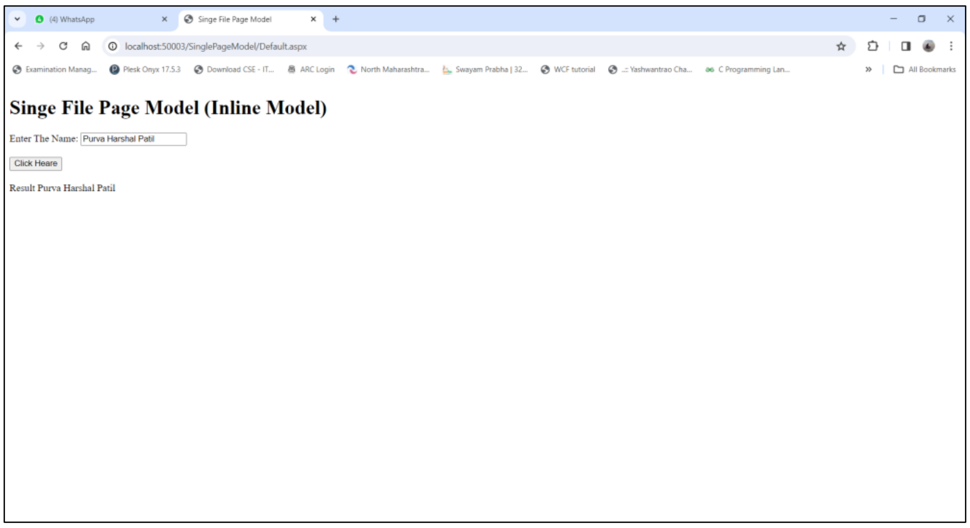

Singe File Page Model (Inline Model)

Enter The Name:

Click Heare

Result

Step – 3 : Built and run the application.


localhost:50003/SinglePageModel/Default.aspx

Singe File Page Model (Inline Model)

Enter The Name: Purva Harshal Patil

Click Heare

Result Purva Harshal Patil

2. Create an ASP .NET web application to demonstrate ASP.NET Page Life Cycle.

Default.aspx :

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="_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>Page Life Cycle</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
     <h1> Page Life Cycle</h1>
     <asp:Label ID="lblName" runat="server" Text="Label"></asp:Label>
     <br />
        <asp:Button ID="submit_Btn" runat="server" Text="Button"
            onclick="submit_Btn_Click" />
    </div>

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

Default.aspx.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_PreInit(object sender, EventArgs e)
    {
        //Work and It will assign the values to label.
        lblName.Text = lblName.Text + "<br/>" + "PreInit";
    }
    protected void Page_Init(object sender, EventArgs e)
    {
        //Work and It will assign the values to label.
        lblName.Text = lblName.Text + "<br/>" + "Init";
    }
    protected void Page_InitComplete(object sender, EventArgs e)
    {
        //Work and It will assign the values to label.
        lblName.Text = lblName.Text + "<br/>" + "InitComplete";
    }
    protected override void OnPreLoad(EventArgs e)
    {
        //Work and It will assign the values to label.
        //If the page is post back, then label contrl values will be loaded from view state.
        //E.g: If you string str = lblName.Text, then str will contain viewstate values.
        lblName.Text = lblName.Text + "<br/>" + "PreLoad";
    }
    protected void Page_Load(object sender, EventArgs e)
    {
        //Work and It will assign the values to label.
        lblName.Text = lblName.Text + "<br/>" + "Load";
        if (!IsPostBack)
        {
            lblName.Text = lblName.Text + "<br/>" + "Purva";
        }
    }
    protected void submit_Btn_Click(object sender, EventArgs e)
    {
        //Work and It will assign the values to label.
        lblName.Text = lblName.Text + "<br/>" + "btnSubmit_Click";
    }
    protected void Page_LoadComplete(object sender, EventArgs e)
    {
        //Work and It will assign the values to label.
        lblName.Text = lblName.Text + "<br/>" + "LoadComplete";
    }
    protected override void OnPreRender(EventArgs e)
    {
        //Work and It will assign the values to label.
        lblName.Text = lblName.Text + "<br/>" + "PreRender";
    }
```

```
    protected override void OnSaveStateComplete(EventArgs e)
    {
        //Work and It will assign the values to label.
        //But "SaveStateComplete" values will not be available during post back. i.e. View
state.
        lblName.Text = lblName.Text + "<br/>" + "SaveStateComplete";
    }
    protected void Page_UnLoad(object sender, EventArgs e)
    {
        //Work and it will not effect label contrl, view stae and post back data.
        lblName.Text = lblName.Text + "<br/>" + "UnLoad";
    }
}
```

: QUESTIONS :

1. What is ASP.NET? Explain the history of ASP.NET?
2. What is ASP.NET? Explain the features of ASP.NET?
3. Write short notes on structure of ASP.NET Page?
4. Write short notes on ASP.NET Web Pages Model?
5. Explain the Single Page Model of ASP.NET with example?
6. Explain the Code Behind Page Model of ASP.NET with example?
7. Write short notes on ASP.NET Page Life Cycle?
8. Write short notes on ASP.NET Application Life Cycle?

***

4. Chapter

ASP.NET Controls

Syllabus :

Working with Web Server Controls (Label, Textbox, Button, Checkbox, Radio button, ListBox, Dropdown etc, Navigation controls (Tree view, Menu navigation), ASP.Net Rich Controls (Adrotator, Calender), Validation Controls (Required Field Validator, Range Validator, Compare Validator)

(10 L, 20 M)

Introduction to ASP.NET Web Form :

ASP.NET Web Forms is a part of the ASP.NET framework developed by Microsoft for building dynamic web applications. It's a web application framework that allows developers to build interactive and data-driven web applications using a variety of server-side controls, event-driven programming model, and a rich set of tools for rapid development.

Using Visual Studio you can create ASP.NET web forms (.aspx file). The Visual Studio IDE that allows us to drag and drop server controls to the web form. It also allow us to set properties, events and methods for the controls.

ASP.NET web forms are event driven pages with server controls, server code and server events. ASP.NET web forms application utilize master template which are the centralized file where you can manage the header, footer and body of your site all in one place.

ASP.NET web forms are made up of two components – The visual portion (aspx file) and the code behind the file (aspx.cs file). The main purpose of ASP.NET web form is to overcome the limitation of ASP and separate view from the application logic.

ASP.NET web forms offers :

- A Separation of HTML and other UI code from application logic.
- A rich suite of server controls from common task including data access.
- A powerful data binding with grate tools support.

- A support for client side scripting that executed in the browser.
- A support the state management.

ASP.NET web forms controls must appear within <form id = “form1” runat=”server”> </form> tag. You can have only one <form> tag per web form.

Features of Web forms :

ASP.NET web forms are full of features and provide awesome platform to create and developed web application.

- **Server-Side Controls** : Web Forms use server-side controls, which are objects that run on the server and encapsulate user interface and other related functionality. Examples include textboxes, buttons, and data controls like GridView and ListView.
- **Event-Driven Programming Model** : Web Forms follow an event-driven programming model. User interactions and other events trigger server-side events that can be handled in the code-behind file. For example, clicking a button can trigger the OnClick event.
- **ViewState** : Web Forms use ViewState to persist the state of controls across postbacks. ViewState is a mechanism that allows the state of server-side controls to be stored on the client and sent back to the server with each request. This helps maintain the state of controls across page postbacks.
- **Code-Behind Model** : Code-behind separates the HTML markup of a page from the server-side code. The markup (aspx) file contains the structure and layout of the page, while the code-behind file (aspx.cs or aspx.vb) contains the server-side logic.
- **ASP.NET Server Controls** : These controls are components that run on the server and encapsulate the user interface and related functionality. Server controls are used in the development of Web Forms pages. Examples include TextBox, Button, DropDownList, and GridView.
- **Data Binding** : Web Forms provide powerful data-binding capabilities, making it easy to bind data from various sources, such as databases or XML, to server-side controls.
- **Master Pages and Themes** : Master Pages allow you to create a consistent layout for your site, and Themes provide a way to define a consistent look and feel.
- **State Management** : In addition to ViewState, ASP.NET Web Forms support other state management techniques like Session state and Application state to manage data across multiple requests.

ASP.NET Server Controls :

The base class for any server control is System.Web.UI.Control. All ASP.NET server controls use the runat =”server” attributes which indicates that the code related to the controls always executes on the web server. ASP.NET also works with event driven model. Server controls also have events defined. These are raised when a user performs any action. Every control in ASP.NET will have three things Properties, Methods and Events.

Properties – A properties is an attribute of control which mainly has its impact on the look of the control.

Example – Name, Text, BackColor, ForeColor, Font, Width, Height etc.

Methods – A method is an action that has to be performed.

Example – Focus(), Add(), Clear() etc.

Events – An events is the time period which tells when an action has to be performed.

Example – Button Click event, TextBox Changed event etc.

```
Protected void submitBtn_Click(Object sender, EventArgs e)
          {
                    …………..
                    …………..
          }
```

ASP.NET provides two sets of server controls. Html Server Controls that directly map to HTML tags. Web Server Controls are provide a more consistent programming model and a higher level of abstraction.

HTML Server Controls :

In ASP.NET, HTML Server Controls are a type of server control that provides a way to encapsulate standard HTML elements as server-side objects. These controls allow you to programmatically manipulate HTML elements on the server side, providing more control and flexibility in your ASP.NET Web Forms applications.

HTML elements could be converted to a HTML Server controls by adding the runat and id attribute.

Example :

```
<input type=”text” id = “nameTxt” runat=”server” size=”40”>
<Button type=”button” id = “submitBtn” runat=”server”> </Button>
```

The HTML server control has the same output and same properties as their corresponding HTML tags. HTML server control provides automatic state management and server side events. HTML server controls are derived from System.Web.UI.HTMLControls base class.

List of ASP.NET HTML Server Controls :

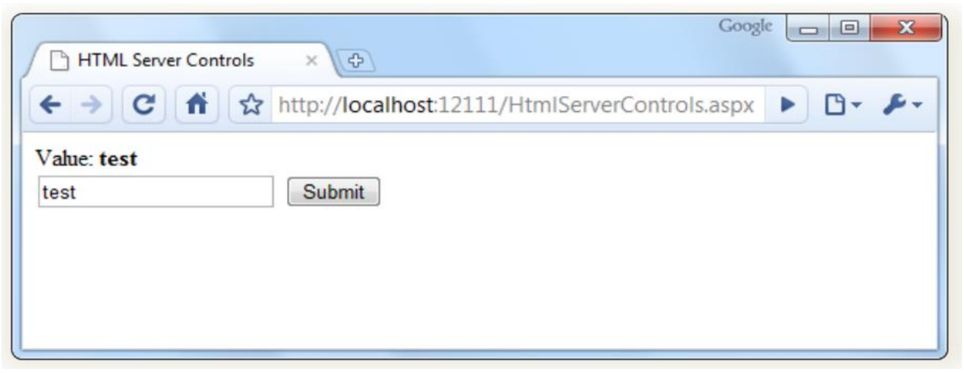
Control	**Description**	**Web Form Code Example**
Button	A normal button that you can use to respond to Click events	<input type=button runat=server>
Reset Button	Resets all other HTML form elements on a form to a default value	<input type=reset runat=server>

Submit Button	Automatically POSTs the form data to the specified page listed in the Action= attribute in the FORM tag	<input type=submit runat=server>
Text Field	Gives the user an input area on an HTML form	<input type=text runat=server>
Text Area	Used for multi-line input on an HTML form	<input type=textarea runat=server>
File Field	Places a text field and a Browse button on a form and allows the user to select a file name from their local machine when the Browse button is clicked	<input type=file runat=server>
Password Field	An input area on an HTML form, although any characters typed into this field are displayed as asterisks	<input type=password runat=server>
CheckBox	Gives the user a check box that they can select or clear	<input type=checkbox runat=server>
Radio Button	Used two or more to a form, and allows the user to choose one of the controls	<input type=radio runat=server>
Table	Allows you to present information in a tabular format	<table runat=server> </table>
Image	Displays an image on an HTML form	<img src="FileName" runat=server>
Dropdown	Displays a list of items to the user, but only one item at a time will appear. The user can click a down arrow from the side of this control and a list of items will be displayed.	<select> <option></option> </select>
ListBox	Displays a list of items to the user. You can set the size from two or more to specify how many items you wish show. If there are more items than will fit within this limit, a scroll bar is automatically added to this control.	<select size=2 runat=server> </select>

Example of ASP.NET HTML Server Controls :

```
 <%@ Page Language="C#" %>
script language="c#" runat="server">
  void ButtonSubmit_Click(Object sender, EventArgs e)
```

```
{
    Response.Write("Value:<b>"+TextField.Value+"</b>");
  }
</script>
<html>
<head>
  <title>HTML Server Controls</title>
</head>
<body>
  <form id="formMain" runat="server">
    <input id="TextField" type="text" runat="server" />
    <input id="ButtonSubmit" runat="server" value="Submit"
     onserverclick="ButtonSubmit_Click" type="button"/>
  </form>
</body>
</html>
```



Value: **test**

test

Web Server Controls :

Web Server Controls in ASP.NET are server-side components that encapsulate user interface elements and programmable logic. These controls run on the server and emit HTML or other markup to the client's browser. Web Server Controls provide a higher level of abstraction compared to HTML Server Controls, offering a more feature-rich and event-driven model for building dynamic web applications. Web server controls are specially designed by ASP.NET for processing at server side only. The web server controls are inherited from WebControl class. Web controls have asp: prefix. The web server controls can detects browser capabilities. It provides rich object model and type safe programming. Web Server Controls in ASP.NET provide a powerful and flexible framework for building interactive and dynamic web applications.

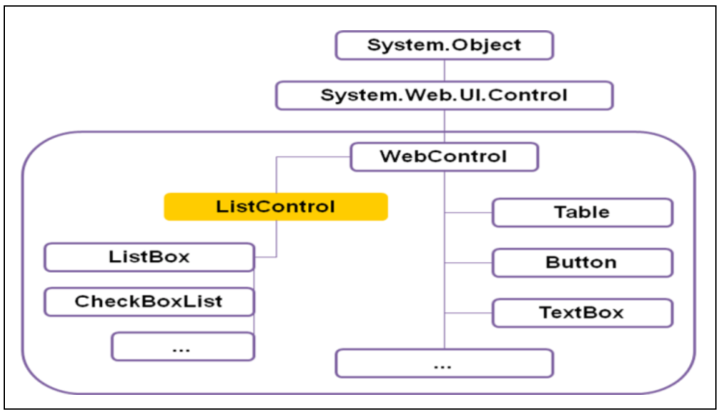


Fig. 4.1 : Web Server Controls.

Syntax :

```
<asp:ControlName ID="controlID" runat="server" property1=value
event="eventName"></asp:ControlName>
```

Example:

```
<asp:TextBox ID="TextBox1" runat="server"
ontextchanged="TextBox1_TextChanged"></asp:TextBox>
<asp:Button ID="Button1" runat="server" Text="Button" onclick="Button1_Click" />
```

Common properties of Web server controls are :

- BackColor – Set the background color of the controls.
- Enabled – Enable or disable the control (true/false value).
- Font – Set the font of controls.
- ForeColor – Set the color of the text, specified as an HTML color or hex value.
- TabIndex – Specified the tab order of the controls.
- Visible – Visible or invisible the control (true/false value).
- Width – Set the width of the control in terms of characters, pixels, or percentage.

ASP.NET Web controls offer following features :

- Every Web control has two mandatory attributes like runat=”Server” and ID=”Some ID Value”.
- These controls provide a rich object model that uses familiar object oriented techniques along with type safety.
- Can be programmatically accessed in code behind files.
- State of controls is saved automatically.

- Events are processed on the server side.
- Functionality of the controls is defined in event procedures which are invoked when an event occurs.

Types of Web server controls are :

Web Server controls can be divided in to following categories –

i. Basic Web Server Controls,
ii. List Web Server Controls,
iii. Rich Web Server Controls
iv. Validation Web Server Controls.
v. Navigation Web Server Controls.

i. Basic Web Server Controls :

Basic Web Server controls provide the same functionality as their HTML Server controls counterparts. However Basic Web Server controls includes additional methods, events and properties against which you can program.

- **Label web server control:** The label web server acts as non-editable text control and allow to programmatically setting text in an Asp.Net web page.
  Example :
  ```
  <asp:Label id="lblMsg" runat="server" Text="Online Application Form"> </asp:Label>
  ```
- **Text Box Web server control :** Gives the user an input area on a Web form. Commonly used server-side events are Textchanged. Commonly used method is focus().
  Example:
  ```
  <asp:TextBox id="txtName" runat="server"> </asp:TextBox>
  txtName.Text = "Purva Harshal Patil";
  txtName. Focus();
  ```
- **Button web Server Control:** A normal button that you can use to respond to click event on the server. Commonly used server-side events are Click, Command etc.
  Example :
  ```
      <asp:Button id="SubmitBtn" runat="server" Text="Submit"
      onclick="SubmitBtn_Click"> </asp:Button>
  ```
- **Link Button Web Server Control:** Create a hyperlink - style button on a Web Form page. Commonly used server side events are click, Command etc.
  Example:
  ```
      <asp:LinkButton id="signoutBtn" runat="server" Text="Sign Out">
      </asp: LinkButton >
  ```
- **ImageButton Web Server Control:** It can display a graphical image and when Clicked postback to the server command information such as a mouse co-ordinates within the

image when clicked. Specify the image to display in the control by setting ImageUrl property. Both the click and command events are raised when the image Button control clicked.

Example :

```
<asp: ImageButton" id="imgBtn" runat="server" "ImageUrl="~/Image/Go.png"
onclick="imgBtn_click" />
```

- **RadioButton Web Server control:** Creates an individual radio button on page. You can group multiple radio buttons together to provide a mutually exclusive set of choice. Commonly used Server-side event is Cheekchanged.

  Example:

  ```
  <asp:RedioButton ID="radioMale" runat="Server" Text = "Male"
  GroupName="gender" Checked ="True"/>
  <asp:RedioButton ID="radioFemale" runat="Server" Text="Female""
  GroupName="gender"/>
  ```

- **Hyperlink Web Server Control:** Create a link on the page that user can click to move to another page, or location on the page. Specify the page location to link to by using NavigateUrl property. The link can either be displayed as text or an image. To display an image set the ImageUrl property.

  Example:

  ```
  <asp:HyperLink id="hyperlink1" runat="server"" Text="Go To Next"
  NavigateUrl="~/home.aspx"/>
  <asp:Hyperlink id="hyperlink2" runat="server"
  Navigateurl="www.basponccollege.org" ImageUrl="~/Image/clglogo.png"/>
  ```

- **Image Web Server Control:** Display a web-compatible image on the web form page. Setting the ImageUrl property specifies the path to displayed Image. You can specify the text to display in a place of the image when the image is not available by setting AlternateText property.

  Example:

  ```
  <asp:Image id="imgclglogo" runat="server" ImageUrl="/Image/clglogo.png"
  AlternateText="College Logo"/>
  ```

- **Check Box Web Server Control:** It is used to get multiple inputs from user. It allows user to select choices from the Set of choices. Commonly used server-side events checkchanged.

  Example:

  ```
  <asp:CheekBox id="color1" runat="server" Text="Red" />
  <asp:CheckBox id="color2" runat="server" Text= "Blue" />
  ```

```
<asp:CheekBox id = "color3" runat="server" Text = "Green"/>
```

- **Panel Web Server Control:** The panel control is a container for other controls. It is especially useful for generating Controls programmatically and displaying and hiding groups of controls.

  Example:

  ```
  <asp:Panel id="Studbasicinfo" runat="server" Height="200px" width="300px">
      (Other Controls declared here)
  </asp:Panel>
  ```

ii. List Web Server Controls :

List controls are special web server control that support binding to Collections. You can used to List controls to display rows of data in a customized template format. All list controls expose Datasource and DataMember properties which are used to bind to collections. Common event is selectedIndexChanged.

- **ListBox :** The listbox web server control can be used to display multiple items at once and to enable user to select one or more items from predefined List. The common event of listbox control is SelectedIndexChanged. To enables multiple item selection set the SelectionMode property with value Single or Multiple. To specify items that you want to appear in the ListBox control, place a ListItem element for each entry between the opening and closing tags of the ListBox control.

  Example :

  ```
  <asp:ListBox id="Color" runat="Server" Rows = "6" width="100px"
  selectionMode: "Single">
          <asp:ListItem value="Red" Selected="True"> Red <asp:ListItem>
          <asp:ListItem values "Blue"> Blue </asp:ListItem>
          ……………..
          ……………..
          <asp:ListItem values "Black"> Black </asp:ListItem>
  </asp:ListBox>
  ```

- **CheckBoxList :** The checkBoxlist controls create a multi-selection check box group. This control supports binding to a data source. To specify items that you want to appear in the cheekBoxList control, place a ListItem element for each entry between the opening and closing tags of the CheekBoxList control.

  Example :

  ```
  <asp:CheckBoxList id = "CheckBoxList1" runat="server" TextAlign =
  "Right|Left">
  ```

```
        <asp:ListItem value="values1" Selected= "T/F"> Text </asp:ListItem>
        <asp:ListItem value="values2" Selected= "T/F"> Text </asp:ListItem>
        ……………..
        ……………..
    </asp:CheckBoxList>
```

- **RadioButtonList :** The RadioButtonList control creates a group of radio buttons. This control supports binding to data source.

  Example:

```
    <asp:RedioButtonList id=" RadioButtonList1" runat="server" >
        <asp:ListItem value = "Valuel" Selected= "T/F">  Item1 </asp:ListItem>
        ……………..
        ……………..
    </asp:RedioButtonList>
```

- **DropDownList :** This control enables user to select Pros Single-Selection drop-down list. The drop-down list can contain any number of items. This control also support data binding.

  Example 1 :

```
    <asp:DropDownList id="DropDownList1" runat server" >
    <asp:ListItem value="value1" Selected= "T/F"> Item1 </asp:ListItem>
    ……………..
    ……………..
    </asp: DropDownlist >
```

  Example 2 :

```
    <asp:DropDownList id="DropDownList2" runat="server" AutoPostBack="T|F"
    DataSource = "<% databindingexepression>"
    DataTextfield = "DataSourceField"
    DataValueField = "DataSourceField"
    OnSelectedIndexChanged = "onselectedIndexchanged method">
    </asp:DropDownList>
```

- **DataGrid :** The dataGrid control renders a tabular data-bound grid. This controls allows you to define various types of columns to central the layout of the cell contents of the grid (bound column, template column) and add specific functionality (such as edit button column, hyperlink column etc.). The control also supports a variety of options for paging through data. Data Source for the DataGrid can be either a DataTable or database.

  Example : **Default.aspx.**

```
    <asp:DataGrid id="DataGrid1" runat="server">
    <asp:DataGrid>
```

Default.aspx.cs

```
page-Load (object sender, EventArgs e)
{
        DataTable dt = new DataTable();
        dt.Columns. Add ("SrNo");
        dt.columns. Add ("Name");
        dt.Columns.Add("Class");
        dt.Rows.Add("1", "Harshal", "MCA");
        dt.Rows.Add("2", "Sujata", "MCA");
        dt.Rows.Add("3" "Pallavi", "Msc");
        DataGrid1. Datasource = dt;
        DataGrid1. DataBind();
}
```

SrNo	Name	Class
1	Harshal	MCA
2	Sujata	MCA
3	Pallavi	Msc

- **DataList :** Datalist control is a light weight Server Side control that work as container for data items. It is used to display data into a list format to the web pages. It display data from the data Source The data source can be either DataTable or a table from database.

  Example: **Default.aspx.**

```
<asp:Datalist id="Datalist1" runat="server">
        <ItemTemplate>
                <table>
                        <tr>
                                <td>
                                        <b>SrNo:</b>
                                        <span> <%# Eval("SrNo") %></span>
                                        </br>
                                        <b> Name: </b>
                                        <span> <%# Eval("Name") %></span>
                                        </br>
                                        <b> Class: </b>
                                        <span> <%# Eval ("Class")%> </span>
                        </td>
                    </tr>
                </table>
            < /ItemTemplate>
</asp: Datalist>
```

Default.aspx.cs

```
Page load (object sender, EventArgs e)
{
        DataTable dt = new DataTable();
        dt.Columns. Add ("SrNo");
        dt.columns. Add ("Name");
        dt.Columns.Add("Class");
        dt.Rows.Add("1", "Harshal", "MCA");
        dt.Rows.Add("2", "Sujata", "MCA");
        dt.Rows.Add("3" "Pallavi", "Msc");
        Datalist1. Datasource = dt;
        Datalist1. DataBind();
}
```

SrNo: 1 Name: Harsh Class: MCA
SrNo: 2 Name: Sujata Class: MCA
SrNo: 3 Name: Pallavi Class: Msc

- **Repeater Control :** This control creates a data-bound list control that allows custom layout by repeating specified template for each item displayed in the list. Repeater Control has no built in layout or Style, you must explicitly declare all HTML layout, Formatting and style tags within the controls templates.

  Example : **Default.aspx.**

```
 <asp:Repeater id="rptstud" runat="server">
     <HeaderTemplate>
     <table>
```

```
        <tr>
        <th> So. No </th>
        <th> Name </th>
        <th> Class </th>
        </tr>
        </HeaderTemplate>
        <ItemTemplate>
        <tr>
        <td><%# Eval("SrNo") %></td>
        <td><%# Eval("Name")%></td>
        <td><%# Eval ("Class") %> </td>
        </ItemTemplate>
        <FooterTemplate>
        </table>
        </FooterTemplate>
</asp:Repeater>
```

Default.aspx.cs

```
Page load (object sender, EventArgs e)
{
        if(!Page.IsPostBack)
        {
                DisplayData();
        }
}
public void DisplayData()
{
        DataTable dt = new DataTable();
        dt.Columns. Add ("SrNo");
        dt.columns. Add ("Name");
        dt.Columns.Add("Class");
        dt.Rows.Add("1", "Harshal", "MCA");
        dt.Rows.Add("2", "Sujata", "MCA");
        dt.Rows.Add("3" "Pallavi", "Msc");
        rptstud. Datasource = dt;
        rptstud. DataBind();
}
```

Sr. No.	Name	Class
1	Harshal	MCA
2	Sujata	MCA
3	Pallavi	Msc

iii. Rich Web Server Controls :

The controls which are not standard Control and allow you to do some advancing functionality are called as Rich controls. Rich controls are built with multiple HTML elements & contain rich functionality. Rich controls is combination of other Standard Controls, it works as Single control. Rich controls are web controls that are defined as single object but give a complex functionality. Rich controls provide an object model that has a complex HTML representation and also client side JavaScript.

Different rich controls are as follows :

- FileUpload Control
- Calendar Control
- AdRotator Control
- MultiView Control
- **FileUpload Control :** Fileupload control is used to browse & upload files. After the file is uploaded, you can store the file on any drive or database. Fileupload Control is the combination of a browser button and textbox for entering the filename.

  Some important properties of FileUpload control

  - FileBytes - It returns the contents of uploaded File as a byte array.
  - FileContent - you can get the uploaded file Contents as a stream.
  - FileName - provides the name of uploaded file
  - **Hasfile** - It is a Boolean property that cheek whether particular file is available or not.

    Example: **Default.aspx.**

```
<asp: FileUpload id="fileupload1" runat="Server"/>
</br>
<asp:Button id="SaveBtn" runat="server"> </asp:Button>
```

Default.aspx.cs

```
Void SaveBtn_Click (object sender, EventArgs e)
{
    StringBuilder sb=new StringBuilder();
    if(fileUpload1.Hasfile)
    {
```

```
                sb. AppendFormat ("Uploading File: {0}", FileUpload1.FileName);
                // Saving file
                fileUpload1.SaveAs("D:\\myfile\" + fileupload1.FileName);
                // Showing File information
                sb. AppendFormat ("</br> SaveAs: {0}", Fileuplod1.
                postedFile.FileName);
                sb. Appendformat("</br> File Type: {0}, Fileuplod1.
                postedFile.ContentType);
                }
            }
```

- **Calendar Control :** Asp.Net provides a Calendar Control that is used to display a calendar on Web page. The calendar control is used to create a rich functionality and good looking Calendar that shows one month at a time. Calendar Control provides you lots of property and events. By using these properties and events you can perform the following task with calendar Control - select data, Selecting a day, a week. or a month and Customize the calendar's appearance.

  Syntax : <asp:calendar id="calendar1" runat="server"> </asp: calendar>

  A calendar control Supports SelectionMode property that allows you to select single day, week or entire month. The calendar control supports three important events.

  - **SelectionChanged** - This event is fired when you select a day, week or an entire month.
  - **DayRender** - This event is fired when each data cell of the calendar control is rendered.
  - **VisibleMonthChanged -** It is raised when user Changes a month.

Example: Default.aspx.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title>Calendar Control Example</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <asp:Calendar ID="Calendar1" runat="server" BackColor="#FF6600"
BorderColor="#336600" BorderStyle="Solid" BorderWidth="2px" Caption="Select
date and month" DayNameFormat="Full"
OnSelectionChanged="Calendar1_SelectionChanged" ShowGridLines="True"
```

```
Width="459px"></asp:Calendar>
        <br />
        <asp:TextBox ID="TextBox1" runat="server" Height="26px"
Width="203px"></asp:TextBox>
        <br />
    </div>
    </form>
</body>
</html>
```

Default.aspx.cs

```
protected void Calendar1_SelectionChanged(object sender, EventArgs e)
{
    TextBox1.Text = "selected date is " + Calendar1.SelectedDate;
}

protected void Page_Load(object sender, EventArgs e)
{
    Calendar1.SelectedDate = DateTime.Now;
 }
```

- **AdRotator Control :** AdRotator control is used to display different advertisements randomly in a page. The list of advertisement is stored in either extended XML file or in a database table. The extended XML file is called as the advertisement file. Before using AdRotator control you need to create the Xml file.

Advertisement File (XML File): The advertisement file is an extended XML file which contains the information about the advertisement to be displayed. Like all XML file, the advertisement file need to be a structured text file with well-defined tags. Advertisement file begins and ends with <Advertisement> tag. Inside the <Advertisement> tag there are several <Ad> tags which define each Advertisement. **The pre-defined elements inside the <Ad> tag are :**

- <ImageUrl> - The path of image that will be displayed.
- <NavigateUrl> - The link that will be followed when the user clicks the ad.
- <AlternateText> - An alternate text for image.
- <Keyword> - A category for the ad. i.e. a group of advertisements. This used for filtering.
- <impressions> - The number indicating how often an advertisement will appear.

Syntax :

```
<asp:AdRotator id = ”AdRotator1” runat = “server”
AdvertisementFile = “XMLfile” Target = “_blank”/>
```

Property of AdRotator control

- AdvertisementFile:- The path to the advertisement file.
- Target:- The browser window or frame that displays the content of the content of the page links .

Example :

- Step 1: Open Visual Studio -> Create a new empty website.
- Step 2: Create a New web page for display AdRotator.
- Step 3: Drag and Drop AdRotator Control on a webpage from toolbox.
- Step 4: Right click on solution Explorer -> Add New Item -> Add New XML file in project for write advertisement details.
- Step 5: Write code in XML file for advertisement.
- Step 6: Assign XML file to advertisementFile property of AdRotator Control.

MyAdvFile.xml

```
<?xml version = “1.0” encoding = “utf-8” ?>
<Advertisements>
<Ad>
  <ImageUrl> img/nmulogo.png </ImageUrl>
  <NavigateUrl> www.nmu.ac.in </NavigateUrl>
  <AlternateText> KBC NMU Jalgaon </AlternateText>
  <Impressions> 20 </Impressions>
</Ad>
<Ad>
  <ImageUrl> img/ponclogo.png </ImageUrl>
<NavigateUrl> www.basponcollege.org </NaviagateUrl>
  <AlternateText> BASPONC BSL </AlternateText>
  <Impressions> 20 </Impressions>
</Ad>
</Advertisements>
```

Default.aspx

```
<h1> AdRotator Control Example </h1>
<asp:AdRotator id = “Adrotator1” runat = “server” Target = “_blank”
          AdvertisementFile =     “~/MyAdvfile.xml”/>
```

- **MultiView Control :** Multiview control is an asp.net web server control. MultiView control is same as tab control. If you want to make a complex designing on one web form then you have to use multiview. Multiview control is a container of several view control. In a multiview control there are many view control for designing separation of view for use. Using a multiviews control we can feel the multiple page view design on a single webpage. Multiview control is reponsible for display one view control at a time. The view displayed is called the active view.

  Syntax :

```
< asp:MultiView id =”Multiview1” runat = server >
<asp:View id = “view1” runat = server >
                ------
                ------
</asp:View>
<asp:View id = “view2” runat = server >
                ------
                ------
</asp:View>
        |
        |
</asp : Multiview>
```

We can see single view at a time on web page by specify the ActiveViewIndex property of multiview control. All view control assign automatically index to all it. The index always starts from zero. i.e. First view1 index is zero. Second view2 index is one and so on. If we want to display first view on web page then we need to write. Multiview1.ActiveViewIndex = 0;

Example : Default.aspx

```
<table>
        <tr>
                <td>
                        <h1> Multiview Control </h1>
                </td>
        </tr>
        <tr>
                <td align = “center”>
                        <asp:Button id = “btnView1” runat = “server”
                                OnClick = “btnView1_Click”/>
                        <asp:Button id = “btnView2” runat = “server”
```

```
                              OnClick = “btnView2_Click”/>
                    </td>
          </tr>
          <tr>
                    <td>
                              <asp:Multiview id = “multiview1” runat  = server >
                    <asp:View id = “view1” runat = “Server”>
                                        <h3> View- I </h3>
                                                            :
                                                            :
                              </asp:view>
                              <asp:view id = “view2” runat =”server”>
                                        <h3> view – II </h3>
                              </asp:view>
                    </asp:multiview>
                    </td>
          </tr>
</table>
```

Default.aspx.cs

```
Page_Load (Object Sender, EventArgs e)
{
          Multiview1.ActiveViewIndex = 0;
}
btnView1_Click (Object Sender, EventArgs e)
{
          Multiview1.ActiveViewIndex = 0;
}
btnview2_Click (Object Sender, EventArgs e)
{
          Multiview1.ActiveViewIndex = 1;
}
```

iv. Validation Web Server Controls :

Validation is an important concept to get a correct output as the user gives the input in waveform must be in proper format then only the server will result in a meaningful output. Data entered by the user must perform validation before sending to the server for processing.

Definition: Validation is checking whether the user has entered correct data or input or not ends the process the request and sends it to the server for output.

Types of validation : there are two types of validation in ASP.NET.

i. **Client-side validation : C**lient side validation is done on the client browser. Client side validation is faster than server side validation. We use JavaScript, JQuery for client side validation. Client side validation is good and fast but we have to be dependent on browser and scripting language support.

ii. **Server-Side Validation :** Server side validation is done on web server. We use custom logic code for server side validation. Server side validation is used to remove the limitation of client side validation. i.e. Browsers dependencies and scripting language support. Server side validation takes place before data processing in the database.

BaseValidator class is parent for all asp.net validation controls. Some common and essential properties provided by the BaseValidator Control class are as follows :

- **ControlToValidate** : Which control to be validate.
- **Display** : A made in which error message is shown like static, dynamic, none.
- **ErrorMessage** : Error message to be displayed
- **Text** : If validation fails, the message to be shown
- **SetFocusOnError** : Focus is set to the control having incorrect values.

Asp.net provides a set of validation controls that provides an easy-to-use but powerful way to check for error and if necessary display message to user. Asp.net 4.0 contains the six types of validation controls.

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

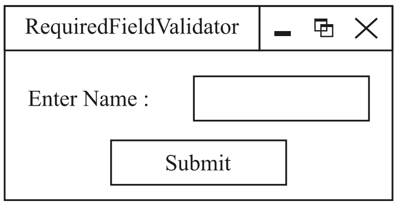
1. RequiredFieldValidator Controls : This control is used to check that the validated control (Textbox) must contain a value. If validated, textbox control empty, RequiredFieldValidator controls shows error message as we mention in text property in validator control. RequiredFieldValidator control simply checks whether something enter into the Web Form element or not.

Syntax :

```
<asp: RequiredFieldValidator runat = “server” ID = “RequiredFieldValidator1”
        ControlToValidate = “ Controlid” ErrorMessage = “*”>
</asp: RequiredFieldValidator>
```

Example :

```
<b>Enter Name: </b>
<asp:TextBox id = “txtname” runat = server>
</asp:TextBox>
<asp: RequiredFieldValidator  id = “reqField_txtname”  runat = server
ControlToValidate = “Red”>
</asp: RequiredFieldValidator>
<asp:Button  id = “submitBtn”  runat = server  Text = “Submit” />
```



2. **CompareValidator :** This control is used to perform a comparison between the values contained in two controls basis of specified operator.

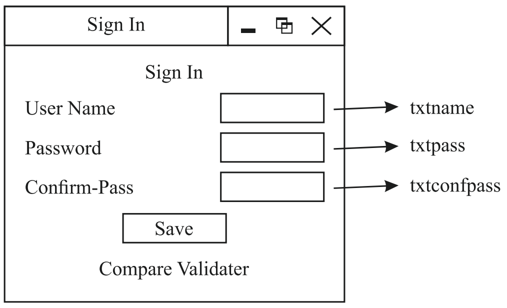
Syntax :

```
<asp: CompareValidator  id = “ CompareValidator1”  runat = server
        ControlToCompare = “firstcontrolid”
        CompareValidate = “SendControlid”  errorMessage = “ErrorMessage” >
</asp: CompareValidator>
```

Some important properties :

- Type - It specifies the data type.
- ControlToCompare – It specifies the value of the input control to compare with. ControlToValidate - It specifies the constant value to compare with.
- Operator - It specifies the comparisons Operator (Equal, NotEqual, LessThan, GreaterThan etc.)

Example :



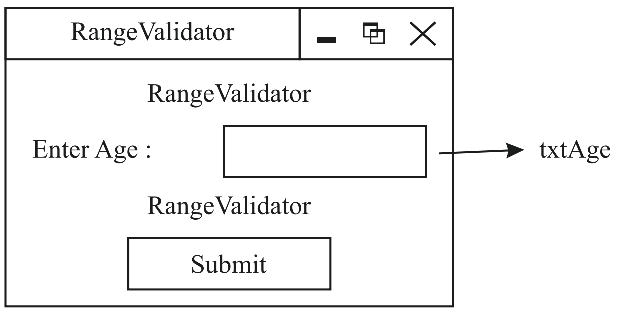
```
<asp:CompareValidator id = "comparevalidator1" runat = "server" Type =
"String" ControlToValidate = "txtconfpass" ControlToCompare = "txtpass"
ErrorMessage = "Password Not Same !!">
</asp:CompareValidator>
```

3. **RangeValidator Control :** The RangeValidator Control is used to check the input control value is within a specifies range or not. It has minimum and maximum value.

Syntax :

```
<asp: RangeValidator  id = "RangeValidator1"  runat = "server" Type = "
RangeValidator1" Type= "Interger" CampareToValidate = "ControlID"
MiximumValue = "High range value" MinimumValue ="Lower range value"
Error Message = "Err.Message" > </asp: RangeValidator>
```

Example :



```
<asp: RangeValidator  id="ageRangeValidator" runat = server  Type = "Integer"
        CompareToValidate = "txtAge" MaximumValue = "35"
        MinimumValue = "30" ErrorMessage = "Not in Range "
        ForeColor = "Red" >
</asp: RangeValidator>
```

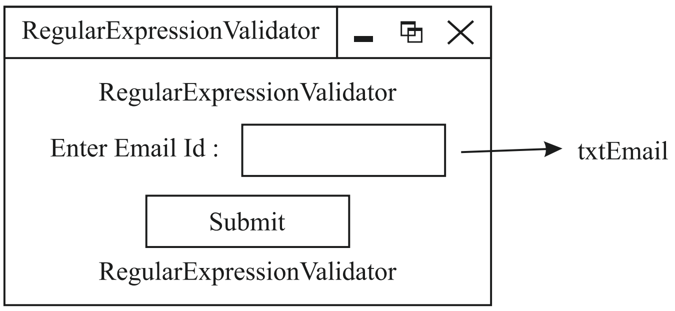
4. **RegularExpressionValidator:** You can check a user input based on a pattern that you define a regular expression. This expression can be phone number, email id, zip code, Internet URL and many more. The regular expression is set in the validationExpression property.

Syntax:

```
<asp: RegularExpressionValidator runat = "server" id =
"RegularExpressionValidator1" ErrorMessage = "Msg" ControlToValidate =
```

"ControlID" ValidationExpression = "expression" > </asp:
RegularExpressionValidator>

Example :


RegularExpressionValidator
RegularExpressionValidator
Enter Email Id :
txtEmail
Submit
RegularExpressionValidator

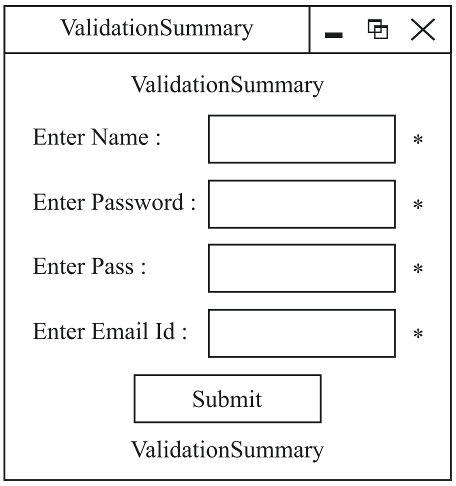
<asp: RegularExpressionValidator runat="server" Id = "txtEmail
RegularExpressionValidator" ControlToValidate = "txtEmail" ErrorMessage =
"Not valid Email" ForColor = "Red" ValidationExpression = "[\w] + @[\w]+
\.(com/net/org/in) " > </asp. RegularExpressionValidator>

5. **ValidationSummary Control :** ValidationSummary Control is reporting control which is used by the other validate control on a page. This control is used to display a summary of all validation error occurred in a web page.

Syntax :

<asp: ValidationSummary id = "vsForm" runat = server DisplayMode = "BulletLiast" ShowSummary = "true" />

Example :



Name is required
Password Mandatory
Confirm-Password Mandatory & should match password

6. **CustomValidator Control :** CustomValidator can be used to execute client-side or server-side validation code. The control allows you to write a method to handle the validation of the value entered. If the validation fails Page.IsValid property is set to false. The client side validation is accomplished through the ClientValidationFunction property. The client side validation routine should be written in a scripting language, like JavaScript or VBScript, which the browser can understand. The server side validation routine must be called from the control ServerValidate event handler. The server side validation routine should be written in any .Net language, like C# or VB.Net.

The basic syntax for the control

```
<asp:CustomValidator ID="CustomValidator1"
    runat="server"
    ClientValidationFunction=”functionName”.
    ErrorMessage="CustomValidator">
</asp:CustomValidator>
```

Example :

```
<script language="javascript">
  function CheckEven(source,arguments)
  {   arguments.isvalide=false;
      if(arguments.value %2 == 0)
      {
        arguments.isvalid=true;
      }
      else
      {
       arguments.isvalid=false;
      }
      return arguments.isvalid;
  }
  </script>
<asp:CustomValidator ID="CustomValidator1" runat="server"
     ClientValidationFunction="CheckEven" ControlToValidate="TextBox5"
     ErrorMessage="Enter a even number" Font-Bold="True"
ForeColor="Blue"></asp:CustomValidator>
```

v. Navigation Web Server Controls :

Page navigation is the process of moving from one page to another page in your website. Asp.Net site navigation enables you to store links to all of your pages in a central location and render those links in the lists or navigation menus on each page.

There are many ways to navigate from one page to another in Asp.Net.

- Client-Side Navigation – (Hyperlink)
- Cross-Page Posting – TextBox obj = (TextBox) Page.PreviousPage.FineControl("TextBox1");
- Cient-Side Browser - Redirect Response.Redirect ("aboutus.aspx");
- Server-Side Transfer - Server.Transfer ("aboutus.aspx");

Navigation control in Asp.Net manages the data passing between Asp.Net pages. Web application is having multiple pages interconnected to each other.

There are three navigation controls in Asp.Net

1. SiteMapPath
2. Menu Control
3. Treeview Control

1) SiteMapPath Control :

A sitemaps is a file where you provide information about the pages, videos and other files on your website and the relationships between them. A sitemaps tells Google which pages and files you think are important in your site and also provides valuable information about these files. You can use the sitemap to provide information about specific type of content on your pages including video, image and news content. Sitemap can improve the crowing of larger or more complex sites or more specialized files.

Need of a sitemap :

- Your site is really large.
- Your site has a large achieve of content pages that are isolated or not well linked to each other.
- Your site is new and has few external links to it.
- Your site has a lot of rich media content (video, images).

Types of sitemaps :-

- Visual sitemap
- HTML sitemap
- XML sitemap
- XML Media sitemap
- RSS Feed (Rich site summary)
- News sitemap

A sitemap is a blueprint of your website that helps search engines find crow1.and index all of your website content. Sitemap also tell search engines which pages on your website are most important. Sitemaps are XML file which are mainly used to describe the logical structure of the web application.

How to add Sitemap file :

- It is an XML file that describes the hierarchical website structure.
- Web sitemap file must be placed in a root of web application.
- Right click the application in the solution Explorer and click add new item. Option after that select the sitemap template and click ok.

Structure of sitemap file :

```
< ? xml version = “1.0”? >
  <sitemap>
     < sitemapNode url = “ “ title “ “  description = “ “ >
      < sitemapNode  url = “ “ title = “ “ />
      < sitemapNode  url = “ “ title = “ “/>
       </ sitemapNode >
       < sitemapNode url = “ “ title “ “ description = “ “ />
              |
            |
        </ sitemap>
```

There is inserting element sitemapNode which we can use to describe the structure of our website.

2) Menu Control :

Menu is a navigation control in Asp.Net which is used to display menu in Web page. It displays two types of menu – static and dynamic menu.

Static Menu : It is always displayed in menu control. By default only menu items at the root levels are displayed.

Home	About us	Research	Contact

Dynamic Menu : It is displayed only when the user moves the mouse pointer over the parent / root menu that contain a dynamic sub menu.

```
Home         About us                Research          contact
             About College
             Principal Message
             Committee
```

Properties of menu control :

- **DataSourceID** – This property is used to specify the data source to be used using sitemap file as data source.
- **CssClass** – This property is used to specify the css attribute for the control.
- **ImgeUrl** – This property is used to specify the image that appears next to the menu item.
- **Orientation** – This property is used to specify the alignment of menu control It can be horizontal or vertical.
- **Tooltrip** – This property is used to specify the tooltrip of menu item when you mouse over.
- **Text** – This property is used to specify the text to display in the menu.
- **NaviagateUrl** – This property is used to specify the target location to send the user when menu item is called.
- **Target** – This property is used to specify the target page location. It can be a new window / Tab or Same window/ Tab.
- **Value** – This property is used to specify the unique id to use in server side events.

Example of menu control :

```
<asp:Menu ID="Menu1" runat="server">
   <Items>
  <asp:MenuItem Text="Home" NavigateUrl="~/Default.aspx"> </asp:MenuItem>
  <asp:MenuItem Text="Products">
         <asp:MenuItem Text="Laptops" NavigateUrl="~/Products/Laptops.aspx">
        </asp:MenuItem>
         <asp:MenuItem Text="Desktops" NavigateUrl="~/Products/Desktops.aspx">
        </asp:MenuItem>
      </asp:MenuItem>
      <asp:MenuItem Text="Contact" NavigateUrl="~/Contact.aspx">
      </asp:MenuItem>
   </Items>
</asp:Menu>
```

3) TreeView Control :

The `TreeView` control in ASP.NET is used to display hierarchical data in a tree-like structure. It is commonly used for navigation menus, file structures, or any scenario where data has a parent-child relationship.

1. **Basic Structure :**

   The `TreeView` control is defined in your ASP.NET markup (.aspx) file. It is usually placed within a <asp:Content> tag within a <asp:ContentPlaceHolder> in a master page or directly in a page if you're not using master pages.

```
 <asp:TreeView ID="TreeView1" runat="server">
     <Nodes>
         <!-- Tree nodes go here -->
     </Nodes>
 </asp:TreeView>
```

2. **Tree Nodes :**

   The tree structure is built using `<asp:TreeNode>` elements. Each node represents an item in the hierarchy. Nodes can have child nodes, creating a hierarchical structure.

```
 <asp:TreeView ID="TreeView1" runat="server">
     <Nodes>
         <asp:TreeNode Text="Root Node">
             <asp:TreeNode Text="Child Node 1"></asp:TreeNode>
             <asp:TreeNode Text="Child Node 2"></asp:TreeNode>
         </asp:TreeNode>
     </Nodes>
 </asp:TreeView>
```

3. **Data Binding :**

   You can bind the `TreeView` control to a data source, such as a database, using the `DataSource` property. This is useful when you have dynamic or large sets of hierarchical data.

```
<asp:TreeView ID="TreeView1" runat="server" DataSourceID="yourDataSource">
    <DataBindings>
        <asp:TreeNodeBinding DataMember="yourDataMember" TextField="NodeText"
ValueField="NodeValue" />
    </DataBindings>
</asp:TreeView>
```

4. **Event Handling :**

   You can handle events like `SelectedNodeChanged` when a tree node is clicked. This allows you to perform actions based on user interaction.

```
          <asp:TreeView ID="TreeView1" runat="server"
                      OnSelectedNodeChanged="TreeView1_SelectedNodeChanged">
              <!-- Nodes go here -->
```

```
        </asp:TreeView>
protected void TreeView1_SelectedNodeChanged(object sender, EventArgs e)
{
   // Your event handling code
}
```

5. **Styles and Appearance :**

   You can customize the appearance of the `TreeView` using various properties such as `NodeStyle`, `ParentNodeStyle`, `LeafNodeStyle`, etc. These properties allow you to control the appearance of different types of nodes.

```
<asp:TreeView ID="TreeView1" runat="server">
   <NodeStyle Font-Bold="true" ForeColor="Blue" />
   <LeafNodeStyle ForeColor="Green" />
</asp:TreeView>
```

: QUESTIONS :

1. Explain the Web Form in ASP.NET?
2. Describe the navigation controls?
3. Enlist and Explain HTML Server Controls?
4. Describe the rich controls?
5. Explain any two Validation Controls with example?
6. Write short notes on ASP.NET Web Server Controls?

***

5. Chapter

ASP.Net Intrinsic Objects and State Management

S y l l a b u s :

HTTP Request Object, HTTP Responce Object, HTTP Server, HTTP Application State Object, HTTP SessionState Object, State Management (View State, Session, Application, Cookies) Global Application Class (global.asax),WebConfig File

(10 L, 20 M)

Introduction to ASP.Net Intrinsic Objects :

ASP .NET provides intrinsic objects to enable low level access to the web application framework. ASP.NET Intrinsic Objects are a set of objects provided by the ASP.NET framework that represent various elements of the web environment. These objects are accessible within ASP.NET applications and are essential for building dynamic and interactive web applications. They encapsulate key aspects of the web request and response lifecycle, enabling developers to interact with the underlying infrastructure easily.

With the help of intrinsic objects you can work directly with the underlying HTTP streams, server, session and application objects. The intrinsic objects can be accessed in a web form through the properties of Page class. The following table shows the intrinsic objects and their mapping to the Page class properties.

Sr. No	Intrinsic objects	Properties of Page Class
1	HTTP Request	Request
2	HTTP Response	Response
3	HTTP Server Utility	Server
4	HTTP Application State	Application
5	HTTP Session State	Session

These intrinsic objects play a crucial role in ASP.NET development, providing developers with a powerful and convenient way to interact with the web environment. Understanding how to effectively use these objects is fundamental for building robust and dynamic web applications with ASP.NET.

HTTP Request Object :

Http Request object represent the incoming request from the client to the web server. The request from the client can come in two ways – GET or post. GET attaches the data to URL. Post cannot attaches the data to URL. The requested page and its details are encapsulated in an Http Request object. The Http Request intrinsic object can be accessed by the request property of the page class. This object is an instance of system.web.HttpRequest class. This object represents the value and property of the HTTP request that caused your page to be loaded. It contains all the Url parameters and all other information sent by the client. It provides access to information about the request, such as form data, query parameters, cookies, and more.

Collections available for Request Object :

1. Cookies – Contains all the cookie values sent in HTTP request.
2. Form – Contain all the form (input) values from a form that uses the post method.
3. QueryString – URL paramerter value from collection is known as QueryString. Contain all variable values in a HTTP query string.
4. ServerVariable – Contain all the server variable values.

Properties of Request Object :

1. TotalBytes :- Returns the total number of bytes the client sent in the body of the request.
2. UserHostAddress :- It request users best address and write using the response objects write method.
3. ApplicationPath :- It requests the application path (which application is used that application path).
4. CurrentExecutionFilePath :- It request the current execution path.
5. filePath :- It request the path to the the file that you are currently working with.
6. HttpMethod :- It get the Http. Method being used.
7. Browser :- It gets the browser that is being used (IE, Chrome, mozila- client used)
8. userAgent :- It gets the information about his computer (client-OS)

By using above request properties we can know the IP address to computer information.

Methods of Request Object :

1. **BinaryRead** :- Retrieves the data sent to the server from the client as part of post request and store it in a safe array. [Used for – client sent data to the server with the half of post method retrieve].

2. **MapPath** :- Retrieves the physical path of a virtual path either absolute or relative or an application – relative path maps too.

Example - 1 :

```
String S1,S2,S3;
S1 = Request.UserHostAddress();
Response.write(S1);
S2 = Request.ApplicationPath();
Response.write(S2);
S3 = Request.Browser.Browser;
Response.write(S3);
```

Example-2 : Here are some examples of using the HTTP request object in ASP.NET :

1. Retrieving Query String Parameters :
```
string parameterValue = Request.QueryString["parameterName"];
```
2. Retrieving Form Data (POST request):
```
string formDataValue = Request.Form["formDataName"];
```
3. Retrieving Cookies:
```
HttpCookie cookie = Request.Cookies["cookieName"];
if (cookie != null)
{
   string cookieValue = cookie.Value;
}
```
4. Retrieving Headers:
```
string userAgent = Request.Headers["User-Agent"];
```
5. Retrieving Server Variables:
```
string serverVariableValue = Request.ServerVariables["ServerVariableName"];
```
6. Retrieving File Uploads:
```
HttpPostedFile file = Request.Files["fileInputName"];
if (file != null && file.ContentLength > 0)
{
   // Process the uploaded file
}
```
7. Checking HTTP Methods:
```
if (Request.HttpMethod == "GET")
{
   // Handle GET request
}
else if (Request.HttpMethod == "POST")
```

```
{
   // Handle POST request
}
```

8. Checking for AJAX Requests :

```
if (Request.Headers["X-Requested-With"] == "XMLHttpRequest")
{
   // Handle AJAX request
}
```

9. Retrieving the URL :

```
string url = Request.Url.ToString();
```

10. Checking for HTTPS:

```
bool isHttps = Request.IsSecureConnection;
```

HTTP Response Object :

In ASP.NET, the HTTP response object is an instance of the HttpResponse class, which represents the server's response to a client's request. It provides methods and properties that allow you to control the output that the server sends to the client. The Http Response object represents the response sent back to the client from the web server. The Response property of the page class provides access to Http Response object. In the web server world, a response is exactly opposite to the request. A Response is the message sent from web server to the client when client make a request. For each request from client, the server gives a response unless there is an error.

How does a browser process the responses?

1. The Response includes several information about the page requested including the cookies to be saved on the client machine the actual content to be displayed.
2. The browser accepts the response and processes it.
3. Browser does several things including saving the cookies, checking security etc. and then displays the page content to the user.

The Asp.Net provides a classed called Http Response which is defined in the namespace system.web. This class provides various methods and properties which help you use various information related to a web response. An instance of this class is created by default in all pages, so that you can use this object without creating again each time in all pages.

Collections of Response Object :

1. **Cookies** : Sets a cookies value. If the cookies do not exist, it will be created and take the value that is specified.

Properties of Response Object :

1. **Buffer** :- Specifies whether to buffer the page output or not.

2. **CacheControl** :- Sets whether a proxy server can cache the the ouput or not.
3. **Expiers** :- Set hoe long (in minutes) a pge will be cached on a browser before it expires.
4. **CharSet** :- Append the name of character- set to the content- type header in response object (which character-set our web page UTF-F, ASCII)
5. **ExpiresAbsolute** :- Sets a date and time when a page cached on a browser will expire.
6. **IsClientConnected** :- Indicates if the client has disconnected from the server.
7. **Status** :- Specifies the value of the status line returned by the server (particular request status.)

Method of Response Objects :

1. **Clear** : Clears any buffered HTML output.
2. **End** : Stops processing a script and returns the current result.
3. **Flush** : Sends buffered HTML output immediately.
4. **Redirect** : Redirects the user to the different URL
5. **Write** : Writes a specified string to the output without any character conversion.

Example-1 :

```
Response.Redirect ( http://www.nmu.ac.in);
Response.Write (“Harpal”);
```

Example-2 : Here are some commonly used features of the `HttpResponse` class with examples :

1. **Setting Content Type :** You can set the content type of the response to specify the type of data being sent.
   ```
   Response.ContentType = "text/html";
   ```
2. **Writing to the Response Stream:** You can write content directly to the response stream.
   ```
   Response.Write("Hello, World!");
   ```
3. **Redirecting to Another Page:** You can redirect the client to another page.
   ```
   Response.Redirect("~/NewPage.aspx");
   ```
4. **Setting Cookies :** You can set cookies to be sent to the client's browser.
   ```
   HttpCookie cookie = new HttpCookie("UserName", "Purva H Patil");
   Response.Cookies.Add(cookie);
   ```
5. **Setting Headers:** You can set custom headers in the response.
   ```
   Response.AddHeader("CustomHeader", "CustomValue");
   ```
6. **Setting Status Code:** You can set the HTTP status code for the response.
   ```
   Response.StatusCode = 404;
   ```
7. **File Download:** You can prompt the user to download a file.
   ```
   Response.ContentType = "application/octet-stream";
   Response.AppendHeader("Content-Disposition", "attachment; filename=example.txt");
   ```

```
Response.TransmitFile(Server.MapPath("~/Files/example.txt"));
Response.End();
```

8. **Output Caching:** You can cache the output for a specific duration.
   ```
   Response.Cache.SetCacheability(HttpCacheability.Public);
   Response.Cache.SetExpires(DateTime.Now.AddMinutes(10));
   ```
9. **Clearing Content:** You can clear the content of the response.
   ```
   Response.Clear();
   ```
10. **Buffering Output:** You can enable or disable output buffering.
    ```
    Response.BufferOutput = true; // or false
    ```

HTTP Server Object :

In ASP.NET, an HTTP server object is an instance of the HttpServerUtility class, which provides methods for performing various server-side tasks during the processing of an HTTP request. The HttpServerUtility class is part of the System.Web namespace.

The HttpServerUtility Object contains Utility, Methods and properties to work with the web server. The server Property of the page class provides access to the HttpServerUtility Object. The server object is used to access properties and methods on the server.

Properties of HttpServerUtility Object :

1. **MachineName** : It provides the name of computer on which page is running. It Returns the name of server that hosts the web Application.
2. **ScriptTimeout** : Sets or returns the maximum number of seconds a script can run before it is terminated.

Methods of HttpServerUtility Object :

1. ClearError():- Clears the last exception from memory.
2. CreateObject():- Creates a COM object on the server.
3. Execute():- Execute an Asp.Net page within the current request page.
4. GetLastError():- Returns the last exception that occurred on the web server.
5. Tranfer():- Allow the tranfer of Aspx page exception from the current page to another ASPX page on the same web server.

Examples : Here are some examples of using the HTTP server object in ASP.NET:

Example 1 : Server.MapPath

`Server.MapPath` is used to convert a virtual path to a physical path on the server. This is useful, for example, when working with file operations.

```
string virtualPath = "~/Content/Images/myImage.jpg";
string physicalPath = Server.MapPath(virtualPath);
Response.Write("Physical Path: " + physicalPath);
```

Example 2 : Server.Transfer

`Server.Transfer` is used to transfer the execution of the current page to another page on the server.

```
string destinationPage = "DestinationPage.aspx";
Server.Transfer(destinationPage);
```

Example 3: Server.Execute

`Server.Execute` is similar to `Server.Transfer`, but it allows the execution of another page without ending the current page's execution.

```
string destinationPage = "DestinationPage.aspx";
Server.Execute(destinationPage);
```

Example 4: Server.UrlEncode and Server.UrlDecode

These methods are used for URL encoding and decoding.

```
string originalString = "Hello, World!";
string encodedString = Server.UrlEncode(originalString);
string decodedString = Server.UrlDecode(encodedString);
Response.Write("Encoded String: " + encodedString);
Response.Write("<br/>");
Response.Write("Decoded String: " + decodedString);
```

Example 5: Server.HtmlEncode and Server.HtmlDecode

These methods are used for HTML encoding and decoding.

```
string originalString = "<b>Hello, World!</b>";
string encodedString = Server.HtmlEncode(originalString);
string decodedString = Server.HtmlDecode(encodedString);

Response.Write("Encoded String: " + encodedString);
Response.Write("<br/>");
Response.Write("Decoded String: " + decodedString);
```

HTTP Application State Object:

In ASP.NET, the HTTP Application State is a way to store and share global data across all users and sessions of an application. It provides a centralized location for storing data that should be accessible by all parts of the application. The data stored in the Application State is available throughout the lifetime of the application and is shared among all users.

Http Application State is used to store application data typically does not change. Application state is data repository available to all classes. Application state is stored in memory on the server and is faster than storing and retrieving information in database. Global data can be stored in the

HttpApplicationState class. An instance of the HttpApplicationState class is created the first time any client request. The application property of the page class provides access to the HttpApplicationState Object. Information stored in HttpApplicationState object is stored for the life of the application.

Collections of Application State Object :

1. **AllKeys** : Gets the access keys in the HttpApplicationState Collections.
2. **Contents** : Gets a reference to the HttpAppllicationState object.
3. **StaticObject** : Gets all objects declared by an <object> tag where the scope is set to "Application" which the Asp.Net application.

Properties of Application State Object :

1. Count : Gets the number of objects in the HttpApplicationState collection.
2. Item :
   a. Item [Int32] – Get a single HttpApplicationState object by index.
   b. Item [String] – Get the value of the HttpApplicationState object by name.

Methods of Application State Object :

1. Add():- Add new object to HttpApplicationState collections.
2. clear():- Removes all objects from collection.
3. Get(Int32) :- Get object by numeric index.
4. Get(string):- Get object by name.
5. GetKey(Int32):- Get object name by index.
6. Lock():- Lock access to an HttpApplicaionState variable.
7. Unlock():- Unlock access to an HtttpApplicationState variable.
8. Set():- Update the value of collections.

Examples : Here's how you can use the HTTP Application State object in ASP.NET with examples

1. **Storing Data in Application State :** You can store data in the Application State using the `Application` object, which is accessible from the `HttpContext.Current` property. Here's an example:
   ```
   // Storing data in Application State
   HttpContext.Current.Application["AppName"] = "MyASPApp";
   HttpContext.Current.Application["Version"] = 1.0;
   ```
2. **Retrieving Data from Application State:** To retrieve data from the Application State, you can use the same `Application` object:
   ```
   // Retrieving data from Application State
   string appName = HttpContext.Current.Application["AppName"].ToString();
   double version = Convert.ToDouble(HttpContext.Current.Application["Version"]);
   ```

3. **Updating Data in Application State:** You can update the data in Application State just by assigning new values:

```
// Updating data in Application State
HttpContext.Current.Application["Version"] = 2.0;
```

4. **Removing Data from Application State:** To remove data from Application State, you can use the `Remove` method:

```
// Removing data from Application State
HttpContext.Current.Application.Remove("Version");
```

   Alternatively, you can use the `Clear` method to remove all items from the Application State:

```
// Clearing all data from Application State
HttpContext.Current.Application.Clear();
```

5. **Example - Counter Incrementing on Each Request:**

```
  protected void Page_Load(object sender, EventArgs e)
  {
     if (HttpContext.Current.Application["Counter"] == null)
     {
        HttpContext.Current.Application["Counter"] = 1;
     }
     else
     {
        int counter = Convert.ToInt32(HttpContext.Current.Application["Counter"]);
        counter++;
        HttpContext.Current.Application["Counter"] = counter;
     }
     // Display the counter value on the page
     Response.Write($"Application Counter:
{HttpContext.Current.Application["Counter"]}");
  }
```

HTTP Session State Object :

In ASP.NET, you can use the Session object to work with session state. In ASP.NET, the HTTP session state is a mechanism that allows you to store and retrieve user-specific information across multiple HTTP requests. It enables you to persist data between different pages in a web application for a specific user. The session state is maintained on the server, and a unique identifier (usually stored in a cookie) is used to associate the session data with a particular user.

The HttpSessionState class is used primarily for storing and accessing data that is shared across all the pages accessed by a particular user during a given session of interacting with the application. The Session properly of a page class provides access to HttpSessionState object. Each user session in Asp.net is identified by a unique session ID which is created at the same time as the user session.

Here are some of the common properties, methods, and collections associated with the `Session` object:

Properties Session State Object :

1. **SessionID:** Gets the unique identifier for the current session.
   string sessionId = Session.SessionID;
2. **IsNewSession:** Indicates whether the session was created during the current request.
   bool isNewSession = Session.IsNewSession;
3. **Timeout:** Gets or sets the timeout period for the session (in minutes).
   int sessionTimeout = Session.Timeout;

Methods Session State Object :

1. **Add:** Adds a new item to the session state.
   Session.Add("UserName", "JohnDoe");
2. **Remove:** Removes the specified item from the session state.
   Session.Remove("UserName");
3. **Clear:** Removes all items from the session state.
   Session.Clear();

Collection Session State Object :

The `Session` object can be used as a collection to store and retrieve values:

```
// Storing a value in the session
Session["UserName"] = "Purva H Patil";
// Retrieving a value from the session
string userName = (string)Session["UserName"];
```

Example : Here's a simple example demonstrating the use of the `Session` object in an ASP.NET page:

```
// Storing a value in the session
Session["UserName"] = "Purva H PAtil";
// Retrieving and displaying the value from the session
string userName = (string)Session["UserName"];
Response.Write("Welcome, " + userName);
```

Make sure that session state is enabled in your application. You can enable it in the `Web.config` file:

```xml
<configuration>
        <system.web>
                <sessionState mode="InProc" timeout="20" />
        </system.web>
</configuration>
```

In this example, `mode="InProc"` indicates that session state is stored in-process, and `timeout="20"` sets the session timeout to 20 minutes.

State Management :

Web page developed in Asp.Net is HTTP based. Http protocol is stateless protocol. It means that web server does not have any idea about the request from where they coming i.e. from same client or new client. On each request web pages are created and destroyed. So how do we make web pages in Asp.Net which will remember about the user, would be able to distinguish between old clients requests and new clients requests and user previous filled information while navigating to other web page in website?

Solution of the above problem is state management in Asp.Net. State management is the process of maintain the state of values between multiple request of the page. State management maintains and stores the information of any user till the end of the user session. To maintain the state of values, Asp.Net provides us different option where those values can be maintained either client machine or the server machine.

Asp.net technology offers following state management techniques :

1. Client Side State Management
   - i. Cookies
   - ii. Hidden Fields
   - iii. View State
   - iv. Query string
2. Server Side State Management
   - i. Session State
   - ii. Application State

1. **Client-side State Management** : Client-side state management in ASP.NET refers to the techniques used to store and manage data on the client's browser, allowing information to persist between page requests. Whenever we use client side State Management the State related information will directly get stored on the client side.

i. **Cookies :** Cookies are client-side state management techniques that store small pieces of data on the user's machine. They are suitable for preserving data between requests and sessions. Cookies are sent to the client's browser and can be retrieved on subsequent requests.

Cookies are small piece of the information (up to 4kb) which is stored on the client's computer by the browser. Cookies are always sent with the request to the web server and information can be retrieved from the cookies at the web server. A cookie does not use server memory. It may contain username, ID, Password or any information. They are limited to 4kb in size. There are two types of cookies, Persistence Cookie and Non-persistence Cookie.

1) **Persistence Cookie :** This types of cookies are permanently stored or user hard drive. Cookies which have an expiry date time are called Persistence Cookie. These types of cookies stored user hard drive permanently till the date time we set. It is saved as a text file in the file system of the client computer usually under Temporary Internet file folder.

   Example : 1 - with the help of response object we can create a cookies in Asp.Net.

   The Response.Cookies command is use to create cookies and it must appear before <html> tag.

```
<%
        Response.Cookies (“username”) = “Harpal”;
%>
```

   Setting a date time when the cookies should expire.

```
<%
        Response.Cookies (“username”) = “Harpal”;
        Response.Cookies (“username”) . Expires = # April 10,2024#;
%>
                              OR
<%
        Response.Cookies (“username”) = “Harpal”;
        Response.Cookies (“username”) . Expires =

DateTimeNow.AddMinute (10);
%>
        After 10 minute username cookies automatically expires.
```

   Example to Retrieve Cookies : Request.Cookies is used to retrieve cookies value.

```
Lable 1.Text = Request.Cookies (“username”).value;
```

   Example : 2 - We can also create cookies using HttpCookie class.

```
HttpCookie username = new HttpCookie (“user”);
```

```
username . Value = “Harpal”;
username . Expire = DateTime.Now.AddMinutes (10);
Response.Cookies.Add (username);
```

Example to Retrieve Cookies :

```
HttpCookie reqCookie = Request.Cookies (“username”);
If (reqCookie ! = null)
{
        Lable1.Text = reqCookie (“username”).ToString ();
}
```

2) **Non-Persistence Cookies :** These types of cookies are not permanently stored on user hard drive. It is also called as session cookies or In-memory cookies are saved only while your web browser is running. They can be used by a web server only until you close your browser i.e. automatically deleted.

Example: 1- Create:-

```
Response.Cookies (“name”) . Value = “Harpal”;
                                        OR
HttpCookie StrName = new HttpCookie (“name”);
StrName . Value = “Harpal”;
Response . Cookie . Add (StrName);
```

Example: 2 - Read Cookie Information;-

```
if(Request . Cookies (“name”) != null)
{
        Lable1 . Text = Request . Cookies (“name”). Value;
 }
```

ii. **Hidden Fields** : Hidden fields are a control provided by ASP.NET, which is used to store small amount of data on the client. Hidden fields are not rendered to the browser and it is invisible on the browser.

Syntax :

<asp:Hiddenfield ID = “Hiddenfield1” runat = “server” />

This is non-visual control in Asp.Net where you can save the value. Hidden fields are not encrypted or protected and can be change by anyone. However from security point of view this not supported. Asp.Net uses Hidden Field control for managing the view state. So don’t store any important or confidential data like password and credit card details in this control.

Example : HiddenFiledDemo.aspx

```
<asp: Hiddenfield ID = “HiddenFieldDateTime” runat = “server” />
<asp: Lable ID = “lbl1” runat = “server”> </asp: Label >
```

HiddenFieldDemo.aspx.cs

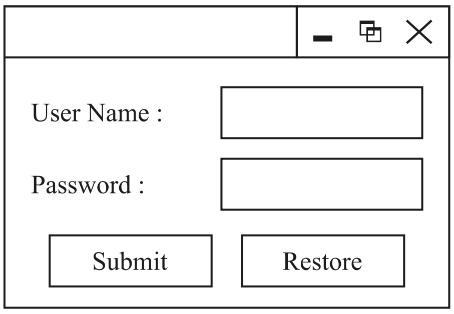
```
Protected void Page_Load ( Object sender, EventArgs ex)
{        // Set value
         HiddenFieldDateTime.Value = DateTime.Now.ToString();
         // Get value
         lbl1.text = HiddenFieldDateTime.value.ToString();
}
```

iii. **View State** : View State is a client-side state management technique that allows the preservation of page and control values between postbacks. It is suitable for preserving small amounts of data, such as control values, between round trips to the server. View State data is stored in a hidden field on the page and is sent to the client as part of the HTML.

View State is used to store user data. It is used to manage Page_level State and is used for storing, sending and receiving information as long as the user is on the current page the state will be available, once the user redirects to the next page, and the current state will be lost. We can store small values in View State will be lost. We can Store small values in View State. If you store more data the page is heavy. View State is one of the methods of Asp.Net Page framework used to preserve and store the page and control values between round trip. It is maintained internally as a hidden field in the form of encrypted value and a key.

Example : Using with View State and without View State

Login.aspx



```
<form id = "form1" runat = server>
        userName:
        <asp: textbox id = "userName" runat = "Server" > </asp: textbox >
        <br />
        Password:
                <asp: textbox id = “Pass” runat = “server”> </asp: textbox>
                <br />
                <asp: button id = “submitbtn” runat = “server” text = “Submit” OnClick =
“Submitbtn_Click” />
                <asp: button id = “Restorebtn” runat = “server” text = “Restore” OnClick
= “Restorebtn_Click” />
        </form>
```

Login.aspx.cs (Without View State)

```
Public string S1, S2 ;
Protected void submitbtn_Click (object sender, EventArgs e)
{
        S1 = userName.Text ;
        S2 = Pass.Text;
        userName.Text = Pass.Text = String.Empty;
}
Protected void Restorebtn_Click (Object Sender, EventArgs e)
{
        userName.Text = S1;
        Pass.Text = S2 ;
}
```

Login.aspx.cs (Without View State)

```
Protected void Submitbtn_Click (object sender, EventArgs e)
{
        ViewState [“userName”] = userName.Text ;
        ViewState [“Pass”] = Pass.Text ;
        userName.Text = Pass.Text = String.Empty ;
}
Protected void Restorebtn_Click ( Object Sender, EventArgs e)
{
        If (ViewState [“userName”] != null )
        {
                userName. Text = viewstate [“userName”].ToString() ;
        }
        If ( viewstate [“Pass’] != null )
        {
                Pass.Text = viewstate [ “Pass”] .ToString() ;
        }
}
```

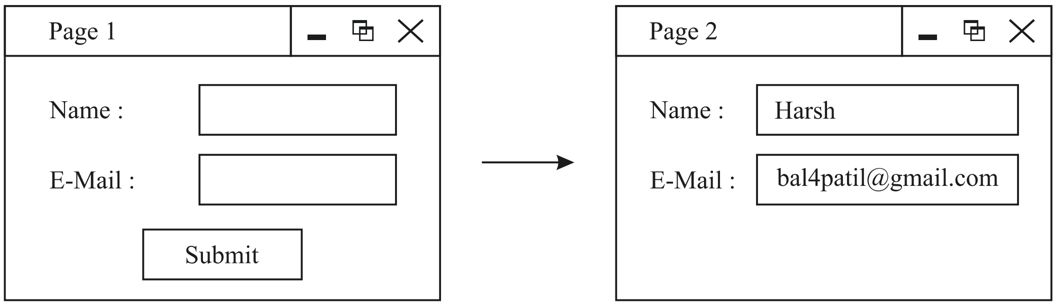
iv. **Query String** : Query Strings are client-side state management techniques that pass data between pages as part of the URL. They are suitable for passing small amounts of data between pages. Data is appended to the URL as key-value pairs.

Query string is used to pass the values form one page to another page. Query string will display the data within the Browser URL so it is not recommended to send sensitive information like password using Query String. The maximum capacity of the browser is 4kb we cannot send more than 4kb of information using query string. The ? (Question Mark) indicating the beginning of Query String and its value. It is possible to use more than one query string. The First Query String is specified using ? Subsequent query string can be appended to the URL using the & (ampersand) symbol.

Syntax :

1) To Pass value by using Query String –
Response.Redirect ("desturl ? querystringName1 = Value & querystringName2 = value");
2) To Read Value by using Query String –
Request.QueryString ["querystringName1"];

Example :



Page1.aspx.cs

```
Protected void Submitbtn_Click (Object Sender, EventArgs e)
{
        String namevalue, Emailvalue ;
        namevalue = nametxt.Text.ToString() ;
        Emailvalue = Emailtxt.Text.ToString() ;
        Response.Redirect ("~/Page2.aspx?Name = namevalue & Email = Emailvalue" ) ;
}
```

Page2.aspx.cs

```
Protected void Page_Load (Object Sender, EventArgs e)
{
        Namelbl.Text = Request.QueryString [“name”].ToString();
        EmailIDlbl.Text = Request.QueryStrig [“Email”]. ToString() ;
}
```

2. Server-side State Management :

Server-side state management in ASP.NET involves storing and managing data on the server to maintain information between multiple requests from the same user or different users. ASP.NET provides various mechanisms for server-side state management, each with its own characteristics and use cases.

Whenever we use server side state management the state related information will directly get stored on the server side. Server-Side for storing page information typically has higher security than client side state management. The types of Server-Side state management techniques are Session State and Application State.

i. **Session State** : Session State is a server-side state management technique that allows you to store and retrieve user-specific information for the duration of a user's session. It is suitable for maintaining state across multiple requests during a user's visit to a website. Session data is stored on the server, and a session identifier is sent to the client, usually as a cookie or in the URL. Session state management is very strong technique to maintain state. Generally session is used to stored user. The server maintains the state of user information by using a session ID. We should not store any type of matter data or application data in session. Session is very user specific, once user logout the session is destroyed and all data for that particular session gets clear. For each and every user, a separate session is created and each and every session has its unique ID. The Session has a default timeout value (20 minute). We can also set the timeout value for a session in the web.config file.

Syntax :

1) **Store information in session object**

```
Session [“SessionName”] = “user Information”;
```

2) **Read from Session object**

```
Variable / Control = Session [“SessionName”];
```

Session Event in ASP.NET : To manage a Session Asp.Net provides two events Session_Start and Session_end.

1. **Session_Start :** This event is raised every time a new user makes a request without a Session ID.

```
Void Session_Start ( Object Sender, EventArgs e)
{
        Session [“count”] = 0 ;  //ur code
}
```

2. **Session_end :** This event is raised when session ends either because of a timeout expiry or explicitly by using session.Abandom().

Example :

Session [“userID”] = userIDtxt.Text ;

userIDlbl.Text = Session [“userID”].ToString();

ii. **Application State** : Application State is a server-side state management technique for storing data that is shared among all users of an application. It is suitable for storing global data that needs to be accessible to all users. Application data is stored on the server and is accessible to all users of the application.

The data stored in Application state is common for all users of that particular Asp.Net application and can be access to anywhere in the application. Data stored in the application state should be small size. It stores information as a Dictionary collection in key-value pairs. This value is accessible across the pages of the application..

Syntax:

1) Store information in Application Object

    Application [“PageTittle”] = “BASPONC”;

2) Read information from Application

    Page.title = Application [“PageTitle”].ToString();Example:

Application Event in ASP.NET : Application Event is written in a special file called Global.aspx. This file is not created by default; it is created explicitly by developer in the root directory. There are three types of events in Asp.Net.

- Application_Start
- Application_Error
- Application_End

1. **Application_Start:** This event is raised when am application domain starts. When the first request is raised to an application then the Application_Start event is raised.

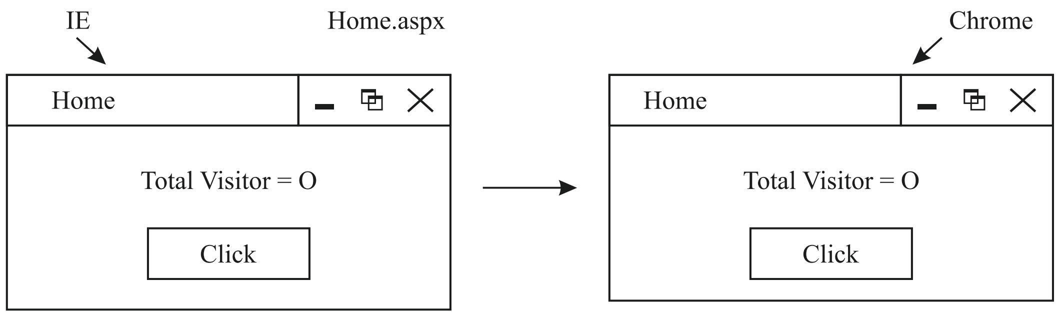
```
Void Application_Start ( Object Sender, EventArgs e)
{
        Application [“count”] = 0;
}
```

2. **Application_Error:** It is raised when an unhandled exception occurs. And we can manage the exception in this event.

```
Void Application_Error ( Object Sender, EventArgs e)
{
        //ur  exception managae code
}
```

3. **Application_End:** This event is raised just before an application domain ends because of any reasion may server restarting.

   Example:



Home.aspx.cs

```
Protected void Clickbtn_Click ( Object Sender, EventArgs e)
{
        int count = 0;
        If (Application [“visit”] != null)
        {
                Count = Convert.ToInt32 ( Application [“visit”].ToString());
        }
        Application [“visit”] = count1;
        Label1.Text = “Total visitor = “ + count.ToString() ;
}
```

Global Application Class (global.asax) :

The Global Application Class, often represented by the Global.asax file in an ASP.NET web application, serves as a central location to handle application-level events and tasks. This file contains code for event handlers that are triggered during various stages of the application lifecycle. The Global.asax file is automatically detected and executed by the ASP.NET runtime.

Global.aspx is also known as Application File in Asp.Net, The File is used to handle application level and session-level events. If you need any particular task to be carried out in any of those events, we can write code there as per need. The Global.aspx file allows you to write code that responds to global application events. These events fire at various points during the lifetime of a web application. This file resides in the root directory of an Asp.Net based application.Global.aspx file are never called directly by user, they are called automatically in response to application events.

Point to remember about global.aspx.

- They do not contain any HTML or Asp.Net tags.
- They defined methods for single class, application class.
- They are optional but a web-application has no more than one global.aspx file.
- Global.application class in asp.net is defined into the file global.aspx.cs
- This class cotains event handlers which are fire for the different stages of the application and request life cycle.
- Global.aspx contains a class that is inherited from system.web.HttpApplication class which has all events for application life cycle.

Global.aspx can handle two types of application events.

1) **Event that occurs only under specific condition i.e. request / response events :**
    - Application_BeginRequest() This method is called at the start of every request.
    - Application_AuthenticateRequest() This method is called just before authunting is performed.
    - Application_AuthorizeRequest() After the user is authenticated (identified) it's time to determine user permissions.
    - Application_EndRequest() This method is called at the end of the request, just before the objects are released Its suitable point for cleanup code.
2) **Events that don't get fired with every request :**
    - Application_Start() This event occurs when the application starts which is the first time. It receives a request from any users. It does not occur on subsequent requests. This event is commonly used to create some initial inforamtion that is used for later.
    - Application_End() This event occurs when the application shutting down you can insert cleanup code.
    - Application_Error() This event occurs when an unhandled exception occurs in the application.
    - Session_Start() This event occurs whenever a new user request is received and a session is started.
    - Session_End() This event oocurs when a session time outs or is programmatically ended (typically 20 min).

How to add Global.aspx file :

Select website -> Add New File and choose the Global Application class template.

```
< % Application Language = " C#"%>
 <Script runat = "Server" >
        Void Application_Start ( Object Sender, EventArgs e)
        {
                // Code that run on application startup.
        }
        Void Application_End ( Object Sender, EventArgs e)
        {
                //Code that runs on Application shutdown.
        }
        Void Application_Error ( Object Sender, EventArgs e)
        {
                //Code that runs when an unhandled error oocures.
        }
        Void Session_Start ( Object Sender, EventArgs e)
        {
                // Code that runs when a new session is start
        }
        Void Session_End ( Object Sender, EventArgs e)
        {
                //Code that runs when a session ends.
        }
</ Script>
```

Web Configuration File (Web.config) :

In ASP.NET, the Web.config file is a configuration file that is used to store settings and configuration information for a web application. It is an XML file that resides in the root directory of an ASP.NET web application. The Web.config file is used to configure various aspects of the web application, such as authentication, authorization, custom error pages, session state, and more.

Every web application includes a web.config file that configures fundamental settings. The web.config file uses a predefine XML format. This file is responsive for controlling the application behavior. Generally a website contains a single web.config file stored inside the application root directly. Asp.Net configuration system is used to describe the property and behaviors of various aspects of Asp.Net application.

Benefits of Web.config file :

1. Asp.Net configuration system is extensive and application specific information can be stored and retrieved easily.
2. It is human readable.
3. You need not restart the web server when the setting is changed in Web.config file Asp.Net automatically depends the changes and applies then to the running Asp.Net application.
4. You can use any standard text editor or XML parser to create and edit web.config file.
5. They are never locked – you can update these file setting at any point even while your application is running.

Web.Config file contains :

There are numbers of important settings that can be stored in the configuration file. The entire content of the file is nested in root <configuration> element. Web.config file is case-sensitive. Some of the most frequently used configuration stored conveniently inside web.config file is as follows :

- Database Connections,
- Caching settings,
- Session State,
- Error Handling,
- Security,
- App Settings, etc.

These are just a few examples, and the Web.config file can contain many other configuration settings based on the requirements of your ASP.NET application. It plays a crucial role in controlling the behavior and settings of the web application.

Example :-

```
    <?xml version = “1.0”?>
  <Configuration >
            <configSections>
                    ---
                    ---
            </configSections>
            <appSetting>
                    ---
                    ---
```

```
        </appSetting>
        <connectionStrings>
                ---
                ---
        </connectionStrings>
        <system.web>
                ----
                ---
        </system.web>
                |
                |
                |
</Configuration >
```

: PRACTICE :

1. **Create an ASP .NET web application to demonstrate use of global.asax file.**

   Step – 1 Create a new ASP.NET Web Application:

   Open Visual Studio and create a new ASP.NET Web Application. Choose the template that fits your needs (e.g., ASP.NET Web Forms, ASP.NET MVC, etc.).

   Step – 2 Add Global.asax File:

   Right-click on your project in Solution Explorer, choose "Add," and then select "Global Application Class." This will add a file named `Global.asax` to your project.

   Step – 3 Modify Global.asax:

   Open the `Global.asax` file and add code for handling some of the global events.

```
<%@ Application Language="C#" %>
<script runat="server">
    void Application_Start(object sender, EventArgs e)
    {
        // Code that runs on application startup
        Application["TotalRequests"] = 0;
    }
    void Session_Start(object sender, EventArgs e)
    {
        // Code that runs when a new session is started
        Application.Lock();
```

```
            Application["TotalRequests"] = (int)Application["TotalRequests"] + 1;
            Application.UnLock();
        }
        void Application_Error(object sender, EventArgs e)
        {
            // Code that runs when an unhandled error occurs
            Exception ex = Server.GetLastError();
            // Log the error or perform other error handling tasks
        }
        void Session_End(object sender, EventArgs e)
        {
            // Code that runs when a session ends (e.g., user logs out or session expires)
        }
        void Application_End(object sender, EventArgs e)
        {
            // Code that runs on application shutdown
        }
    </script>
```

Step – 4 Run the Application:

Run your application, and the `Global.asax` events will be triggered as you navigate through the application.

2. **Write an ASP .net program that demonstrates use of Intrinsic Objects.**

Default.aspx:

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

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>ASP.NET Intrinsic Objects Example</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <h2>ASP.NET Intrinsic Objects Example</h2>
            <p>Visit this page with a query string parameter, e.g., ?name=John</p>
            <hr />
            <asp:Label ID="lblOutput" runat="server" EnableViewState="false" />
        </div>
    </form>
</body>
</html>
```

Default.aspx.cs (Code-Behind):

```
using System;
namespace demoProgram
{
    public partial class Default : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            // Request Object Example
            string nameFromQueryString = Request.QueryString["name"];

            // Response Object Example
            Response.Write("Hello from Response object!<br />");

            // Session Object Example
            Session["username"] = "John";

            // Server Object Example
            string appDataPath = Server.MapPath("~/App_Data");

            // Application Object Example
            Application["totalVisits"] = ((int)(Application["totalVisits"] ?? 0)) + 1;

            // Display output in a Label
            lblOutput.Text = $@"
                <h3>Request Object</h3>
                <p>QueryString[\"name\"]: {nameFromQueryString}</p>

                <h3>Response Object</h3>
                <p>Hello from Response object!</p>

                <h3>Session Object</h3>
                <p>Session[\"username\"]: {Session["username"]}</p>

                <h3>Server Object</h3>
                <p>MapPath(\"~/App_Data\"): {appDataPath}</p>

                <h3>Application Object</h3>
                <p>Application[\"totalVisits\"]: {Application["totalVisits"] ?? 0}</p>
            ";
        }
    }
}
```

3. **Create an ASP .NET web application to demonstrate use of session and cookies.**

Default.aspx:

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

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Session and Cookies Example</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <h2>Session and Cookies Example</h2>
            <p>Enter your name:</p>
            <asp:TextBox ID="txtName" runat="server"></asp:TextBox>
            <asp:Button ID="btnSubmit" runat="server" Text="Submit" OnClick=
"btnSubmit_Click" />
        </div>
    </form>
</body>
</html>
```

Default.aspx.cs:

```
using System;
namespace demoProgram
{
    public partial class Default : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!IsPostBack)
            {
                // Check if there is a stored username in the session
                if (Session["username"] != null)
                {
                    // Redirect to the welcome page if the session already contains a username
                    Response.Redirect("Welcome.aspx");
                }
            }
        }

        protected void btnSubmit_Click(object sender, EventArgs e)
        {
            // Store the entered username in the session
            Session["username"] = txtName.Text;

            // Create a cookie to remember the username
            HttpCookie cookie = new HttpCookie("UsernameCookie");
            cookie.Value = txtName.Text;
            cookie.Expires = DateTime.Now.AddDays(7); // Cookie expiration (7 days)
            Response.Cookies.Add(cookie);

            // Redirect to the welcome page
            Response.Redirect("Welcome.aspx");
        }
    }
}
```

Welcome.aspx:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Welcome.aspx.cs"
Inherits="demoProgram.Welcome" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Welcome Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <h2>Welcome Page</h2>
            <p>Welcome, <%= Session["username"] %>!</p>
            <p>Remembered username from cookie: <%=
Request.Cookies["UsernameCookie"]?.Value %></p>
        </div>
    </form>
</body>
</html>
```

Welcome.aspx.cs:

```
using System;
namespace demoProgram
{
    public partial class Welcome : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            // Check if there is a stored username in the session
            if (Session["username"] == null)
            {
                // Redirect to the default page if the session doesn't contain a username
                Response.Redirect("Default.aspx");
            }
        }
    }
}
```

: QUESTIONS :

1. What is Intrinsic Objects? Explain its Types?
2. Write short note on following Intrinsic Objects
   - Response Object
   - Request Object
   - Application State Object
   - Session State Object
3. Explain State Management in details? Explain the Cookies in ASP.NET?
4. Write short notes on Cookies?
5. Write short notes on Session?
6. Explain the View State and Query String with example?
7. Write notes on Global Application Class?
8. Explain the Web Configuration File in ASP.NET?

***

6. Chapter

Master Page & ADO .NET

S y l l a b u s :

Master pages and content Pages Basic, Architecture of ADO.NET, Create and retrieve data from Database Connections, SqlDataSource Controls, ASP.NET Data-Bound Controls (GridView, Repeater, DataList, Details View, Form View)

(15 L, 20 M)

Introduction to Web Design:

Web design is a concept of planning, Creating and maintaining websites. Web Design is the very process of using creativity to design and construct a website and updating it regularly. Besides the creation and updating, this concept also involves taking care of UI, the architecture of information present, the layout the colors, content, navigations as well as the designs of the various icons. Some other areas in web design include search engine optimization, user experience design, and graphic design etc. Web designing is a process that can be done by anyone who has the right knowledge of the various disciplines involved.

The Basic elements of good web design are-

- Shape - on most websites and webpages the shapes cred are squares or rectangular Shapes are responsible for the creation of an enclosed boundary in the overall design.
- Texture - It is one element that can help provide your website with a feeling of a Surface. This element must be used in such a way that it brings out the content given on the website and makes it look more.
- Direction – It is the element of web design which is responsible for lending it movement or motion.
- Color - Another basic element of good web- -site is the case of color. The color are added in the later stage and not during the designing.

Web Design Principles:

Web design is not only about how the web site looks and feels but is also a lot about how it works and responds. In order to create a highly usable and effective website designer follow certain Principles that acts as to thumb rules and Standard points to keep in mind. The following are the various principles of an effective web design –

- Highly intuitive structure: The first law or principle of usability of website says that a web page mist have a highly intuitive structure and should be simple to understand so that user would not have to think which way to go. Makes the navigation intuitive and Simple.
- Visual Hierarchy: The next principle that contributes to Creating a successful and effective website is a visual hierarchy. Visual hierarchy is the order or sequence in which our eye moves, and perceives the things it sees, i.e. one topic/content/block to another. When designing a web page, a designer importance of the various topic and place them in such a way that the visitors first view what is most important and then moves onto the others in hierarchical manner
- Accessibility: Another highly important principle that must not be ignored when designing the web page is the accessibility of it. When the visitor enters the website; he/she must be able to access each bit of information in the easiest manner. This means that the text must be legible the color must not be harsh on the eyes and the background: must not overpower the content etc.
- Hicks Law: Hicks Law State that with every additional choice increases the time required to take. a decision. If you visit a restaurant and are provided with too many food items to pick from, you will take a longer time to take a decision. This means that we need to reduce the number of choices in order to provide a better user experience. Hicks Law can also be translated to more options means less sales.
- Fitts Law: Another law that acts as a major principle in web design is Fitts law. If you want your website visitor to take actions like order a product, read about a service etc. then you must make sure that they can reach the click here! more easily and quickly.
- Communication and Content: Everyone who visits your website is looking for some or other kind of information or content and thus it is very important for you to communicate with them clearly and in an engaging manner. Communication is not just about providing written information but also about offering images and other form of media such as videos and audio:
- White Space and Simple design: A simple design is an effective design. Complexity is just not something that a visitor wants to see on your web page and one of the most important: aspects of a Simple design are white space. White space helps to divide the

web page into several distinct parts or area which makes it simpler to process information. It is always better to have a whitespace. A solution to the problem of complex hierarchical structure. Following are some of the other things that can be consider as a part of Simple design:

- Grid-based design - To avoid a messy structure or appearance of the website you must use grid-based layout in which Content is divided into columns, boxes and different sections.
- F-pattern design - It is a fact that the human eye scan screen in an F pattern. Thus it is a good idea to design webpage or website in a way that complements the natural reading behavior of the visitors.

- Regular Testing: Test early and Test often or TETO is another web design principle that all designer and website owners must consider website constantly need upgrades and updates so as to maintain the visitor footsteps and customer interest.

Master Page:

A professional website will have a Standardized look across cell pages. For example one popular layout type places a navigation menu on the left side of the page, a copyright on the bottom and content in middle. It can be difficult to maintain a standard look with every web page. The Solution of above problem in Asp.Net is Master page.

Most of website pages are divided into three parts – header, page content and footer. Usually header and footer are the same across all pages on the website leaving only the page content to vary from page to page. A Master page allows the page header and footer to be created once and then reused. A Master page in Asp.Net work as template page. All aspx pages can refer master page with the help of master page we can define the look and feel of all pages in our website in single location. This makes it easier to maintain and update the overall design of a website, as changes only need to be made in one place.

If we have done changes in master page, then the changes will reflect in all the web pages that refer to master page. Asp.Net web page that uses master Page for common UI is called as Content page. Content pages merge with the master page at compile time to produce final output. The master pages define placeholders for the content, which are overridden for the content. The result is combination of master and content page. Every master page has one or more content pages in an application.

A master page has the extension .master that can comprise static text, HTML element and Server control. A master page is similar to Asp.Net Page but it contain @Master directive at the top and one or more ContentPlaceHolder server control. Master page inherits from MasterPage class. ContentPlaceHolder is an important control that is related with master page. This control is available only on master page.

Benefits of MasterPage:

There are several benefits of using a Master Page in ASP.NET:

- Consistent layout: A Master Page allows for a consistent layout across all pages in a website, which helps to improve the user experience.
- Code reuse: By defining a common layout in a Master Page, developers can reduce the amount of code that needs to be written and maintained, resulting in less time and effort required to create and update a website.
- Easy maintenance: By centralizing the layout on a Master Page, it is much easier to make updates or changes to the overall look and feel of a website.
- Improved organization: Master Pages can help to organize the codebase of a website, making it easier for developers to understand the structure and layout of the site.
- Enhance security: By using Master Pages, the common elements like the header, footer, and menus are separate and can be easily secured without affecting the other pages.
- Better SEO: With a consistent layout and navigational links, Master Pages can help improve the SEO of the website.
- Better performance: By using Master Pages, the common elements are cached on the browser which improves the performance of the website.

Terminologies used in MasterPage:

Below are several key terminologies used in Master Pages in ASP.NET:

- Master Page: A file that defines the layout and common elements of a website, such as a header, footer, and navigation menu.
- Content Page: A file that "inherits" the layout defined in a Master Page and contains unique content for a specific page.
- ContentPlaceHolder: A control that marks a region in a Master Page where content from a Content Page will be displayed.
- MasterType: A directive that allows a Content Page to access properties and methods of Master Page.
- Content: A control that defines the content that will be displayed in a ContentPlaceHolder.
- MasterPageFile: A property that specifies Master Page that a Content Page will inherit from.
- Page directive: A control that specifies Master Page that a Content Page will inherit from and other properties of the page.

- Event handling: Master Pages can raise events and the content pages can handle those events through event bubbling.
- Nested Master Pages: Master Pages can also inherit from other Master Pages.
- Data binding: Master Pages can access data from the content pages and bind it to its controls.

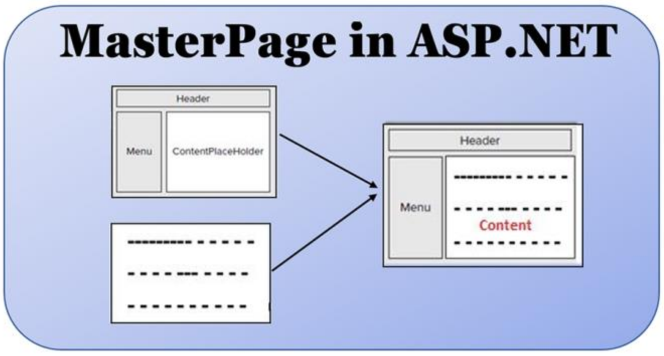
Content Page:

You define the content for the master Page placeholder controls by creating individual Content pages, which are Asp.Net.aspx pages that are bound to a specific master page. The binding is established in the content pages @page directive by including MasterPageFile attribute that point to the master page to be used. You can perform any task in a content Page that you can do in an Asp.Net page.

In Content page you create the Content by adding Content controls and mapping them to ContentPlaceHolder controls on the master page. Example The master page might have contentplaceholder called head and main. In the Content page (Aboutus.aspx) You can create two content controls one that is mapped to the contentplaceholder control head and other mapped to the ContentPlaceHolder control main. After Creating Content controls you add text and controls to them. In a Content page anything that is not inside the content controls (except- Script block) result in an error.

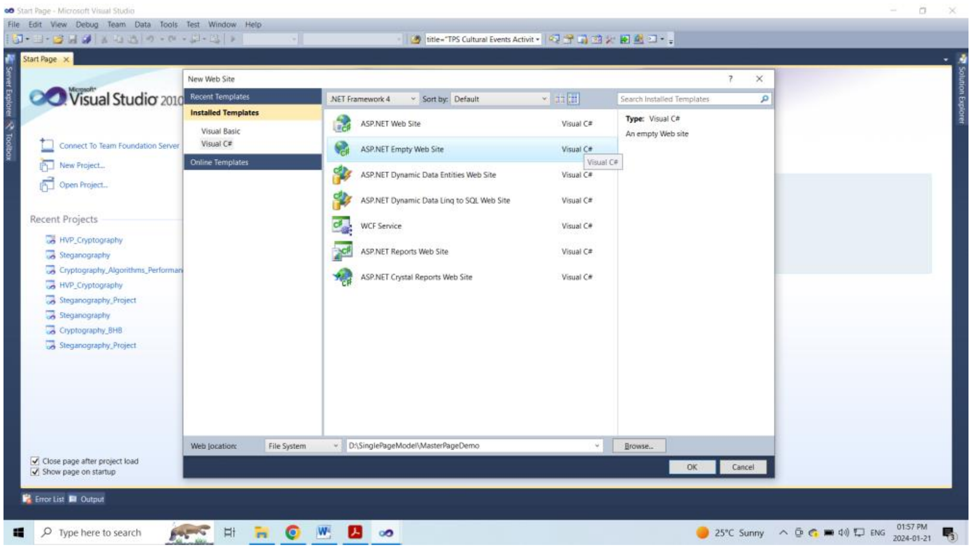
Important points about master pages

- The @ Master directive defines it as a master page.
- ContentPlaceHolder control is only available for master pages.
- ASP.NET page that uses master pages is called content pages.
- Master page - Content page relationship, let's look at the picture given below.



Steps to create the Master Page:

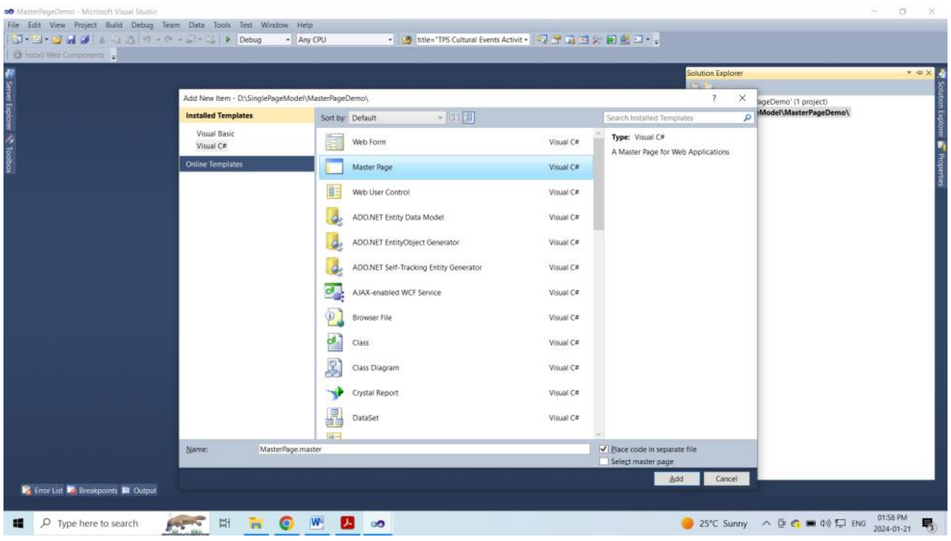
1. Create a New Web Project named MasterPageDemo.



Type: Visual C#
An empty Web site

Web location: File System D:\SinglePageModel\MasterPageDemo

2. Right-click the project in the solution explorer and select “Add Existing Item”.
3. Browse for the image named logo.jpg in the “Add Existing Item” dialog box and click “Add” button.
4. Right-click the project in the solution explorer and select Add New Item.
   Select “Master Page” from the “Visual Studio installed templates” section of the “Ad New Item” Dialog. Enter “MasterPage.master” in the name TextBox. Check the “Place code in separate file” checkbox and click “Add” button.



Type: Visual C#
A Master Page for Web Applications

Name: MasterPage.master

☑ Place code in separate file
☐ Select master page

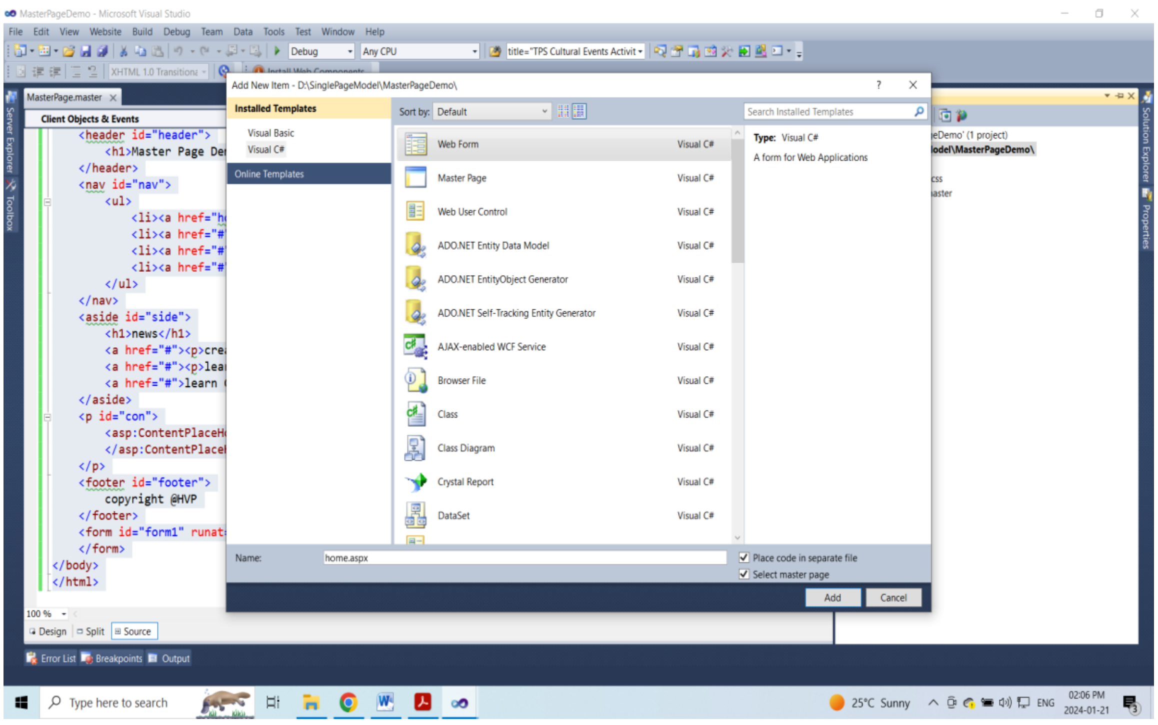
5. The source view of the MasterPage.master would look as below:

```
<%@ Master Language="C#" AutoEventWireup="true"
CodeFile="MasterPage.master.cs" Inherits="MasterPage" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
   <title>Master Page Demo</title>
   <link href="CSS/myCSSFile.css" rel="stylesheet" />
   <asp:ContentPlaceHolder ID="head" runat="server">
   </asp:ContentPlaceHolder>
</head>
<body>
   <header id="header">
      <h1>Master Page Demo</h1>
   </header>
   <nav id="nav">
      <ul>
         <li><a href="home.aspx">Home</a></li>
         <li><a href="#">About</a></li>
         <li><a href="#">Article</a></li>
         <li><a href="#">Contact</a></li>
      </ul>
   </nav>
   <aside id="side">
      <h1>news</h1>
      <a href="#"><p>creating HTML website</p></a>
      <a href="#"><p>learn CSS</p></a>
      <a href="#">learn C#</a>
   </aside>
   <p id="con">
      <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
      </asp:ContentPlaceHolder>
   </p>
   <footer id="footer">
      copyright @HVP
   </footer>
```

```
<form id="form1" runat="server">
</form>
 </body>
 </html>
```

Steps to create the Content Page:

1. Right-click the "MasterPageDemo" project in the solution explorer and select "Add New Item".
2. Select "Web Form" from the "Visual Studio installed templates" section of the "Add New
3. Item Dialog". Enter "home.aspx" in the name TextBox. Check the "Place code in separate file" checkbox. Check the "Select master page" checkbox and click "Add" button as shown below:



MasterPageDemo - Microsoft Visual Studio

Add New Item - D:\SinglePageModel\MasterPageDemo\

```
<header id="header">
    <h1>Master Page De
</header>
<nav id="nav">
    <ul>
        <li><a href="h
        <li><a href="#
        <li><a href="#
        <li><a href="#
    </ul>
</nav>
<aside id="side">
    <h1>news</h1>
    <a href="#"><p>cre
    <a href="#"><p>lea
    <a href="#">learn
</aside>
<p id="con">
    <asp:ContentPlaceH
    </asp:ContentPlace
</p>
<footer id="footer">
    copyright @HVP
</footer>
<form id="form1" runat
</form>
</body>
</html>
```

Web Form	Visual C#
Master Page	Visual C#
Web User Control	Visual C#
ADO.NET Entity Data Model	Visual C#
ADO.NET EntityObject Generator	Visual C#
ADO.NET Self-Tracking Entity Generator	Visual C#
AJAX-enabled WCF Service	Visual C#
Browser File	Visual C#
Class	Visual C#
Class Diagram	Visual C#
Crystal Report	Visual C#
DataSet	Visual C#

Type: Visual C#

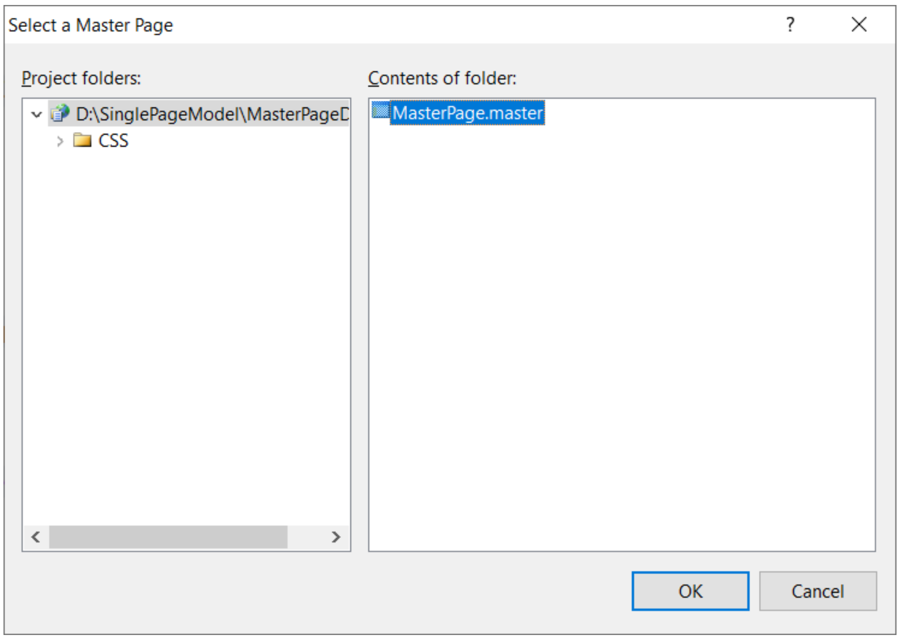
A form for Web Applications

Name: home.aspx

[✓] Place code in separate file

[✓] Select master page

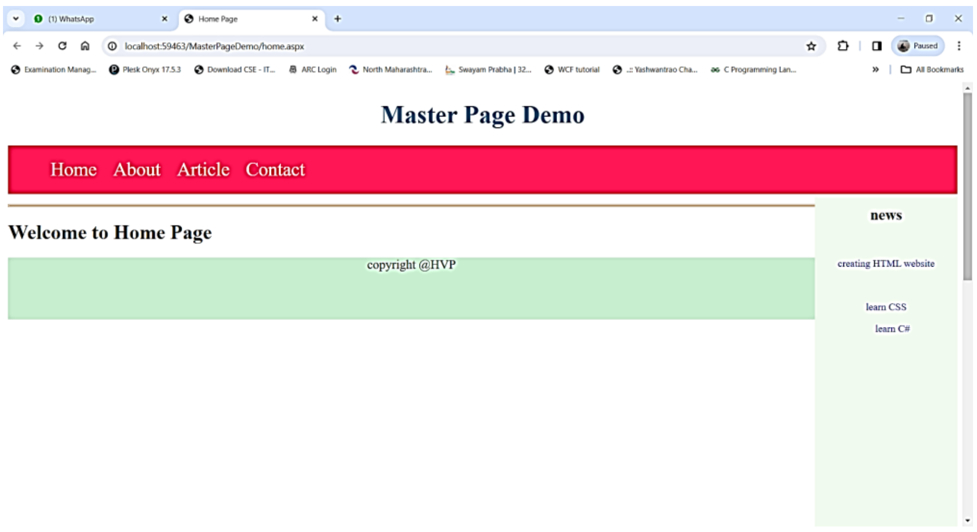
4. Select "MasterPage.master" from the "Contents of folder:" section in the "Select a Master Page" dialog and click the "OK" button as shown below:



5. The Design View of the "home.aspx" Design the "ContentPlaceHolder1" Content control as shown below by adding a Label to display "Welcome to Home Page".

```
   <%@ Page Title="Home Page" Language="C#"
MasterPageFile="~/MasterPage.master" AutoEventWireup="true"
CodeFile="home.aspx.cs" Inherits="home" %>


   <asp:Content ID="Content1" ContentPlaceHolderID="head"
Runat="Server">
   </asp:Content>
   <asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1"
Runat="Server">
     <h1>Welcome to Home Page</h1>
   </asp:Content>
```



Introduction to ADO.NET:

ADO.NET (Active Data Objects for .NET) is a set of classes in the .NET Framework designed to provide data access services to applications, including those built using ASP.NET. ADO.NET allows developers to interact with databases to retrieve, manipulate, and update data. It provides a consistent and efficient way to connect to various data sources, execute commands, and manage data in a disconnected or connected environment.

Basically it is container of all the standard classes which are responsible for database connectivity of .net application to the any kind of third party database software like Ms Sql Server, Ms Access, MySql, Oracle or any other database software. All classes belongs to ADO.NET are defined and arrange in "System.Data" namespace. ADO.NET was primarily developed to address two ways to work with data that we are getting from data sources. The two ways are as follows:

- The first is to do with the user's need to access data once and to iterate through a collection of data in a single instance i.e caching the data in runtime memory.
- The second way to work with data is in connected way which is we do not cache data. And we always go to database to retrieve it

Features of ADO.NET:

The following are the features of ADO.NET –

- Interoperability: We know that XML documents are text-based formats. So, one can edit and edit XML documents using standard text-editing tools. ADO.NET uses XML in all data exchanges and for internal representation of data.
- Maintainability: ADO.NET is built around the idea of separation of data logic and user interface. It means that we can create our application in independent layers.

- Programmability (Typed Programming): It is a programming style in which user words are used to construct statements or evaluate expressions.
- Performance: It uses disconnected data architecture which is easy to scale as it reduces the load on the database. Everything is handled on the client-side, so it improves performance.
- Scalability: It means meeting the needs of the growing number of clients, which degrading performance. As it uses disconnected data access, applications do not retain database lock connections for a longer time. Thus, it accommodates scalability by encouraging programmers to conserve limited resources and allow users to access data simultaneously.

The architecture of ADO.NET:

ADO.NET has two main components that are used for accessing and manipulating data are the .NET Framework data provider and the DataSet.

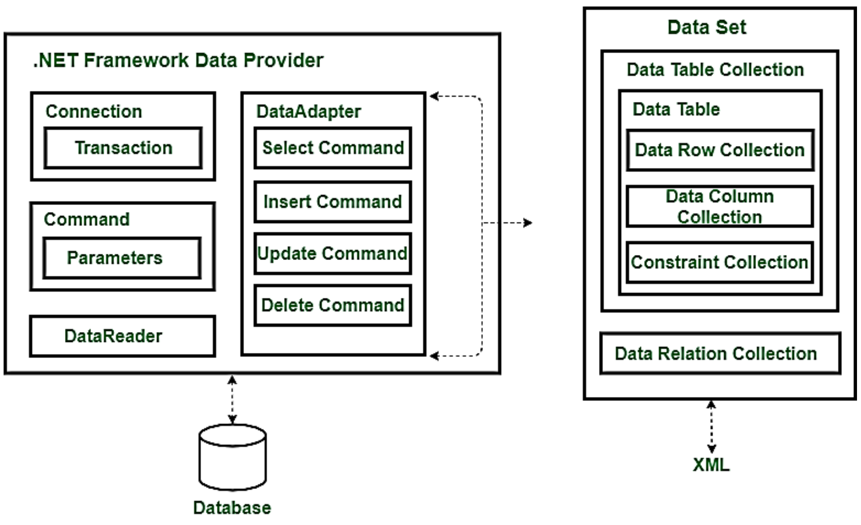


Figure : Architecture of ADO.NET

ADO.NET contain following important components.

- Data Provider
- Connection
- Command
- DataReader
- DataAdapter
- Dataset

Data Providers:

Data Providers are a collection of classes specifically designed to interact with various types of data sources. These classes serve as the intermediaries between the application and the underlying databases, enabling seamless execution of data management operations. Whether it is retrieving data, performing updates, or executing queries, Data Providers handle the intricate tasks associated with specific databases, ensuring smooth and efficient data access. These are the components that are designed for data manipulation and fast access to data. It provides various objects such as Connection, Command, DataReader and DataAdapter that are used to perform database operations.

Within the .NET Framework, there are primarily three Data Providers available for ADO.NET: the Microsoft SQL Server Data Provider, the OLEDB Data Provider, and the ODBC Data Provider. The Microsoft SQL Server Data Provider is specifically designed for interacting with Microsoft SQL Server databases. The OLEDB Data Provider, on the other hand, facilitates connectivity with databases that support the OLEDB (Object Linking and Embedding, Database) technology such as Oracle or Access, etc. Lastly, the ODBC Data Provider is designed to work with databases that conform to the ODBC (Open Database Connectivity) standard.

Connection:

The Connection Object within ADO.NET serves as the means to establish a physical connection to the data source. In order to establish this connection successfully, the Connection Object requires specific information that allows it to identify and authenticate with the data source. This crucial information is typically provided through a connection string.

Command:

The Command Object is a vital component in ADO.NET that facilitates the execution of SQL statements or stored procedures at the data source. It serves as the bridge between the application and the data source, allowing developers to interact with the data by issuing SQL queries or executing stored procedures.

DataReader:

The DataReader Object plays a crucial role in ADO.NET by providing a stream-based, forward-only mechanism for retrieving query results from the data source. Unlike other components, the DataReader is specifically designed for read-only operations and does not support data modification.

DataAdapter:

The DataAdapter Object plays a critical role in ADO.NET by facilitating the population of a Dataset Object with results retrieved from a data source. It acts as a bridge between disconnected Dataset objects and the physical data source, enabling seamless interaction and synchronization between the two.

DataSet:

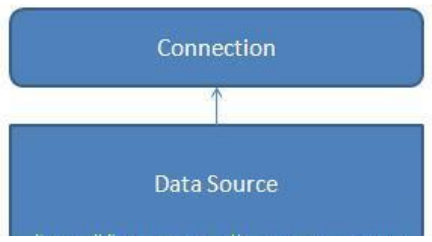
A DataSet is an in-memory cache of data retrieved from a data source. It can contain multiple DataTable objects, each representing a table of data. The DataSet Object in ADO.NET serves as a disconnected representation of result sets retrieved from a data source. It provides a flexible and independent in-memory storage for data, enabling efficient manipulation and management within the application. Within the DataSet, data is organized into rows and columns, similar to a traditional database table.

ASP.NET Connection Class:

The Connection Object is an integral part of the ADO.NET Data Provider, serving as a means to establish and maintain a connection with the data source. It represents a unique session between the application and the data source, enabling communication and data exchange.

Connection Object

To establish a connection, the Connection Object requires the necessary information to identify and authenticate with the data source. This information is typically provided through a connection string, which contains details such as the server name, database name, credentials, and other parameters required to establish the connection.



When the Connection Object connects to the specified data source, it initiates a connection between the application and the data source. This connection allows SQL commands and queries to be executed against the data source using the Command Object. The Command Object is responsible for issuing SQL statements, stored procedures, or other database operations to retrieve or manipulate data. The specific type of Connection Object used depends on the data source system being utilized. For example, when working with Microsoft SQL Server, the SqlConnection object is used to establish a connection. Other data providers, such as the OleDbConnection or OdbcConnection, are employed when working with different database systems.

ASP.NET Sql Server Connection:

The SqlConnection object plays a vital role in facilitating the physical communication between an ASP.NET application and a SQL Server database. It serves as a crucial component within the

Data Provider for SQL Server Database, enabling seamless data exchange between the application and the database.

SqlConnection class

When an instance of the SqlConnection class is created in an ASP.NET application, it establishes a reliable and secure connection to the SQL Server database. This connection acts as a conduit for executing SQL commands and queries, allowing the application to retrieve or manipulate data stored in the database. The execution of SQL commands is made possible with the assistance of the Command Object, which works in conjunction with the SqlConnection to carry out database operations.

SqlConnection connection = new SqlConnection(connectionString);

Close() method

Once the desired database activities, such as data retrieval or modification, are completed, it is essential to properly close the SqlConnection. The Close() method provided by the SqlConnection class serves this purpose, allowing for the graceful termination of the database connection. When the Close() method is invoked, any pending transactions are rolled back, ensuring data integrity, and the connection is released from the SQL Server database. By closing the connection, valuable data source resources are freed, promoting efficient resource management within the application.

Example:

web.config

```
 <?xml version="1.0"?>
<configuration>
  <connectionStrings>
<add name="SQLDbConnection" connectionString="Server=servername; Database=studDB; User Id=username; password=password" providerName="System.Data.SqlClient" />
  </connectionStrings>
</configuration>
```

Default.aspx

```
 <html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>Database Connection Demo</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
      <asp:Button ID="Button1" runat="server" Text="Button" onclick="Button1_Click" />
    </div>
    <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
    </form>
    </body>
    </html>
```

Default.aspx.cs

```
using System;
using System.Data ;
using System.Data.SqlClient ;
using System.Configuration;
public partial class _Default : System.Web.UI.Page
{
    protected void Button1_Click(object sender, EventArgs e)
    {
            string connectionString = ConfigurationManager.ConnectionStrings
            ["SQLDbConnection"].ToString();
            SqlConnection connection = new SqlConnection(connectionString);
            connection.Open();
            Label1.Text = "Connected to Database Server !!";
            connection.Close();
    }
}
```

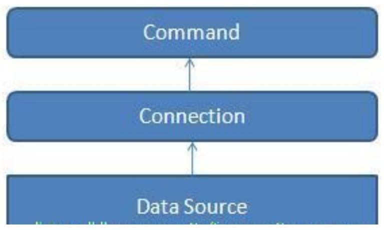
ASP.NET Command Class:

The Command Object in ADO.NET is a fundamental component that executes SQL statements and stored procedures against the specified data source. To execute these commands, the Command Object relies on an instance of an ASP.NET Connection Object.

Command Object

The Command Object serves as a bridge between the application and the data source, enabling the execution of SQL statements and stored procedures. It provides various properties and methods that allow developers to configure and control the execution process.

SqlCommand cmd = new SqlCommand(sql, connection);



Command Object Property:

One important property of the Command Object is CommandText. This property holds a String value representing the SQL command or stored procedure that will be executed against the data source. For example, if the CommandType property is set to Text (indicating a SQL statement), the CommandText property should contain the actual SQL query. On the other hand, if the CommandType is set to StoredProcedure, the CommandText property should contain the name of the stored procedure to be executed.

Command Object Methods:

- **ExecuteNonQuery() method**

  The ExecuteNonQuery() method in ADO.NET is used to execute a Transact-SQL statement against a connection and returns the number of rows affected by the operation. This method is capable of performing both Data Definition tasks and Data Manipulation tasks. Data Definition tasks involve actions such as creating stored procedures, views, or other database objects. These tasks can be accomplished using the ExecuteNonQuery() method. When executing a Data Definition statement, the method does not return any rows, but it can still affect the structure or schema of the database.

  Example:

  Default.aspx

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
<title>Insert Record into Table</title>
</head>
<body>
<form id="form1" runat="server">
<div>
            <h1> Insert Record into Table</h1>
            <asp:Button ID="Button1" runat="server" Text="Button"
            onclick="Button1_Click" />
</div>
<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
</form>
</body>
</html>
```

Default.aspx.cs

```
using System;
using System.Data ;
using System.Data.SqlClient ;
using System.Configuration;
public partial class _Default : System.Web.UI.Page
{
        protected void Button1_Click(object sender, EventArgs e)
        {
                string connectionString =
ConfigurationManager.ConnectionStrings ["SQLDbConnection"].ToString();
        SqlConnection connection = new SqlConnection(connectionString);
        string sql = "insert into studTBL values('Purva Patil',101, 'SYBCA')";
                try
                {
                        connection.Open();
                        SqlCommand cmd = new SqlCommand(sql, connection);
                        cmd.ExecuteNonQuery();
                        connection.Close();
                        Label1.Text = "Successfully Inserted!!";
                }
                catch (Exception ex)
                {
                        Label1.Text = "Error inserting data" + ex.ToString();
                }
        }
}
```

- **ExecuteReader() method**

  The ExecuteReader() method in the SqlCommand Object is used to send SQL statements to the Connection Object and retrieve query results from the data source. When this method is executed, it creates an instance of the SqlDataReader Object, specific to the SqlClient namespace.

  Example

  Default.aspx

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>Retrieve Record From Table </title>
</head>
```

```
<body>
      <form id="form1" runat="server">
      <div>
               <h1> Retrieve Record From Table</h1>
               <asp:Button ID="Button1" runat="server" Text="Button"
               onclick="Button1_Click" />
               <br />
               <asp:ListBox ID="ListBox1" runat="server"></asp:ListBox>
               <br />
               <asp:Label ID="Label1" runat="server"
Text="Label"></asp:Label>
      </div>
      </form>
</body>
</html>
```

Default.aspx.cs

```
 using System;
 using System.Data ;
 using System.Data.SqlClient ;
 using System.Configuration;
 public partial class _Default : System.Web.UI.Page
 {
      protected void Button1_Click(object sender, EventArgs e)
       {
      string connectionString = ConfigurationManager.ConnectionStrings
      ["SQLDbConnection"].ToString();
       SqlConnection connection = new SqlConnection(connectionString);
      string sql = "select *  from studTBL";
      try
      {
      connection.Open();
 SqlCommand cmd=new SqlCommand(sql, connection);
      SqlDataReader reader = cmd.ExecuteReader();
      while (reader.Read())
      {
       ListBox1.Items.Add(reader.GetValue(0) + " " + reader.GetValue(1) + " "
      + reader.GetValue(2));
```

```
                    }
                    connection.Close();
                }
                catch (Exception ex)
                {
                Label1.Text = "Error in ExecuteReader " + ex.ToString();
                }
            }
        }
```

- **ExecuteScalar() method**

  The ExecuteScalar() method in ADO.NET is specifically designed to retrieve a single value from a database. It is commonly used when you expect a result set to contain a single row with a single column. This method is particularly useful when working with aggregate functions such as Count(*) or Sum(). Instead of retrieving an entire result set, which might be unnecessary if you only need a single value, you can use ExecuteScalar() to retrieve that specific value efficiently.

  Example:

  Default.aspx

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
   <title> Retrieve Single Record From Table </title>
</head>
<body>
        <form id="form1" runat="server">
        <div>
              <h1> Retrieve Single Record From Table</h1>
              <asp:Button ID="Button1" runat="server" Text="Button"
              onclick="Button1_Click" />
        </div>
        <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
</form>
</body>
</html>
```

Default.aspx.cs

```
using System;
using System.Data ;
using System.Data.SqlClient ;
using System.Configuration;
public partial class _Default : System.Web.UI.Page
{
        protected void Button1_Click(object sender, EventArgs e)
        {
                string connectionString = ConfigurationManager.
               ConnectionStrings ["SQLDbConnection"].ToString();
               SqlConnection connection = new SqlConnection
               (connectionString);
              string sql = "select count(*) from studTBL";
              try
             {
                    connection.Open();
                    SqlCommand cmd = new SqlCommand(sql, connection);
                    int result = Convert.ToInt32(cmd.ExecuteScalar());
                    connection.Close();
                    Label1.Text = "Number of rows in studTBL table - " + result;
                    }
                    catch (Exception ex)
                    {
                        Label1.Text = "Error in ExecuteScalar " + ex.ToString();
                    }
        }
    }
```

ASP.NET DataReader Class:

The DataReader Class in ADO.NET provides a stream-based, forward-only, and read-only retrieval of query results from data sources. It is specifically designed for efficiently accessing and processing large result sets without the need to load the entire set into memory. It is important to note that the DataReader Object is read-only and does not support data modification or updating operations.

DataReader Object

To create a DataReader Object, you need to first instantiate a Command Object, such as a SqlCommand Object, and then call the ExecuteReader() method of the Command Object. This method executes the SQL command or query and returns a DataReader Object containing the retrieved rows from the data source.

SqlDataReader reader = cmd.ExecuteReader();

Once the DataReader Object is created, it should be open and positioned prior to the first record in order to start reading from it. The Read() method of the DataReader Object is used to sequentially read the rows from the DataReader. Each call to the Read() method moves the DataReader to the next valid row, if one exists, allowing you to access the columns and values of that row.

Example

Default.aspx

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
        <title>Retrieve Record From Table </title>
</head>
<body>
        <form id="form1" runat="server">
        <div>
                <h1> Retrieve Record From Table</h1>
                <asp:Button ID="Button1" runat="server" Text="Button"
                onclick="Button1_Click" />
                <br />
                <asp:ListBox ID="ListBox1"
runat="server"></asp:ListBox>
                <br />
                <asp:Label ID="Label1" runat="server"
Text="Label"></asp:Label>
        </div>
        </form>
</body>
</html>
```

Default.aspx.cs

```
using System;
using System.Data ;
using System.Data.SqlClient ;
using System.Configuration;
public partial class _Default : System.Web.UI.Page
{
            protected void Button1_Click(object sender, EventArgs e)
            {
              string connectionString = ConfigurationManager.ConnectionStrings
              ["SQLDbConnection"].ToString();
              SqlConnection connection = new SqlConnection(connectionString);
              string sql = "select *  from studTBL";
               try
                   {
                           connection.Open();
                           SqlCommand cmd = new SqlCommand(sql, connection);
                           SqlDataReader reader = cmd.ExecuteReader();
                           while (reader.Read())
                           {
                              ListBox1.Items.Add(reader.GetValue(0) + " " +
                               reader.GetValue(1) + " " + reader.GetValue(2));
                           }
                           connection.Close();
                   }
                   catch (Exception ex)
                   {
                   Label1.Text="Error in ExecuteReader " + ex.ToString();
            }
    }
}
```

ASP.NET DataAdapter Class:

The DataAdapter in ADO.NET acts as a bridge between a DataSet object and a SQL Server database, facilitating the retrieval and saving of data. It provides a seamless integration between the DataSet and the database by utilizing the SqlDataAdapter object.

DataAdapter with DataSet object

By combining the DataAdapter with a DataSet object, developers can perform both data access and data manipulation operations. The **Fill method** of the DataAdapter is used to populate the DataSet with data from the database, ensuring that the data in the DataSet matches the data in the data source. On the other hand, the Update method of the DataAdapter is used to update the data in the data source to match the data in the DataSet.

SqlDataAdapter adapter = new SqlDataAdapter(sql,connection);
adapter.Fill(ds);

When working with the DataAdapter, the user can perform SQL operations such as Select, Insert, Update, and Delete on the data contained within the DataSet. However, these operations do not directly affect the database until the Update method of the DataAdapter is invoked. This ensures that any changes made to the data in the DataSet are reflected in the database.

Example: program shows a select operation using SqlDataAdapter.

Default.aspx

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Button ID="Button1" runat="server" Text="Button"
        onclick="Button1_Click" />
        <br />
        <asp:ListBox ID="ListBox1" runat="server"></asp:ListBox>
        <br />
        <asp:Label ID="Label1" runat="server"
Text="Label"></asp:Label>
    </div>
    </form>
</body>
</html>
```

Default.aspx.cs

```
using System;
using System.Data ;
using System.Data.SqlClient ;
using System.Configuration;
public partial class _Default : System.Web.UI.Page
{
        protected void Button1_Click(object sender, EventArgs e)
        {
                string connectionString = ConfigurationManager.ConnectionStrings
                ["SQLDbConnection"].ToString();
                SqlConnection connection = new SqlConnection(connectionString);
                DataSet ds = new DataSet ();
                string sql = "select studName from studTBL";
                try
                        {
                        connection.Open();
                        SqlDataAdapter adapter = new SqlDataAdapter(sql,connection );
                        adapter.Fill(ds);
                        for (int i = 0;i < ds.Tables[0].Rows.Count -1;i++)
                        {
                                ListBox1.Items.Add(ds.Tables[0].Rows
                                [i].ItemArray[0].ToString ());
                        }
                        connection.Close();
                        }
                        catch (Exception ex)
                {
                         Label1.Text = "Error in execution " + ex.ToString();
                }
        }
    }
```

ASP.NET DataSet Class:

The DataSet in ADO.NET provides a disconnected representation of result sets from a data source. It serves as an in-memory container that holds data retrieved from a database, and it is completely independent of the original data source. This disconnected nature of the DataSet allows it to work with the data without being tied to the data source, providing greater flexibility. A DataSet is composed of one or more DataTable objects, each representing a table of data retrieved from the database. These DataTables can be related to each other using Data Relations within the DataSet, allowing for complex data structures and hierarchies to be represented.

The DataAdapter object is used to populate the DataTables within the DataSet. The Fill() method of the DataAdapter is used to populate the data in the DataSet from the data source. This method retrieves data from the data source and populates the corresponding DataTables, enabling efficient data retrieval and manipulation. The DataSet can be filled from a data source using the DataAdapter or dynamically by programmatically adding and manipulating DataTables and their data within the DataSet.

Example:

Default.aspx

```
    <%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="Default.aspx.cs" Inherits=" datasetexample._Default" %>
    <!DOCTYPE html>
    <html xmlns="http://www.w3.org/1999/xhtml">
    <head runat="server">
      <title>DataSet Example</title>
    </head>
    <body>
      <form id="form1" runat="server">
        <div>
          <asp:GridView ID="GridView1" runat="server"
AutoGenerateColumns="True"></asp:GridView>
        </div>
      </form>
    </body>
    </html>
```

Default.aspx.cs

```
using System;
using System.Data;
using System.Data.SqlClient;
namespace datasetexample
{
  public partial class _Default : System.Web.UI.Page
  {
    protected void Page_Load(object sender, EventArgs e)
    {
      if (!IsPostBack)
      {
```

```
                    // Connection string (update with your database details)
                      string connectionString = "Data Source=YourServer;Initial
                      Catalog=studDB;User ID=YourUsername;Password=YourPassword";
                    // SQL query
                    string query = "SELECT * FROM studTBL";
                    // Create SqlConnection and SqlDataAdapter
                    using (SqlConnection connection = new SqlConnection(connectionString))
                    {
                     using (SqlDataAdapter adapter = new SqlDataAdapter(query, connection))
                        {
                            // Create a DataSet
                            DataSet dataSet = new DataSet();
                            // Fill the DataSet with data from the database
                            adapter.Fill(dataSet, " studTBL");
                            // Bind the DataSet to the GridView
                            GridView1.DataSource = dataSet.Tables["studTBL"];
                            GridView1.DataBind();
                        }
                    }
                }
            }
        }
    }
```

:: QUESTIONS ::

1. What are Master Pages? How content page are created using Master Page? Explain?
2. What is Web Design? Explain the web page design principles?
3. Write short note on following ADO.NET Objects
   - Connection object
   - Command object
   - DataReader Object,
   - DataAdapter Object
4. Define ADO.NET. How Database connection are made using ADO.NET? Explain?
5. What is ADO.NET? Explain the architecture of ADO.NET?
6. Explain the steps of creating master page in web development?
7. Explain ADO.NET objects in details?

Notes...

As Per New Syllabus of Kavayitri Bahinabai Chaudhari
North Maharashtra University, Jalgaon

Bachelor of Computer Application (BCA) (w.e.f. 2023-24)

SEMESTER I

BCA 101	Fundamentals of Accounting
BCA 102	Fundamentals of Computer
BCA 103	Programming in C – I
BCA 104	Web Design – I

SEMESTER II

BCA 201	Professional Communication Skill
BCA 202	Database Management System
BCA 203	Programming in C – II
BCA 204	Web Design - II

SEMESTER III

BCA 301	Fundamental Mathematics and Statistics
BCA 302	Operating System
BCA 303	Programming in C++
BCA 304A	Web Development Technology – I
BCA 304C	Python Programming

SEMESTER IV

BCA 401	Software Engineering
BCA 402	Data Structure
BCA 403	Java Programming
BCA 404A	Web Development Technology - II
BCA 404C	Artificial Intelligence

SEMESTER V

BCA 501	Employability Skill
BCA 502	E-Commerce and M-Commerce
BCA 503	Cloud Computing Application
BCA 504A	Web Development Technology – III
BCA 504C	Machine Learning

SEMESTER VI

BCA 601	Entrepreneurship Development
BCA 602	Cyber Security
BCA 603	Android Application Development
BCA 604A	Web Development Technology – IV
BCA 604C	Data Mining

Prashant

BCA ₹ 120

ISBN 978-81-19120-68-0

www.prashantpublications.com
prashantpublication.jal@gmail.com