MCA 401B: Dot Net Technologies

UNITI

The .NET Framework **:** Introduction, Common Language Runtime, Common Type System, Common Language Specification, The Base Class Library, The .NET class library Intermediate language, Justin Time compilation, garbage collection, Application installation & Assemblies, Web Services, Unified classes.**C# Basics** -Introduction, DataTypes, Identifiers, variables & constants, C# statements, Object Oriented Concept, Object and Classes, Arrays and Strings, System Collections, C# - Regular Expressions.

UNIT II

C# Using Libraries -Namespace-System, Input Output, Multi-Threading, Networking and Sockets, Data Handling, Windows Forms, C# in Web application, Error Handling.

UNIT III

Advanced Features Using C#**:** Delegates and Events, Indexes Attributes, versioning, Web Services, Windows services, messaging, Reflection, COM and C#, localization. Distributed Application in C#, XML and C#, Unsafe Mode, Case Study (Messenger Application).

UNIT IV

Advanced Programming Constructs: Database Connectivity with ADO.NET Creating Distributed Web Applications, XML and ADO.NET, Graphics, Printing, data Reports, crystal Reports,C# libraries for Image Processing, .Net applications to Azure platform

UNIT V

Web Forms Processing, Introduction to Server Controls, HTML Controls, Validation Controls, User control, Data Binding Controls, Master-detail forms, Configuration, Personalization, Session State, Database Connectivity with ADO.NET.

LECTURE NOTES

UNIT-1

NET Framework

The **.NET Framework** is a software development platform that was introduced by Microsoft in the late 1990 under the NGWS. On 13 February 2002, Microsoft launched the first version of the .NET Framework, referred to as the **.NET Framework 1.0**.

In this section, we will understand the **.NET Framework, characteristics**, **components,** and its **versions**.

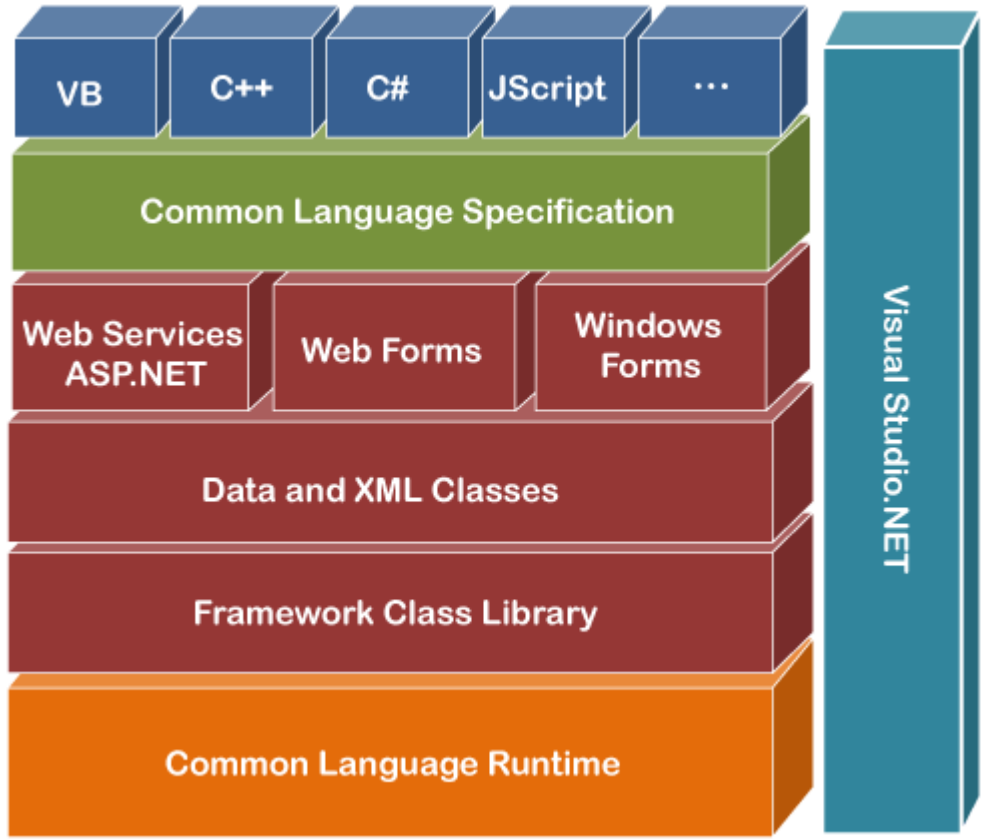
What is .NET Framework

It is a virtual machine that provide a common platform to run an application that was built using the different language such as C#, VB.NET, Visual Basic, etc. It is also used to create a form based, console-based, mobile and web-based application or services that are available in Microsoft environment. Furthermore, the .NET framework is a

pure object oriented, that similar to the Java language. But it is not a platform independent as the Java. So, its application runs only to the windows platform.

The main objective of this framework is to develop an application that can run on the windows platform. The current version of the .Net framework is 4.8.

> ***Note: The .NET Framework is not only a language, but it is also a software and language neutral platform.***



Components of .NET Framework

There are following components of .NET Framework:

1. CLR (Common Language Runtime)
2. CTS (Common Type System)
3. BCL (Base Class Library)
4. CLS (Common Language Specification)
5. FCL (Framework Class Library)
6. .NET Assemblies
7. XML Web Services
8. Window Services

CLR (common language runtime)

It is an important part of a .NET framework that works like a virtual component of the .NET Framework to executes the different languages program like c#, Visual Basic, etc. A CLR also helps to convert a source code into the byte code, and this byte code is known as CIL (Common Intermediate Language) or MSIL (Microsoft Intermediate Language). After converting into a byte code, a CLR uses a JIT compiler at run time that helps to convert a CIL or MSIL code into the machine or native code.

CTS (Common Type System)

It specifies a standard that represent what type of data and value can be defined and managed in computer memory at runtime. A CTS ensures that programming data defined in various languages should beinteract with each other to share information. For example, in C# we define data type as int, while in VB.NET we define integer as a data type.

BCL (Base Class Library)

The base class library has a rich collection of libraries features and functions that help to implement many programming languages in the .NET Framework, such as C #, F #, Visual C ++, and more. Furthermore, BCL divides into two parts:

1. **User defined class library**
    - **Assemblies -** It is the collection of small parts of deployment an application's part. It contains either the DLL (Dynamic Link Library) or exe (Executable) file.
        1. In LL, it uses code reusability, whereas in exe it contains only output file/ or application.
        2. DLL file can't be open, whereas exe file can be open.
        3. DLL file can't be run individually, whereas in exe, it can run individually.
        4. In DLL file, there is no main method, whereas exe file has main method.
2. **Predefined class library**
    - **Namespace -** It is the collection of predefined class and method that present in .Net. In other languages such as, C we used header files, in java we used package similarly we used "using system" in .NET, where using is a keyword and system is a namespace.

CLS (Common language Specification)

It is a subset of common type system (CTS) that defines a set of rules and regulations which should be followed by every language that comes under the .net framework. In other words, a CLS language should be cross-language integration or interoperability. For example, in C# and VB.NET language, the C# language terminate each statement with semicolon, whereas in VB.NET it is not end with semicolon, and when these statements execute in .NET Framework, it provides a common platform to interact and share information with each other.

Microsoft .NET Assemblies

A .NET assembly is the main building block of the .NET Framework. It is a small unit of code that contains a logical compiled code in the Common Language infrastructure (CLI), which is used for deployment, security and versioning. It defines in two parts (process) DLL and library (exe) assemblies. When the .NET program is compiled, it generates a metadata with Microsoft Intermediate Language, which is stored in a file called Assembly.

FCL (Framework Class Library)

It provides the various system functionality in the .NET Framework, that includes classes, interfaces and data types, etc. to create multiple functions and different types of application such as desktop, web, mobile application, etc. In other words, it can be defined as, it provides a base on which various applications, controls and components are built in .NET Framework.

Intermediate Language (IL)

Intermediate language (IL) is an object-oriented programming language designed to be used by compilers for the .NET Framework before static or dynamic compilation to machine code. The IL is used by the .NET Framework to generate machine-independent code as the output of compilation of the source code written in any .NET programming language.

IL is a stack-based assembly language that gets converted to bytecode during execution of a virtual machine. It is defined by the common language infrastructure (CLI) specification. As IL is used for automatic generation of compiled code, there is no need to learn its syntax.

This term is also known as Microsoft intermediate language (MSIL) or common intermediate language (CIL).

Techopedia Explains Intermediate Language (IL)

With the help of a suitable just-in-time (JIT) compiler, IL code can be executed on any computer architecture supported by the JIT compiler. Unlike interpreters, JIT compilation provides better performance, preserves memory, and saves time during application initialization. IL enables the platform- and CPU-independence feature of the .NET framework, by allowing compiled source code to be executed in any environment supporting the CLI specification.

Verification of code safety, for IL code, provides better security and reliability than natively-compiled executable files. The metadata, describing the MSIL code in the portable executable, eliminates the need for type libraries and intermediate definition language files as was used in the Component Object Model (COM) technology. Combined with metadata and a common type system, IL forms the means to integrate modules written in different languages into one single application, thus enabling language independence.

Although IL is similar to Java bytecode in its usage by compilers, it differs from the latter in that it is designed for platform independence and language independence. It also differs in that it is compiled and not interpreted.

Two types of instruction sets are included with IL; base instructions, similar to native CPU instructions, and Object model instructions used by the high-level language. IL includes all instructions necessary for loading, storing, initializing, and calling methods on objects. It also includes arithmetic and logical operations, control flow, direct memory access, exception handling and other operations. Unlike the common object file format used for executable content in the traditional Microsoft portable executable, the portable executable generated, after the compilation of managed code, contains both IL instructions and metadata.

The two tools associated with IL code are the MSIL Assembler (Ilasm.exe) and the MSIL Disassemble (Ildasm.exe). The former generates a portable executable file from IL code and permits viewing the IL code in human-readable format, while the latter converts a portable executable file back to a text file, for viewing and modification. Both are automatically installed as part of Visual Studio.

Just-in-time compiler (JIT)

A just-in-time (JIT) compiler is a program that turns bytecode into instructions that can be sent directly to As opposed to other compiler types,

the main defining characteristic of a JIT compiler is that a JIT compiler runs *after* a program starts and compiles code. A common way to say this is that a JIT compiler compiles code on the fly, or in other words, just in time. The compiled code, or bytecode sequences, are turned into a faster, more readable machine language, or native code. Translating source code into a language that a host CPU natively understands typically means a faster, more easily readable instruction set.

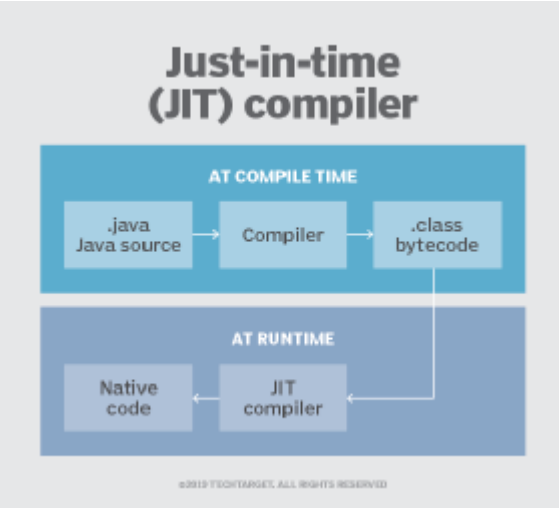
JIT compilers contrast different compiler types such as a traditional compiler, which will compile all code to a machine language *before* a program starts to run. Newer programs will make use of JIT compilers, which generate code while the program is running. Two common uses of JIT compilers include Java Virtual Machine (JVM) which is used in Java, as well as CLR (Common Language Runtime) which is used in C#.

For example, in the Java programming language and environment, a just-in-time (JIT) compiler turns Java bytecode -- a program that contains instructions that must be interpreted -- into instructions that can be sent directly to the processor. JIT compilers are utilized commonly in a Java Runtime Environment, as they are used to optimize performance in java-based applications.

just-in-time compilation work

Bytecode is an important feature in applications that help in cross-platform execution. Additionally, bytecode must be compiled and translated to a language a CPU can properly understand. However, how that bytecode is translated into a native language may have a large impact on the speed and performance of an application. To improve performance, JIT compilers will, at runtime, compile suitable bytecode sequences into machine code. The code is typically sent to a processor, where the code instructions are then carried out. When the same block of bytecode is needed again, the previously created object code will be used. Code that looks like it can be re-optimized is called "hot." Code can be monitored, and hot code paths can be created to optimize code, as opposed to having the same sequence

of code be interpreted multiple times -- which may occur in other compiler types. With less chance of code being interpreted multiple times, there is less overhead, meaning faster execution speeds. This is why most implementations of JVM use JIT compilers.



A visual of how a just-in-time (JIT) compiler works.

A JIT compiler can also make relatively simple optimizations when compiling bytecode into a native machine language. As an example, a JIT compiler can get rid of common sub-expressions, reduce memory access in register allocations, and perform data-analysis and register operations by translating from stack operations.

However, because of the time it takes to load and compile bytecode, there is a startup delay in the initial execution of an application. To help anticipate startup times, a good rule of thumb to follow is that the more JIT compilers are used to optimize a system, the longer the initial startup delay will be. There are a few ways to decrease initial startup delays, such as separating startup modes. For example, including both a client and server mode could allow minimal compilation and optimization when in one mode versus the other, meaning that the chosen mode will have a faster startup time. Another way is to combine JIT compilers with either AOT (ahead-of-time) compilers or interpreters.

Phases of just-in-time compilation

As mentioned, a JIT compiler will compile suitable bytecode sequences into machine code, and then the code is sent to a processor where the code instructions are carried out.

JIT compilation can also be separated by different levels of optimization: cold, warm, hot, very hot and scorching. Each optimization level is set to provide a certain level of performance. The initial, or default, level of optimization is called warm, while code that looks like it can be further re-optimized is called hot. This level increases upwards until scorching, with each level being of higher importance pertaining to performance that can be re-optimized. Through sampling, a JIT compiler can determine which methods appear more often at the top of a stack. The optimization level can also be downgraded to cold, to further improve startup time.

Just-in-time compilers vs. interpreters vs. ahead-of-time compilers

In general, interpreters and compilers can both be used to translate machine language. An interpreter will commonly perform tasks such as Parsing, type checking and lexing, and does the same kind of work as a compiler. Interpreters have a fast startup time, don't have to go through a compilation step and will translate code one line at a time, on the fly. However, an interpreter may be slower than a compiler in a case where an application runs the same code multiple times -- since the same translation must happen as many times as the code is repeated.

A typical compiler does not translate on the fly, as it translates compiled code before the application begins running. Because of this, its startup time is a bit longer, but any code loops that appear will run faster -- since the code translation does not need to be repeated multiple times. Compilers have more time to look at the code and make any optimizations as well.

JIT compilers combine both principals to get the best of both worlds.

Ahead-of-time, or AOT compilers, compile code into a native machine language similar to a normal compiler; however, AOT will transform bytecode of a virtual machine into machine code.

Advantages of just-in-time compilation

Advantages of JIT compilation include:

- JIT compilers need less memory usage.
- JIT compilers run after a program starts.
- Code optimization can be done while the code is running.
- Any page faults can be reduced.
- Code that is used together will be localized on the same page.
- Can utilize different levels of optimization.

Disadvantages of just-in-time compilation

Disadvantages of JIT compilation include:

- Startup time can take a noticeable amount of time.
- Heavy usage of cache memory.
- Increases the level of complexity in a Java program.
- **just-in-time compilers do in Java**

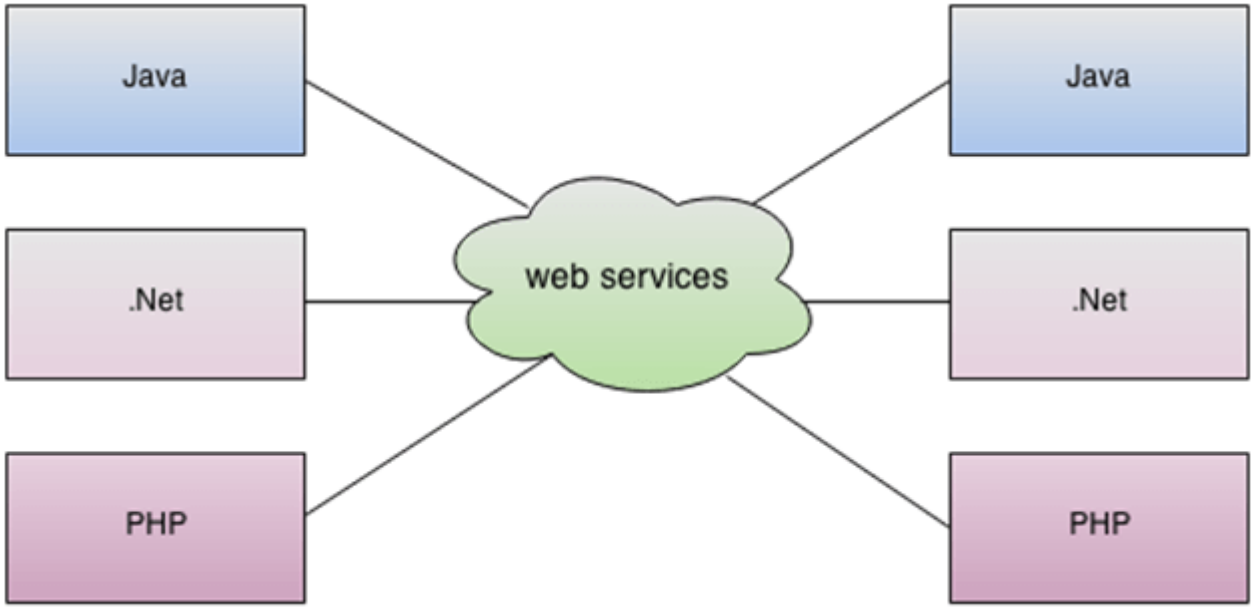
In the past, most programs written in any language have had to be recompiled, and sometimes rewritten, for each computer platform. One of the biggest advantages of Java is that programs only have to be written and compiled once. The Java on any platform will interpret the compiled bytecode into instructions understandable by the particular processor. However, the virtual machine handles one bytecode instruction at a time. Using the Java just-in-time compiler -- really a second compiler -- at the particular system platform compiles the bytecode into the particular system code. This is as though the program had been compiled initially on that platform. Once the code has been recompiled by the JIT compiler, it will usually run more quickly in a computer.

Web Service

A **Web Service** is can be defined by following ways:

- It is a client-server application or application component for communication.
- The method of communication between two devices over the network.
- It is a software system for the interoperable machine to machine communication.
- It is a collection of standards or protocols for exchanging information between two devices or application.

Let's understand it by the figure given below:

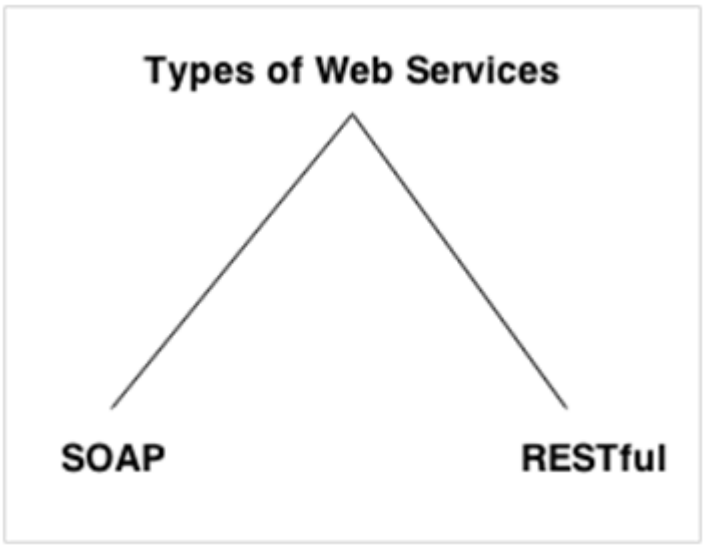


As you can see in the figure, Java, .net, and PHP applications can communicate with other applications through web service over the network. For example, the Java application can interact with Java, .Net, and PHP applications. So web service is a language independent way of communication.

Types of Web Services

There are mainly two types of web services.

1. SOAP web services.
2. RESTful web services.



Web Service Features

XML-Based

Web services use XML at data description and data transportation layers. Using XML exclude any networking, operating system, or platform binding. Web services-based operation is extremely interoperable at their core level.

Loosely Coupled

A client of a web service is not fixed to the web service directly. The web service interface can support innovation over time without negotiating the client's ability to communicate with the service. A tightly coupled system means that the client and server logic are closely tied to one another, indicating that if one interface changes, then another must be updated. Accepting a loosely coupled architecture tends to make software systems more manageable and allows more straightforward integration between various systems.

Coarse-Grained

Object-oriented technologies such as Java expose their functions through individual methods. A specific process is too fine an operation to provide any suitable capability at a corporate level. Building a Java program from scratch needed the creation of various fine-grained functions that are then collected into a coarse-grained role that is consumed by either a client or another service.

Businesses and the interfaces that they prove should be coarse-grained. Web services technology implement a natural method of defining coarse-grained services that approach the right amount of business logic.

Ability to be Synchronous or Asynchronous

Synchronicity specifies the binding of the client to the execution of the function. In synchronous invocations, the client blocks and delays in completing its service before continuing. Asynchronous operations grant a client to invoke a task and then execute other functions.

Asynchronous clients fetch their result at a later point in time, while synchronous clients receive their effect when the service has completed. Asynchronous capability is an essential method in enabling loosely coupled systems.

Supports Remote Procedure Calls (RPCs)

Web services allow consumers to invoke procedures, functions, and methods on remote objects using an XML-based protocol. Remote systems expose input and output framework that a web service must support.

Component development through Enterprise JavaBeans (EJBs) and .NET Components has more become a part of architectures and enterprise deployments over a previous couple of years. Both technologies are assigned and accessible through a variety of RPC mechanisms.

A web function supports RPC by providing services of its own, equivalent to those of a traditional role, or by translating incoming invocations into an invocation of an EJB or a .NET component.

Supports Document Exchange

One of the essential benefits of XML is its generic way of representing not only data but also complex documents. These documents can be as simple as describing a current address, or they can be as involved as defining an entire book or Request for Quotation (RFQ). Web services support the transparent transfer of documents to facilitate business integration.

Introduction to C#

C# is a general-purpose, modern and object-oriented programming language pronounced as **"C sharp"**. It was developed by Microsoft led by Anders Hejlsberg and his team within the .Net initiative and was approved by the European Computer

Manufacturers Association (ECMA) and International Standards Organization (ISO). C# is among the languages for Common Language Infrastructure and the current version of C# is version 7.2. C# is a lot similar to Java syntactically and is easy for the users who have knowledge of C, C++ or Java.

A bit about .Net Framework

.Net applications are multi-platform applications and framework can be used from languages like C++, C#, Visual Basic, COBOL etc. It is designed in a manner so that other languages can use it.

know more about .Net Framework

Why C#?

C# has many other reasons for being popular and in demand. Few of the reasons are mentioned below:

1. **Easy to start:** C# is a high-level language so it is closer to other popular programming languages like C, C++, and Java and thus becomes easy to learn for anyone.
2. **Widely used for developing Desktop and Web Application:** C# is widely used for developing web applications and Desktop applications. It is one of the most popular languages that is used in professional desktop. If anyone wants to create Microsoft apps, C# is their first choice.
3. **Community:**The larger the community the better it is as new tools and software will be developing to make it better. C# has a large community so the developments are done to make it exist in the system and not become extinct.
4. **Game Development:** C# is widely used in game development and will continue to dominate. C# integrates with Microsoft and thus has a large target audience. The C# features such as Automatic Garbage Collection, interfaces, object-oriented, etc. make C# a popular game developing language.

Beginning with C# programming:

Finding a Compiler:

There are various online IDEs such as GeeksforGeeks ide, CodeChef ide etc. which can be used to run C# programs without installing.

Windows: Since the C# is developed within .Net framework initiative by Microsoft, it provide various IDEs to run C# programs: Microsoft Visual Studio, Visual Studio Express, Visual Web Developer

Linux: Mono can be used to run C# programs on Linux.

Programming in C#:

Since the C# is a lot similar to other widely used languages syntactically, it is easier to code and learn in C#.

Programs can be written in C# in any of the widely used text editors like Notepad++, gedit, etc. or on any of the compilers. After writing the program save the file with the extension .cs.

Example: A simple program to print **Hello Geeks**

```
// C# program to print Hello Geeks

using System;



namespace HelloGeeksApp

{

    class HelloGeeks

    {

        // Main function

        static void Main(string[] args)

        {



            // Printing Hello Geeks

            Console.WriteLine("Hello Geeks");



            Console.ReadKey();

        }

    }

}
```

Output:
Hello Geeks

Explanation:
1. Comments: Comments are used for explaining code and are used in similar manner as in Java or C or C++. Compilers ignore the comment entries and does not execute them. Comments can be of single line or multiple lines.

Single line Comments:
Syntax:
```
// Single line comment
```

Multi line comments:
Syntax:
```
/* Multi line comments*/
```

2. using System: using keyword is used to include the System namespace in the program.
namespace declaration: A namespace is a collection of classes. The HelloGeeksApp namespace contains the class HelloGeeks.
3. class: The class contains the data and methods to be used in the program. Methods define the behavior of the class. Class **HelloGeeks** has only one method Main similar to JAVA.
4. static void Main(): static keyword tells us that this method is accessible without instantiating the class. **5. void** keywords tells that this method will not return anything. **Main()** method is the entry-point of our application. In our program, Main() method specifies its behavior with the statement Console.WriteLine("Hello Geeks"); .
6. Console.WriteLine(): WriteLine() is a method of the Console class defined in the System namespace.
7. Console.ReadKey(): This is for the VS.NET Users. This makes the program wait for a key press and prevents the screen from running and closing quickly.
Note: C# is case sensitive and all statements and expressions must end with semicolon (;).
Advantages of C#:

- C# is very efficient in managing the system. All the garbage is automatically collected in C#.
- There is no problem of memory leak in C# because of its high memory backup.
- Cost of maintenance is less and is safer to run as compared to other languages.
- C# code is compiled to a intermediate language (Common (.Net) Intermediate Language) which is a standard language, independently irrespective of the target operating system and architecture.

Disadvantages of C#:

- C# is less flexible as it depends alot on .Net framework.
- C# runs slowly and program needs to be compiled each time when any changes are made.

Applications:

- C# is widely used for developing desktop applications, web applications and web services.
- It is used in creating applications of Microsoft at a large scale.
- C# is also used in game development in Unity.

C# Identifiers

In programming languages, identifiers are used for identification purposes. Or in other words, identifiers are the user-defined name of the program components. In C#, an identifier can be a class name, method name, variable name, or label.
Example:

```
public class GFG {
    static public void Main ()
    {
          int x;
    }
}
```

Here the total number of identifiers present in the above example is 3 and the names of these identifiers are:

- **GFG:** Name of the class
- **Main:** Method name
- **x:** Variable name

Rules for defining identifiers in C#:
There are certain valid rules for defining a valid C# identifier. These rules should be followed, otherwise, we will get a compile-time error.

- The only allowed characters for identifiers are all alphanumeric characters(**[A-Z]**, **[a-z]**, **[0-9]**), '**_**' (underscore). For example "geek@" is not a valid C# identifier as it contain '@' – special character.
- Identifiers should not start with digits([0-9]). For example "123geeks" is not valid in the C# identifier.
- Identifiers should not contain white spaces.

- Identifiers are not allowed to use as keywords unless they include @ as a prefix. For example, **@as** is a valid identifier, but "**as**" is not because it is a keyword.
- C# identifiers allow Unicode Characters.
- C# identifiers are case-sensitive.
- C# identifiers cannot contain more than 512 characters.
- Identifiers do not contain two consecutive underscores in their name because such types of identifiers are used for the implementation.

Example:

- CSharp

```
// Simple C# program to illustrate identifiers

using System;



class GFG {



    // Main Method

    static public void Main()

    {



        // variable

        int a = 10;

        int b = 39;

        int c;



        // simple addition

        c = a + b;

        Console.WriteLine("The sum of two number is: {0}", c);

    }

}
```

Output:

```
The sum of two number is: 49
```

Below table shows the identifiers and keywords present in the above example:

Keywords	Identifiers
using	GFG
public	Main
static	a
void	b
int	c

C# Variables

A typical program uses various values that may change during its execution. For example, a program that performs some operations on the values entered by the user. The values entered by one user may differ from those entered by another user. Hence this makes it necessary to use variables as another user may not use the same values. When a user enters a new value that will be used in the process of operation, can store temporarily in the Random Access Memory of computer and these values in this part of memory vary throughout the execution and hence another term for this came which is known as **Variables**. So basically, a Variable is a placeholder of the information which can be changed at runtime. And variables allows to **Retrieve and Manipulate** the stored information.

Syntax:

```
type variable_name = value;
```

```
or
```

```
type variable_names;
```

Example:

```
char var = 'h'; // Declaring and Initializing character variable
```

```
int a, b, c; // Declaring variables a, b and c of int type
```

Characteristics of Variables:

- **name :** It must be a valid identifier. In above example, var is a valid identifier.
- **type :** It defines the types of information which is to be stored into the variable. In above example char is a type.

- **value :** It is the actual data which is to be stored in the variable. In above example 'h' is the value.

Rules for Naming Variables

- Variable names can contain the letters 'a-z' or 'A-Z' or digits 0-9 as well as the character '_'.
- The name of the variables cannot be started with a digit.
- The name of the variable cannot be any C# keyword say int, float, null, String, etc.

Examples:

- **Valid Variables Names**
- `int age;`
- `float _studentname;`
- **Invalid Variables Names**
- `int if; // "if" is a keyword`
- `float 12studentname; // Cannot start with digit`

Defining or Declaring a Variable

There are some rules that must be followed while declaring variables :

- specify its type (such as int)
- specify its name (such as age)
- Can provide initial value(such as 17)

Example :

```
int geeks;

float interest;
```

Initializing Variables

The term initializing means to assign some value to the variable. Basically, the actual use of variables comes under the initialization part. In C# each data type has some default value which is used when there is no explicitly set value for a given variable. Initialization can be done separately or may be with declaration.

Example :

```
int y = 7; // Declaring and initializing the variable at same time

int x; // Declaring variable x

x = 5; // initializing x with value 5
```

Two Ways for Initialization:

1. Compile time initialization
2. Run time initialization

1. Compile Time Initialization

It means to provide the value to the variable during the compilation of the program. If the programmer didn’t provide any value then the compiler will provide some default value to the variables in some cases. Generally, this type of initialization helpful when the programmer wants to provide some default value.

Example :

```
// C# program to demonstrate the

// Compile Time Initialization

using System;

class Geeks {



        // only declaration, compiler will

        // provide the default value 0 to it

        int y;



    // Main Method

    public static void Main(String []args)

    {



        // Compile Time Initialization of variable 'x'

        // Assigning value 32 to x

        int x = 32;



        // printing the value
```

```
        Console.WriteLine("Value of x is "+x);



        // creating object to access

        // the variable y

        Geeks gfg = new Geeks();



        // printing the value

        Console.WriteLine("Value of y is "+gfg.y);

    }

}
```

Output :

```
Value of x is 32

Value of y is 0
```

2. Run Time Initialization

In this, the user has to enter the value and that value is copied to the required variable. In this type of initialization, there is one more possibility in which value is assigned to variable after completion of a function call.

Example:

```
Input : 45

Output : Value of num is 45


Input : 27

Output : Value of num is 27
```

```
// C# program to demonstrate the
```

```
// Run Time Initialization

using System;

class Geeks {



    // Main Method

    public static void Main(String []args)

    {



        // Value will be taken from user

        // input and assigned to variable

        // num

        int num = Convert.ToInt32(Console.ReadLine());



        // printing the result

        Console.WriteLine("Value of num is " + num);



    }

}
```

Output :

```
Value of num is 45
```

Note: Here the Console.ReadLine() method asks the user to enter the value and later on it puts the same value in the “num” variable. Hence the value will be displayed according to the user input.

C# | Data Types

Data types specify the type of data that a valid C# variable can hold. C# is a **strongly typed programming language** because in C#, each type of data (such as integer, character, float, and so forth) is predefined as part of the programming language and all constants or variables defined for a given program must be described with one of the data types.

Data types in C# is mainly divided into three categories

- **Value Data Types**
- **Reference Data Types**
- **Pointer Data Type**

1. **Value Data Types :** In C#, the Value Data Types will directly store the variable value in memory and it will also accept both signed and unsigned literals. The derived class for these data types are **System.ValueType**. Following are **different Value Data Types** in C# programming language :
   - **Signed & Unsigned Integral Types :** There are 8 integral types which provide support for 8-bit, 16-bit, 32-bit, and 64-bit values in signed or unsigned form.

Alias	Type Name	Type	Size(bits)	Range	Default Value
sbyte	System.Sbyte	signed integer	8	-128 to 127	0
short	System.Int16	signed integer	16	-32768 to 32767	0
Int	System.Int32	signed integer	32	-2^{31} to $2^{31}-1$	0
long	System.Int64	signed integer	64	-2^{63} to $2^{63}-1$	0L
byte	System.byte	unsigned integer	8	0 to 255	0
ushort	System.UInt16	unsigned integer	16	0 to 65535	0
uint	System.UInt32	unsigned integer	32	0 to 2^{32}	0
ulong	System.UInt64	unsigned integer	64	0 to 2^{63}	0

- **Floating Point Types :**There are 2 floating point data types which contain the decimal point.
    - **Float**: It is **32-bit single-precision** floating point type. It has 7 digit Precision. To initialize a float variable, use the suffix f or F. Like, float x = 3.5F;. If the suffix F or f will not use then it is treated as double.
    - **Double**:It is **64-bit double-precision** floating point type. It has 14 – 15 digit Precision. To initialize a double variable, use the suffix d or D. But it is not mandatory to use suffix because by default floating data types are the double type.

Alias	Type name	Size(bits)	Range (aprox)	Default Value
float	System.Single	32	$\pm1.5 \times 10^{-45}$ to $\pm3.4 \times 10^{38}$	0.0F
double	System.Double	64	$\pm5.0 \times 10^{-324}$ to $\pm1.7 \times 10^{308}$	0.0D

- **Decimal Types :** The decimal type is a 128-bit data type suitable for financial and monetary calculations. It has 28-29 digit Precision. To initialize a decimal variable, use the suffix m or M. Like as, decimal x = 300.5m;. If the suffix m or M will not use then it is treated as double.

Alias	Type name	Size(bits)	Range (aprox)	Default value
decimal	System.Decimal	128	$\pm1.0 \times 10^{-28}$ to $\pm7.9228 \times 10^{28}$	0.0M

- **Character Types :** The character types represents a UTF-16 code unit or represents the 16-bit Unicode character.

Alias	Type name	Size In(Bits)	Range	Default value
char	System.Char	16	U +0000 to U +ffff	‘\0’

2. **Example :**

```
// C# program to demonstrate

// the above data types

using System;

namespace ValueTypeTest {
```

```
class GeeksforGeeks {

    // Main function
    static void Main()
    {

        // declaring character
        char a = 'G';

        // Integer data type is generally
        // used for numeric values
        int i = 89;

        short s = 56;

        // this will give error as number
        // is larger than short range
        // short s1 = 87878787878;

        // long uses Integer values which
```

```
// may signed or unsigned
long l = 4564;


// UInt data type is generally
// used for unsigned integer values
uint ui = 95;


ushort us = 76;
// this will give error as number is
// larger than short range


// ulong data type is generally
// used for unsigned integer values
ulong ul = 3624573;


// by default fraction value
// is double in C#
double d = 8.358674532;


// for float use 'f' as suffix
float f = 3.7330645f;
```

```
        // for float use 'm' as suffix
        decimal dec = 389.5m;


        Console.WriteLine("char: " + a);
        Console.WriteLine("integer: " + i);
        Console.WriteLine("short: " + s);
        Console.WriteLine("long: " + l);
        Console.WriteLine("float: " + f);
        Console.WriteLine("double: " + d);
        Console.WriteLine("decimal: " + dec);
        Console.WriteLine("Unsinged integer: " + ui);
        Console.WriteLine("Unsinged short: " + us);
        Console.WriteLine("Unsinged long: " + ul);
    }
}
}
```

3. **Output :**
4. char: G
5. integer: 89
6. short: 56
7. long: 4564
8. float: 3.733064
9. double: 8.358674532
10. decimal: 389.5

11. Unsinged integer: 95
12. Unsinged short: 76
13. Unsinged long: 3624573
14. **Example :**

```
// C# program to demonstrate the Sbyte
// signed integral data type
using System;
namespace ValueTypeTest {


class GeeksforGeeks {


    // Main function
    static void Main()
    {
        sbyte a = 126;


        // sbyte is 8 bit
        // singned value
        Console.WriteLine(a);


        a++;
        Console.WriteLine(a);
```

```
        // It overflows here because
        // byte can hold values
        // from -128 to 127
        a++;
        Console.WriteLine(a);

        // Looping back within
        // the range
        a++;
        Console.WriteLine(a);
    }
}
}
```

15. **Output :**

16. 126
17. 127
18. -128
19. -127

20. **Example :**

```
// C# program to demonstrate
// the byte data type
using System;
```

```
namespace ValueTypeTest {



class GeeksforGeeks {



    // Main function

    static void Main()

    {

        byte a = 0;



        // byte is 8 bit

        // unsigned value

        Console.WriteLine(a);



        a++;

        Console.WriteLine(a);



        a = 254;



        // It overflows here because

        // byte can hold values from

        // 0 to 255
```

```
        a++;

        Console.WriteLine(a);



        // Looping back within the range

        a++;

        Console.WriteLine(a);

    }

}

}
```

21. **Output :**
22. 0
23. 1
24. 255
25. 0

- **Boolean Types :** It has to be assigned either true or false value. Values of type bool are not converted implicitly or explicitly (with casts) to any other type. But the programmer can easily write conversion code.

Alias	Type name	Values
bool	System.Boolean	True / False

- **Example :**

```
// C# program to demonstrate the

// boolean data type

using System;

namespace ValueTypeTest {
```

```
class GeeksforGeeks {



// Main function

static void Main()

{



    // boolean data type

    bool b = true;

    if (b == true)

        Console.WriteLine("Hi Geek");

}

}

}
```

- **Output :**
- Hi Geek

26. **Reference Data Types :** The Reference Data Types will contain a memory address of variable value because the reference types won't store the variable value directly in memory. The built-in reference types are **string, object.**
    - **String :** It represents a sequence of Unicode characters and its type name is **System.String**. So, string and String are equivalent.
      Example :
    - `string s1 = "hello"; // creating through string keyword`
    - `String s2 = "welcome"; // creating through String class`
    - **Object :** In C#, all types, predefined and user-defined, reference types and value types, inherit directly or indirectly from Object. So basically it is the base class for all the data types in C#. Before assigning values, it needs type conversion. When a variable of a value type is converted to object, it's

called **boxing**. When a variable of type object is converted to a value type, it's called **unboxing**. Its type name is **System.Object**.

Example :

```
// C# program to demonstrate

// the Reference data types

using System;

namespace ValueTypeTest {



class GeeksforGeeks {



    // Main Function

    static void Main()

    {



        // declaring string

        string a = "Geeks";



        //append in a

        a+="for";

        a = a+"Geeks";

        Console.WriteLine(a);
```

```
        // declare object obj

        object obj;

        obj = 20;

        Console.WriteLine(obj);



        // to show type of object

        // using GetType()

        Console.WriteLine(obj.GetType());

    }

}

}
```

Output :

```
GeeksforGeeks

20

System.Int32
```

27. **Pointer Data Type :** The Pointer Data Types will contain a memory address of the variable value.
    To get the pointer details we have a two symbols **ampersand (&) and asterisk (*)**.
    ***ampersand (&)*:** It is Known as Address Operator. It is used to determine the address of a variable.
    ***asterisk (*)*:** It also known as Indirection Operator. It is used to access the value of an address.
    Syntax :

```
28.    type* identifier;
```

Example :

```
int* p1, p;   // Valid syntax

int *p1, *p;   // Invalid
```

Example :

```
// Note: This program will not work on

// online compiler

// Error: Unsafe code requires the `unsafe'

// command line option to be specified

// For its solution:

// Go to your project properties page and

// check under Build the checkbox Allow

// unsafe code.

using System;

namespace Pointerprogram {



class GFG {



    // Main function

    static void Main()

    {

        unsafe

        {



            // declare variable

            int n = 10;
```

```
            // store variable n address

            // location in pointer variable p

            int* p = &n;

            Console.WriteLine("Value :{0}", n);

            Console.WriteLine("Address :{0}", (int)p);

        }

    }

}

}
```

C# | String

In C#, ***string*** is a sequence of Unicode characters or array of characters. The range of Unicode characters will be U+0000 to U+FFFF. The array of characters is also termed as the *text*. So the string is the representation of the text. A string is an important concept and sometimes people get confused whether the string is a keyword or an object or a class. So let's clear out this concept.
A string is represented by class ***System.String***. The *"string" keyword* is an alias for System.String class and instead of writing System.String one can use *String* which is a shorthand for ***System.String*** class. So we can say string and String both can be used as an alias of System.String class. So string is an *object* of System.String class.
Example:
string s1 = "GeeksforGeeks"; // creating the string using string keyword
String s2 = "GFG"; // creating the string using String class
System.String s3 = "Pro Geek"; // creating the string using String class

The String class is defined in the **.NET** base class library. In other words a String object is a sequential collection of *System.Char* objects which represents a string. The maximum size of String object in memory is 2GB or about 1 billion

characters. **System.String class is immutable**, i.e once created its state cannot be altered.
Program: To illustrate how to **declare** the string and **initialize** the string. Also, below program show the declaration and initialization of a string in a single line.

- C#

```
// C# program to declare string using

// string, String and System.String

// and initialization of string

using System;

class Geeks {



    // Main Method

    static void Main(string[] args)

    {



        // declare a string Name using

        // "System.String" class

        System.String Name;



        // initialization of String

        Name = "Geek";
```

```
// declare a string id using

// using an alias(shorthand)

// "String" of System.String

// class

String id;



// initialization of String

id = "33";



// declare a string mrk using

// string keyword

string mrk;



// initialization of String

mrk = "97";



// Declaration and initialization of

// the string in a single line

string rank = "1";
```

```
        // Displaying Result

        Console.WriteLine("Name: {0}", Name);

        Console.WriteLine("Id: {0}", id);

        Console.WriteLine("Marks: {0}", mrk);

        Console.WriteLine("Rank: {0}", rank);

    }

}
```

Output:

```
Name: Geek

Id: 33

Marks: 97

Rank: 1
```

String Characteristics:

- It is a reference type.
- It's immutable(its state cannot be altered).
- It can contain nulls.
- It overloads the operator(==).

Differences between *String* and *System.String* :

The string is an alias for System.String. Both String and System.String means same and it will not affect the performance of the application. "string" is keyword in C#. So the main difference comes in the context, how to use these:

- The String is used for the declaration but System.String is used for accessing static string methods.
- The String is used to declare fields, properties etc. that it will use the predefined type System.String. It is the easy way to use.
- The String has to use the System.String class methods, such as String.SubString, String.IndexOf etc. The string is only an alias of System.String.

Note: In .NET, the text is stored as a sequential collection of the Char objects so there is no null-terminating character at the end of a C# string. Therefore a C# string can contain any number of embedded null characters ('\0').

String arrays: We can also create the array of string and assigns values to it. The string arrays can be created as follows:

Syntax:

```
String [] array_variable  =  new  String[Length_of_array]
```

- **Example:** To illustrate the creation of string arrays and assigning values to it

- C#

```
// C# program for an array of strings

using System;

class Geeks {



// Main Method

static void Main(string[] args)

{



    String[] str_arr = new String[3];



    // Initialising the array of strings

    str_arr[0] = "Geeks";

    str_arr[1] = "For";

    str_arr[2] = "Geeks";



    // printing String array

    for(int i = 0; i < 3; i++)

    {

        Console.WriteLine("value at Index position "+i+" is
"+str_arr[i]);
```

```
    }



}

}
```

Output:

```
value at Index position 0 is Geeks

value at Index position 1 is For

value at Index position 2 is Geeks
```

Reading String from User-Input: A string can be read out from the user input. ReadLine() method of console class is used to read a string from user input.

- **Example:**

- C#

```
// C# program to demonstrate Reading

// String from User-Input

using System;

class Geeks {



    // Main Method

    static void Main(string[] args)

    {



        Console.WriteLine("Enter the String");
```

```
        // Declaring a string object read_user

        // and taking the user input using

        // ReadLine() method

        String read_user = Console.ReadLine();



        // Displaying the user input

        Console.WriteLine("User Entered: " + read_user);



    }


}
```

Input:
Hello Geeks !

Output:
```
Enter the String

User Entered: Hello Geeks !
```

Different Ways for Creating a String:

- Create a string from a literal
- Create a string using concatenation
- Create a string using a constructor
- Create a string using a property or a method
- Create a string using formatting

Create a string from a literal: It is the most common way to create a string. In this, a user has to define string variable and then assign the value within the double quotes. We can use any type of characters within double quotes except some special character like a backslash (\).

- **Program:** To illustrate the string creation using literals

- C#

```
// C# program to demonstrate the
// string creation using literals
using System;
class Geeks {


    // Main Method
    static void Main(string[] args)
    {
        string str1 = "GeeksforGeeks";
        Console.WriteLine(str1);


        // Give Error Unrecognized escape sequence \H, \G, \p
        // string str3 = "X:\Home\GFG\Geeks.cs";
        // Console.WriteLine(str3);


        // using double slash \\
        string str2 = "X:\\Home\\GFG\\program.cs";
        Console.WriteLine(str2);


    }
}
```

Output:
GeeksforGeeks

X:\Home\GFG\program.cs

Create a string using concatenation: We can create a string by using string concatenation operator "+" in C#. To create a single string from any combination of String instances and string literals, the string concatenation operator (+) is used to combine or merge one or more string.

- **Program:** To illustrates the use of the string concatenation operator

- C#

```
// C# program to demonstrate the use of

// the string concatenation operator.

using System;

class Geeks {



    // Main Method

    public static void Main()

    {

        string s1 = "Geek";

        string s2 = "s";

        string s3 = "For";

        string s4 = "Geek";



        // using concatenation operator
```

```
        string str = s1 + s2 + s3 + s4 + "s";



        Console.WriteLine(str);

    }

}
```

Output:
GeeksForGeeks

Create a string using Constructor: The String class has been several overloaded constructors which take an array of characters or bytes. Some of the constructors include pointers to character arrays or signed byte arrays as parameters.

- **Program:** To illustrates Creation of a string using the constructor

- C#

```
// C# program to demonstrate the creation

// of string using the constructor

using System;

class Geeks {



    // Main Method

    public static void Main()

    {

        char[] chars = { 'G', 'E', 'E', 'K', 'S' };
```

```
// Create a string from a character array.
string str1 = new string(chars);
Console.WriteLine(str1);


// Create a string that consists of
// a character repeated 20 times.
string str2 = new string('G', 10);
Console.WriteLine(str2);


/* below comment part give the error
   for unsafe mode go through offline
   sbyte[] bytes = { 0x41, 0x42, 0x43,
   0x44, 0x45, 0x00 };
   string stringtoBytes = null;
   string stringtomChars = null;
    unsafe
    {
        fixed (sbyte* pbytes = bytes)
        {


            // Create a string from a pointer
            // to a signed byte array.
```

```
                    stringFromBytes = new string(pbytes);

                }



                fixed (char* pchars = chars)

                {



                    // Create a string from a pointer

                    // to a character array.

                    stringFromChars = new string(pchars);

                }

            }



            Console.WriteLine(stringtoBytes); // output : ABCDE

            Console.WriteLine(stringtoChars); // output : GEEKS */



    }

}
```

Output:

GEEKS

GGGGGGGGGG

Create a string using a Property or a Method: To retrieving a property or calling a method which always returns a string. For example, using methods of the String class to extract a substring from a larger string.

- **Program:** To illustrate the Creation of a string using a Property or a Method

- C#

```
// C# program to extract a substring from a larger

// string using methods of the String class

using System;

class Geeks {



    // Main Method

    public static void Main()

    {

        string sentence = "Geeks For Geeks";



        // Extract the second word.



        // taking the first space position value

        int startpos = sentence.IndexOf(" ") + 1;



        // taking the second space position value

        int endpos = sentence.IndexOf(" ", startpos) - startpos;



        // now extract second word from the sentence

        string wrd = sentence.Substring(startpos, endpos);
```

```
        Console.WriteLine(wrd);

    }

}
```

Output:
For

Create a string using Format: The “Format” method is used to convert the value or object to its string representation. The *String.Format* method returns a string.

- **Program:** To illustrate the creation of string using Format method

- C#

```
// C# method to illustrate the creation

// of string using format method

using System;

class Geeks {



    // Main Method

    public static void Main()

    {

        int no = 10;

        string cname = "BMW";

        string clr = "Red";
```

```
        // string creation using string.Format method

        string str = string.Format("{0} {1} Cars color " +

                    "are {2}", no.ToString(), cname, clr);

        Console.WriteLine(str);

    }

}
```

Output:

```
10 BMW Cars color are Red
```

String Class Properties: The String class has two properties as follows:

1. **Chars:** It is used to get the Char object at a specified position in the current String object.
2. **Length:** It is used to get the number of characters in the current String object. To know more about the string class properties please go to**String Properties in**

SHORT QUESTIONS

1.Define common language runtime?
2.Define base class library?
3.Write a short note on datatypes?
4.Write abour variables and constants?
5.Write about System collections?

LONG QUESTIONS

1.Discuss about common type system?
2.Writs a detailed note on garbage collection?
3.discuss about object oriented concept?
4.Write about Arrays and strings?
5.Write about c#statements?

UNIT-2

System Namespace

Classes

AccessViolationException	The exception that is thrown when there is an attempt to read or write protected memory.
Activator	Contains methods to create types of objects locally or remotely, or obtain references to existing remote objects. This class cannot be inherited.
AggregateException	Represents one or more errors that occur during application execution.
AppContext	Provides members for setting and retrieving data about an application's context.
AppDomain	Represents an application domain, which is an isolated environment where applications execute. This class cannot be inherited.
AppDomainSetup	Represents assembly binding information that can be added to an instance of AppDomain.
AppDomainUnloadedException	The exception that is thrown when an attempt is made to access an unloaded application domain.
ApplicationException	Serves as the base class for application-defined exceptions.
ApplicationId	Contains information used to uniquely identify a manifest-based application. This class cannot be inherited.
ArgumentException	The exception that is thrown when one of the arguments provided to a method is not valid.
ArgumentNullException	The exception that is thrown when a null reference (`Nothing` in Visual Basic) is passed to a method that does not accept it as a valid argument.
ArgumentOutOfRangeException	The exception that is thrown when the value of an argument is outside the allowable range of values as defined by the invoked method.
ArithmeticException	The exception that is thrown for errors in an arithmetic, casting, or conversion operation.
Array	Provides methods for creating, manipulating, searching, and sorting arrays, thereby serving as the base class for all arrays in the common language runtime.
ArrayTypeMismatchException	The exception that is thrown when an attempt is made to store an element of the wrong type within an array.
AssemblyLoadEventArgs	Provides data for the AssemblyLoad event.
Attribute	Represents the base class for custom attributes.
AttributeUsageAttribute	Specifies the usage of another attribute class. This class cannot be inherited.
BadImageFormatException	The exception that is thrown when the file image of a dynamic link library (DLL) or an executable program is invalid.
BitConverter	Converts base data types to an array of bytes, and an array of bytes to base data types.
Buffer	Manipulates arrays of primitive types.
CannotUnloadAppDomainException	The exception that is thrown when an attempt to unload an application domain fails.
CharEnumerator	Supports iterating over a String object and reading its individual characters. This class cannot be inherited.
CLSCompliantAttribute	Indicates whether a program element is compliant with the Common Language Specification (CLS). This class cannot be inherited.
Console	Represents the standard input, output, and error streams for console

	applications. This class cannot be inherited.
ConsoleCancelEventArgs	Provides data for the CancelKeyPress event. This class cannot be inherited.
ContextBoundObject	Defines the base class for all context-bound classes.
ContextMarshalException	The exception that is thrown when an attempt to marshal an object across a context boundary fails.
ContextStaticAttribute	Indicates that the value of a static field is unique for a particular context.
Convert	Converts a base data type to another base data type.
DataMisalignedException	The exception that is thrown when a unit of data is read from or written to an address that is not a multiple of the data size. This class cannot be inherited.
DBNull	Represents a nonexistent value. This class cannot be inherited.
Delegate	Represents a delegate, which is a data structure that refers to a static method or to a class instance and an instance method of that class.
DivideByZeroException	The exception that is thrown when there is an attempt to divide an integral or Decimal value by zero.
DllNotFoundException	The exception that is thrown when a DLL specified in a DLL import cannot be found.
DuplicateWaitObjectException	The exception that is thrown when an object appears more than once in an array of synchronization objects.
EntryPointNotFoundException	The exception that is thrown when an attempt to load a class fails due to the absence of an entry method.
Enum	Provides the base class for enumerations.
Environment	Provides information about, and means to manipulate, the current environment and platform. This class cannot be inherited.
EventArgs	Represents the base class for classes that contain event data, and provides a value to use for events that do not include event data.
Exception	Represents errors that occur during application execution.
ExecutionEngineException	The exception that is thrown when there is an internal error in the execution engine of the common language runtime. This class cannot be inherited.
FieldAccessException	The exception that is thrown when there is an invalid attempt to access a private or protected field inside a class.
FileStyleUriParser	A customizable parser based on the File scheme.
FlagsAttribute	Indicates that an enumeration can be treated as a bit field; that is, a set of flags.
FormatException	The exception that is thrown when the format of an argument is invalid, or when a composite format string is not well formed.
FormattableString	Represents a composite format string, along with the arguments to be formatted.
FtpStyleUriParser	A customizable parser based on the File Transfer Protocol (FTP) scheme.
GC	Controls the system garbage collector, a service that automatically reclaims unused memory.
GenericUriParser	A customizable parser for a hierarchical URI.
GopherStyleUriParser	A customizable parser based on the Gopher scheme.

HttpStyleUriParser	A customizable parser based on the HTTP scheme.
IndexOutOfRangeException	The exception that is thrown when an attempt is made to access an element of an array or collection with an index that is outside its bounds.
InsufficientExecutionStackException	The exception that is thrown when there is insufficient execution stack available to allow most methods to execute.
InsufficientMemoryException	The exception that is thrown when a check for sufficient available memory fails. This class cannot be inherited.
InvalidCastException	The exception that is thrown for invalid casting or explicit conversion.
InvalidOperationException	The exception that is thrown when a method call is invalid for the object's current state.
InvalidProgramException	The exception that is thrown when a program contains invalid Microsoft intermediate language (MSIL) or metadata. Generally this indicates a bug in the compiler that generated the program. This exception is also thrown when internal runtime implementation limits have been exceeded by the program.
InvalidTimeZoneException	The exception that is thrown when time zone information is invalid.
Lazy<T>	Provides support for lazy initialization.
Lazy<T,TMetadata>	Provides a lazy indirect reference to an object and its associated metadata for use by the Managed Extensibility Framework.
LdapStyleUriParser	A customizable parser based on the Lightweight Directory Access Protocol (LDAP) scheme.
LoaderOptimizationAttribute	Used to set the default loader optimization policy for the main method of an executable application.
LocalDataStoreSlot	Encapsulates a memory slot to store local data. This class cannot be inherited.
MarshalByRefObject	Enables access to objects across application domain boundaries in applications that support remoting.
Math	Provides constants and static methods for trigonometric, logarithmic, and other common mathematical functions.
MathF	Provides constants and static methods for trigonometric, logarithmic, and other common mathematical functions.
MemberAccessException	The exception that is thrown when an attempt to access a class member fails.
MemoryExtensions	Provides extension methods for the memory- and span-related types, such as Memory<T>, ReadOnlyMemory<T>, Span<T>, and ReadOnlySpan<T>.
MethodAccessException	The exception that is thrown when there is an invalid attempt to access a method, such as accessing a private method from partially trusted code.
MissingFieldException	The exception that is thrown when there is an attempt to dynamically access a field that does not exist. If a field in a class library has been removed or renamed, recompile any assemblies that reference that library.
MissingMemberException	The exception that is thrown when there is an attempt to dynamically access a class member that does not exist or that is not declared as public. If a member in a class library has been removed or renamed, recompile any assemblies that reference that library.

MissingMethodException	The exception that is thrown when there is an attempt to dynamically access a method that does not exist.
MTAThreadAttribute	Indicates that the COM threading model for an application is multithreaded apartment (MTA).
MulticastDelegate	Represents a multicast delegate; that is, a delegate that can have more than one element in its invocation list.
MulticastNotSupportedException	The exception that is thrown when there is an attempt to combine two delegates based on the Delegate type instead of the MulticastDelegate type. This class cannot be inherited.
NetPipeStyleUriParser	A parser based on the NetPipe scheme for the "Indigo" system.
NetTcpStyleUriParser	A parser based on the NetTcp scheme for the "Indigo" system.
NewsStyleUriParser	A customizable parser based on the news scheme using the Network News Transfer Protocol (NNTP).
NonSerializedAttribute	Indicates that a field of a serializable class should not be serialized. This class cannot be inherited.
NotFiniteNumberException	The exception that is thrown when a floating-point value is positive infinity, negative infinity, or Not-a-Number (NaN).
NotImplementedException	The exception that is thrown when a requested method or operation is not implemented.
NotSupportedException	The exception that is thrown when an invoked method is not supported, or when there is an attempt to read, seek, or write to a stream that does not support the invoked functionality.
Nullable	Supports a value type that can be assigned `null`. This class cannot be inherited.
NullReferenceException	The exception that is thrown when there is an attempt to dereference a null object reference.
Object	Supports all classes in the .NET class hierarchy and provides low-level services to derived classes. This is the ultimate base class of all .NET classes; it is the root of the type hierarchy.
ObjectDisposedException	The exception that is thrown when an operation is performed on a disposed object.
ObsoleteAttribute	Marks the program elements that are no longer in use. This class cannot be inherited.
OperatingSystem	Represents information about an operating system, such as the version and platform identifier. This class cannot be inherited.
OperationCanceledException	The exception that is thrown in a thread upon cancellation of an operation that the thread was executing.
OutOfMemoryException	The exception that is thrown when there is not enough memory to continue the execution of a program.
OverflowException	The exception that is thrown when an arithmetic, casting, or conversion operation in a checked context results in an overflow.
ParamArrayAttribute	Indicates that a method will allow a variable number of arguments in its invocation. This class cannot be inherited.
PlatformNotSupportedException	The exception that is thrown when a feature does not run on a particular platform.
Progress<T>	Provides an IProgress<T> that invokes callbacks for each reported progress value.

Random	Represents a pseudo-random number generator, which is an algorithm that produces a sequence of numbers that meet certain statistical requirements for randomness.
RankException	The exception that is thrown when an array with the wrong number of dimensions is passed to a method.
ResolveEventArgs	Provides data for loader resolution events, such as the TypeResolve, ResourceResolve, ReflectionOnlyAssemblyResolve and AssemblyResolve events.
SerializableAttribute	Indicates that a class can be serialized using binary or XML serialization. This class cannot be inherited.
StackOverflowException	The exception that is thrown when the execution stack exceeds the stack size. This class cannot be inherited.
STAThreadAttribute	Indicates that the COM threading model for an application is single-threaded apartment (STA).
String	Represents text as a sequence of UTF-16 code units.
StringComparer	Represents a string comparison operation that uses specific case and culture-based or ordinal comparison rules.
StringNormalizationExtensions	Provides extension methods to work with string normalization.
SystemException	Serves as the base class for system exceptions namespace.
ThreadStaticAttribute	Indicates that the value of a static field is unique for each thread.
TimeoutException	The exception that is thrown when the time allotted for a process or operation has expired.
TimeZone	Represents a time zone.
TimeZoneInfo	Represents any time zone in the world.
TimeZoneInfo.AdjustmentRule	Provides information about a time zone adjustment, such as the transition to and from daylight saving time.
TimeZoneNotFoundException	The exception that is thrown when a time zone cannot be found.
Tuple	Provides static methods for creating tuple objects.
Tuple<T1>	Represents a 1-tuple, or singleton.
Tuple<T1,T2>	Represents a 2-tuple, or pair.
Tuple<T1,T2,T3>	Represents a 3-tuple, or triple.
Tuple<T1,T2,T3,T4>	Represents a 4-tuple, or quadruple.
Tuple<T1,T2,T3,T4,T5>	Represents a 5-tuple, or quintuple.
Tuple<T1,T2,T3,T4,T5,T6>	Represents a 6-tuple, or sextuple.
Tuple<T1,T2,T3,T4,T5,T6,T7>	Represents a 7-tuple, or septuple.
Tuple<T1,T2,T3,T4,T5,T6,T7,TRest>	Represents an *n*-tuple, where *n* is 8 or greater.
TupleExtensions	Provides extension methods for tuples to interoperate with language support for tuples in C#.
Type	Represents type declarations: class types, interface types, array types, value types, enumeration types, type parameters, generic type definitions, and open or closed constructed generic types.
TypeAccessException	The exception that is thrown when a method attempts to use a type that it does not have access to.
TypeInitializationException	The exception that is thrown as a wrapper around the exception thrown by the class initializer. This class cannot be inherited.
TypeLoadException	The exception that is thrown when type-loading failures occur.
TypeUnloadedException	The exception that is thrown when there is an attempt to access an

	unloaded class.
UnauthorizedAccessException	The exception that is thrown when the operating system denies access because of an I/O error or a specific type of security error.
UnhandledExceptionEventArgs	Provides data for the event that is raised when there is an exception that is not handled in any application domain.
Uri	Provides an object representation of a uniform resource identifier (URI) and easy access to the parts of the URI.
UriBuilder	Provides a custom constructor for uniform resource identifiers (URIs) and modifies URIs for the Uri class.
UriFormatException	The exception that is thrown when an invalid Uniform Resource Identifier (URI) is detected.
UriParser	Parses a new URI scheme. This is an abstract class.
UriTypeConverter	Converts a String type to a Uri type, and vice versa.
ValueType	Provides the base class for value types.
Version	Represents the version number of an assembly, operating system, or the common language runtime. This class cannot be inherited.
WeakReference	Represents a weak reference, which references an object while still allowing that object to be reclaimed by garbage collection.
WeakReference<T>	Represents a typed weak reference, which references an object while still allowing that object to be reclaimed by garbage collection.

Structs

ArgIterator	Represents a variable-length argument list; that is, the parameters of a function that takes a variable number of arguments.
ArraySegment<T>.Enumerator	Provides an enumerator for the elements of an ArraySegment<T>.
ArraySegment<T>	Delimits a section of a one-dimensional array.
Boolean	Represents a Boolean (`true` or `false`) value.
Byte	Represents an 8-bit unsigned integer.
Char	Represents a character as a UTF-16 code unit.
ConsoleKeyInfo	Describes the console key that was pressed, including the character represented by the console key and the state of the SHIFT, ALT, and CTRL modifier keys.
DateOnly	Represents dates with values ranging from January 1, 0001 Anno Domini (Common Era) through December 31, 9999 A.D. (C.E.) in the Gregorian calendar.
DateTime	Represents an instant in time, typically expressed as a date and time of day.
DateTimeOffset	Represents a point in time, typically

	expressed as a date and time of day, relative to Coordinated Universal Time (UTC).
Decimal	Represents a decimal floating-point number.
Double	Represents a double-precision floating-point number.
GCGenerationInfo	Represents the size and the fragmenation of a generation on entry and on exit of the GC reported in GCMemoryInfo.
GCMemoryInfo	Provides a set of APIs that can be used to retrieve garbage collection information.
Guid	Represents a globally unique identifier (GUID).
Half	Represents a half-precision floating-point number.
HashCode	Combines the hash code for multiple values into a single hash code.
Index	Represents a type that can be used to index a collection either from the beginning or the end.
Int128	Represents a 128-bit signed integer.
Int16	Represents a 16-bit signed integer.
Int32	Represents a 32-bit signed integer.
Int64	Represents a 64-bit signed integer.
IntPtr	Represents a signed integer where the bit-width is the same as a pointer.
Memory<T>	Represents a contiguous region of memory.
MemoryExtensions.TryWriteInterpolatedStringHandler	Provides a handler used by the language compiler to format interpolated strings into character spans.
ModuleHandle	Represents a runtime handle for a module.
Nullable<T>	Represents a value type that can be assigned `null`.
Range	Represents a range that has start and end indexes.
ReadOnlyMemory<T>	Represents a contiguous region of memory, similar to ReadOnlySpan<T>. Unlike ReadOnlySpan<T>, it is not a byref-like type.
ReadOnlySpan<T>.Enumerator	Provides an enumerator for the elements of a ReadOnlySpan<T>.
ReadOnlySpan<T>	Provides a type-safe and memory-safe read-only representation of a contiguous region of arbitrary memory.
RuntimeArgumentHandle	References a variable-length argument list.

RuntimeFieldHandle	Represents a field using an internal metadata token.
RuntimeMethodHandle	RuntimeMethodHandle is a handle to the internal metadata representation of a method.
RuntimeTypeHandle	Represents a type using an internal metadata token.
SByte	Represents an 8-bit signed integer.
SequencePosition	Represents a position in a non-contiguous set of memory. Properties of this type should not be interpreted by anything but the type that created it.
Single	Represents a single-precision floating-point number.
Span<T>.Enumerator	Provides an enumerator for the elements of a Span<T>.
Span<T>	Provides a type-safe and memory-safe representation of a contiguous region of arbitrary memory.
TimeOnly	Represents a time of day, as would be read from a clock, within the range 00:00:00 to 23:59:59.9999999.
TimeSpan	Represents a time interval.
TimeZoneInfo.TransitionTime	Provides information about a specific time change, such as the change from daylight saving time to standard time or vice versa, in a particular time zone.
TypedReference	Describes objects that contain both a managed pointer to a location and a runtime representation of the type that may be stored at that location.
UInt128	Represents a 128-bit unsigned integer.
UInt16	Represents a 16-bit unsigned integer.
UInt32	Represents a 32-bit unsigned integer.
UInt64	Represents a 64-bit unsigned integer.
UIntPtr	Represents an unsigned integer where the bit-width is the same as a pointer.
UriCreationOptions	Provides options that control how a Uri is created and behaves.
ValueTuple	Provides static methods for creating value tuples.
ValueTuple<T1>	Represents a value tuple with a single component.
ValueTuple<T1,T2>	Represents a value tuple with 2 components.
ValueTuple<T1,T2,T3>	Represents a value tuple with 3 components.
ValueTuple<T1,T2,T3,T4>	Represents a value tuple with 4

	components.
ValueTuple<T1,T2,T3,T4,T5>	Represents a value tuple with 5 components.
ValueTuple<T1,T2,T3,T4,T5,T6>	Represents a value tuple with 6 components.
ValueTuple<T1,T2,T3,T4,T5,T6,T7>	Represents a value tuple with 7 components.
ValueTuple<T1,T2,T3,T4,T5,T6,T7,TRest>	Represents an *n*-value tuple, where *n* is 8 or greater.
Void	Specifies a return value type for a method that does not return a value.

Multithreading in C#

Multithreading is a specialized form of multitasking and a multitasking is the feature that allows your computer to run two or more programs concurrently. In general, there are two types of multitasking: process-based and thread-based.

Process-based multitasking handles the concurrent execution of programs. Thread-based multitasking deals with the concurrent execution of pieces of the same program.

A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution.

C does not contain any built-in support for multithreaded applications. Instead, it relies entirely upon the operating system to provide this feature.

This tutorial assumes that you are working on Linux OS and we are going to write multi-threaded C program using POSIX. POSIX Threads, or Pthreads provides API which are available on many Unix-like POSIX systems such as FreeBSD, NetBSD, GNU/Linux, Mac OS X and Solaris.

The following routine is used to create a POSIX thread −

```
#include <pthread.h>

pthread_create (thread, attr, start_routine, arg)
```

Here, **pthread_create** creates a new thread and makes it executable. This routine can be called any number of times from anywhere within your code. Here is the description of the parameters.

Parameter	Description
thread	An opaque, unique identifier for the new thread returned by the subroutine.
attr	An opaque attribute object that may be used to set thread attributes. You can specify a thread attributes object, or NULL for the default values.

Parameter	Description
start_routine	The C routine that the thread will execute once it is created.
arg	A single argument that may be passed to start_routine. It must be passed by reference as a pointer cast of type void. NULL may be used if no argument is to be passed.

The maximum number of threads that may be created by a process is implementation dependent. Once created, threads are peers, and may create other threads. There is no implied hierarchy or dependency between threads.

Terminating Threads

There is following routine which we use to terminate a POSIX thread –

```
#include <pthread.h>
pthread_exit (status)
```

Here **pthread_exit** is used to explicitly exit a thread. Typically, the pthread_exit() routine is called after a thread has completed its work and is no longer required to exist.

If main() finishes before the threads it has created, and exits with pthread_exit(), the other threads will continue to execute. Otherwise, they will be automatically terminated when main() finishes.

Example Code

```
#include <iostream>
#include <cstdlib>
#include <pthread.h>
using namespace std;
#define NUM_THREADS 5
void *PrintHello(void *threadid) {
   long tid;
   tid = (long)threadid;
   printf("Hello World! Thread ID, %d
", tid);
   pthread_exit(NULL);
}
int main () {
   pthread_t threads[NUM_THREADS];
   int rc;
   int i;
   for( i = 0; i < NUM_THREADS; i++ ) {
```

```
    cout << "main() : creating thread, " << i << endl;
    rc = pthread_create(&threads[i], NULL, PrintHello, (void *)i);
    if (rc) {
      printf("Error:unable to create thread, %d
", rc);
      exit(-1);
    }
  }
  pthread_exit(NULL);
}
```

Output

```
$gcc test.cpp -lpthread
$./a.out
main() : creating thread, 0
main() : creating thread, 1
main() : creating thread, 2
main() : creating thread, 3
main() : creating thread, 4
Hello World! Thread ID, 0
Hello World! Thread ID, 1
Hello World! Thread ID, 2
Hello World! Thread ID, 3
Hello World! Thread ID, 4
```

Socket Programming in C#

Socket programming is a way of connecting two nodes on a network to communicate with each other. Basically, it is a one-way Client and Server setup where a Client connects, sends messages to the server and the server shows them using socket connection. One socket (node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while the client reaches out to the server. Before going deeper into Server and Client code, it is strongly recommended to go through **TCP/IP Model**.

Client Side Programming

Before creating client's socket a user must decide what '*IP Address*' that he want to connect to, in this case, it is the *localhost*. At the same time, we also need the '*Family*' method that will belong to the socket itself. Then, through the '**connect**' method, we will connect the socket to the server. Before sending any message, it must be converted into a byte array. Then and only then, it can be sent to the server through the '***send***' method. Later, thanks to the '***receive***' method we are going to get a byte array as answer by the server. It is notable that just like in the C language, the 'send' and 'receive' methods still return the number of bytes sent or received.

- C#

```
// A C# program for Client

using System;

using System.Net;

using System.Net.Sockets;

using System.Text;



namespace Client {



class Program {



// Main Method

static void Main(string[] args)

{

    ExecuteClient();

}
```

```
// ExecuteClient() Method

static void ExecuteClient()

{



    try {



        // Establish the remote endpoint

        // for the socket. This example

        // uses port 11111 on the local

        // computer.

        IPHostEntry ipHost = Dns.GetHostEntry(Dns.GetHostName());

        IPAddress ipAddr = ipHost.AddressList[0];

        IPEndPoint localEndPoint = new IPEndPoint(ipAddr, 11111);



        // Creation TCP/IP Socket using

        // Socket Class Constructor

        Socket sender = new Socket(ipAddr.AddressFamily,

                    SocketType.Stream, ProtocolType.Tcp);



        try {
```

```
            // Connect Socket to the remote

            // endpoint using method Connect()

            sender.Connect(localEndPoint);



            // We print EndPoint information

            // that we are connected

            Console.WriteLine("Socket connected to -> {0} ",

                          sender.RemoteEndPoint.ToString());



            // Creation of message that

            // we will send to Server

            byte[] messageSent = Encoding.ASCII.GetBytes("Test
Client<EOF>");

            int byteSent = sender.Send(messageSent);



            // Data buffer

            byte[] messageReceived = new byte[1024];



            // We receive the message using

            // the method Receive(). This

            // method returns number of bytes

            // received, that we'll use to
```

```
            // convert them to string
            int byteRecv = sender.Receive(messageReceived);
            Console.WriteLine("Message from Server -> {0}",
                  Encoding.ASCII.GetString(messageReceived,
                                           0, byteRecv));

            // Close Socket using
            // the method Close()
            sender.Shutdown(SocketShutdown.Both);
            sender.Close();
        }

        // Manage of Socket's Exceptions
        catch (ArgumentNullException ane) {

            Console.WriteLine("ArgumentNullException : {0}",
ane.ToString());
        }

        catch (SocketException se) {

            Console.WriteLine("SocketException : {0}", se.ToString());
```

```
        }



        catch (Exception e) {

            Console.WriteLine("Unexpected exception : {0}",
e.ToString());

        }

    }



    catch (Exception e) {



        Console.WriteLine(e.ToString());

    }

}

}

}
```

Server Side Programming

In the same way, we need an ‘IP address’ that identifies the server in order to let the clients to connect. After creating the socket, we call the ‘***bind***‘ method which binds the IP to the socket. Then, call the ‘***listen***‘ method. This operation is responsible for creating the waiting queue which will be related to every opened ‘*socket*‘. The ‘***listen***‘ method takes as input the maximum number of clients that can stay in the waiting queue. As stated above, there is communication with the client through ‘*send*‘ and ‘*receive*‘ methods.
Note: Don’t forget the conversion into a byte array.

- C#

```
// A C# Program for Server

using System;

using System.Net;

using System.Net.Sockets;

using System.Text;



namespace Server {



class Program {



// Main Method

static void Main(string[] args)

{

    ExecuteServer();

}



public static void ExecuteServer()

{

    // Establish the local endpoint

    // for the socket. Dns.GetHostName

    // returns the name of the host
```

```
// running the application.
IPHostEntry ipHost = Dns.GetHostEntry(Dns.GetHostName());
IPAddress ipAddr = ipHost.AddressList[0];
IPEndPoint localEndPoint = new IPEndPoint(ipAddr, 11111);

// Creation TCP/IP Socket using
// Socket Class Constructor
Socket listener = new Socket(ipAddr.AddressFamily,
             SocketType.Stream, ProtocolType.Tcp);

try {

    // Using Bind() method we associate a
    // network address to the Server Socket
    // All client that will connect to this
    // Server Socket must know this network
    // Address
    listener.Bind(localEndPoint);

    // Using Listen() method we create
    // the Client list that will want
    // to connect to Server
```

```
listener.Listen(10);

while (true) {

    Console.WriteLine("Waiting connection ... ");

    // Suspend while waiting for
    // incoming connection Using
    // Accept() method the server
    // will accept connection of client
    Socket clientSocket = listener.Accept();

    // Data buffer
    byte[] bytes = new Byte[1024];
    string data = null;

    while (true) {

        int numByte = clientSocket.Receive(bytes);

        data += Encoding.ASCII.GetString(bytes,
```

```
                                        0, numByte);


            if (data.IndexOf("<EOF>") > -1)
                break;
        }

        Console.WriteLine("Text received -> {0} ", data);
        byte[] message = Encoding.ASCII.GetBytes("Test Server");

        // Send a message to Client
        // using Send() method
        clientSocket.Send(message);

        // Close client Socket using the
        // Close() method. After closing,
        // we can use the closed Socket
        // for a new Client Connection
        clientSocket.Shutdown(SocketShutdown.Both);
        clientSocket.Close();
    }
}
```

```
    catch (Exception e) {

        Console.WriteLine(e.ToString());

    }

}

}

}
```

To run on Terminal or Command Prompt:

- First save the files with .cs extension. Suppose we saved the files as *client.cs* and *server.cs*.
- Then compile both the files by executing the following commands:

```
$ csc client.cs

$ csc server.cs
```

- After successful compilation opens the two cmd one for Server and another for Client and first try to execute the server as follows

Data Handling

Nowadays, managing and representing data systematically has become very important especially when the data provided is large and complex, This is when Data Handling comes into the picture. The definition of Data handling is in the title itself, that is, Handling the data in such a way that it becomes easier for people to understand and comprehend the given information. Hence, The **process of collecting, Recording**, **and representing data** in some form of graph or chart to make it easy for people to understand is called **Data handling.**
Statistics is another term for data handling, and it is useful not only in the field of Math and Science but also in the fields where the representation of data is required. Let’s learn about some forms of Data handling, and how they work.

Pictographs

A pictograph is the pictorial representation of any data given to us in written form. It can be said that pictograph used to be the earliest form of conversation, since way back in time, people have communicated mostly through pictures with each other since the languages were not present.

Indeed, Pictograph plays a role in our day-to-day life too. For instance, when a friend tells us a story, we start imagining the story in our head and that makes it both easy to understand and easy to remember for a long time for us.

Drawing a Pictograph

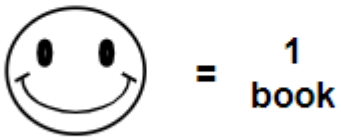
Let's learn to draw the pictograph with the help of an example,

Example: In a reading competition, there were three students participating- Rahul, Saumya, Ankush. They were supposed to read as many books as they can in an hour. Rahul read 3 books, Saumya read 2 books and Ankush read 4 books. Draw the pictograph for the information.

Solution:

There are some basic steps to draw a Pictograph:

- *Decide the particular picture/pictures that is required to represent data, make sure that the picture is a little related in order to memorize information easily.*
- *Here, to successfully read a book, a smiley is denoted.*



- *Now, draw the pictures according to information presented, for example, there will be 3 smilies for Rahul as he completed 3 books in an hour*

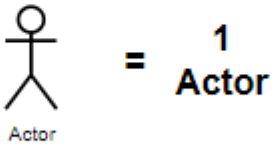
Sample Questions on Pictograph

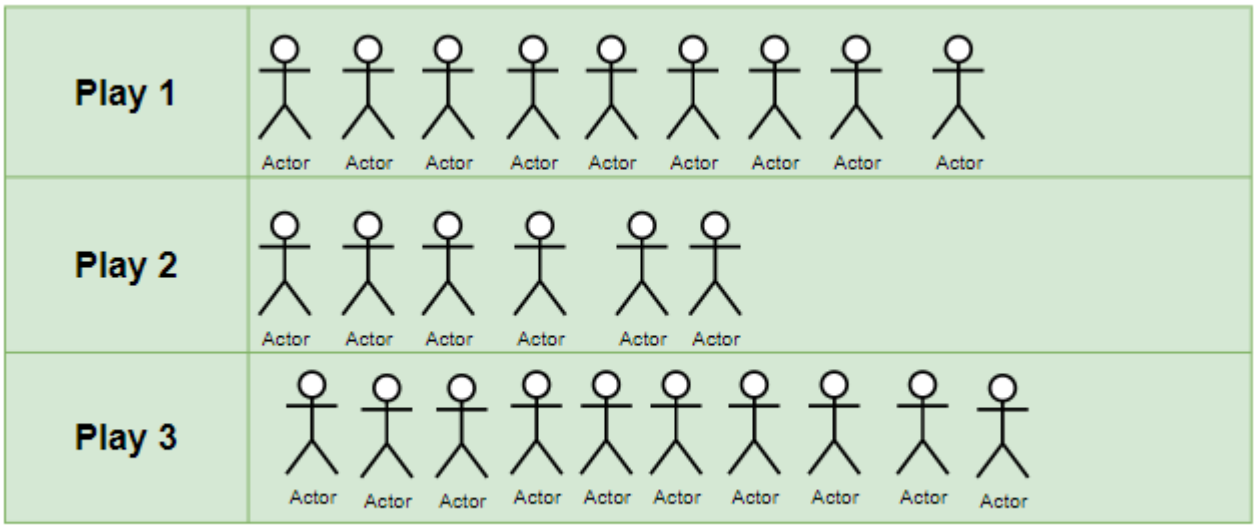
Question 1: In a Theater, there are 3 Plays with different amounts of actors participating in each play. In play 1, there are 9 actors, in play 2, there are 3 lesser actors, and the number of actors in play 3 is one lesser than play 1. Draw the Pictograph for the information given and analyze in which play, the stage will be most crowded.

Solution:

From the information given in the question, we can say that play 1 has 9 actors, play 2 has 6 actors and play 3 has 10 actors

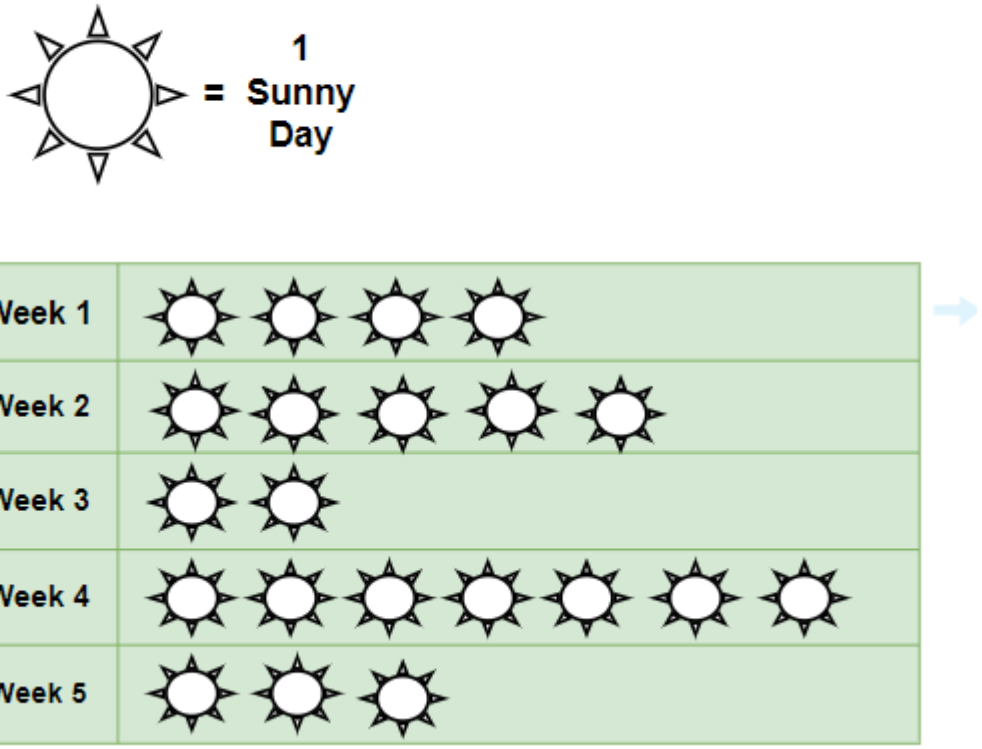
Representing actors in the pictorial form as,





Therefore, we can conclude that Play 3 has the stage most crowded as it has 10 actors acting on stage.

Question 2: In a Weather Report conducted for 5 weeks continuously, it was noted that not all days are sunny days in the season of spring. The observation said that week 1 had 4 sunny days, week 2 had 5 sunny days, week 3 had only 2 sunny days, week 4 had sunny days in the entire week, and week 5 had only 3 sunny days.
Draw a Pictograph for the information given above.
Solution:
Representing sunny days in pictorial form for better understanding,



Bar Graphs

The graphical representation of any quantity, number or data in the form of bars is called a bar graph. With the help of Bar Graph, not only the data look neat and understanding but also it is easier to compare the data given.

Types of Bar Graph

- **Vertical Bar Graph**
- **Horizontal Bar Graph**

Vertical Bar Graph

These are the most common bar graph we come across, the bars of grouped data in vertical bar graphs lie vertically. Sometimes when the data categorized have long names, then Horizontal bar graphs are preferred since, in vertical bar graphs, there is not much space on the x-axis.

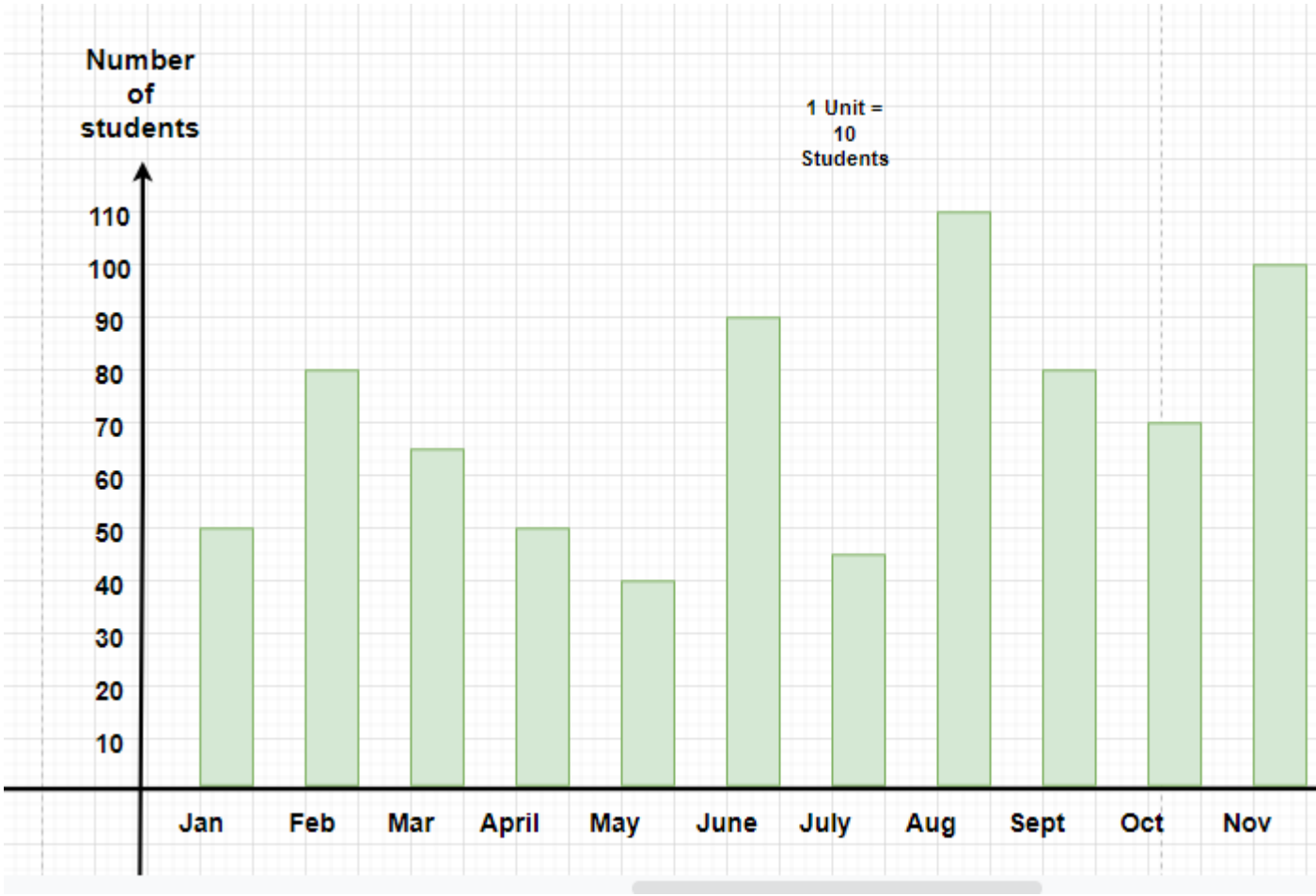
Question: There are 800 students in a school and the table for their birthdays in all 12 months is given below, Draw the Vertical Bar graph and answer,

Months	No. of students
January	50
February	80
March	65
April	50
May	40
June	90
July	45
August	110
September	80
October	70
November	100
December	20

1. **Maximum number of students have their birthdays in which month?**
2. **Which two months have equal number of birthday?**
3. **The minimum number of birthdays occur in which month?**

Solution:

The vertical bar graph for the table given in the question will be,



From the Bar graph we can figure out the answer of the questions

1. *August is that month in which maximum birthdays are happening, since the bar above august is the longest(there are 110 students whose birthday come in August)*
2. *From the graph, we can tell that January and April have equal lengths of bars, That means they have the same number of birthdays (both have 50 birthdays)*
3. *The minimum number of birthdays occur in December since it has the smallest bar.(20 students have their birthdays in December.*

Horizontal Bar Graph

The graphs that have their rectangular bars lying horizontally, which means that the frequency of the data lie on the x-axis while the categories of the data lie on the y-axis are known as Horizontal bar graphs.

Horizontal bar graphs are preferred when the name of the categories of data are long and the minimum space on the x-axis is not sufficie

SHORT QUESTIONS

1.write about namespace system?

2.Write a short note on multithreading?

3.write about input and output system?

4.Write about sockets?

5.Write a short note on error handling?

LONG QUESTIONS

1.Discuss about sockets and networking?

2.Write a detailed note on Data handling?

3.Discuss about c# Applicaton?

4.Write a detailed note on Window forms?

5.Write about networking and sockets?

UNIT-3

Delegates and events in C#

Delegates provide a *late binding* mechanism in .NET. Late Binding means that you create an algorithm where the caller also supplies at least one method that implements part of the algorithm.

For example, consider sorting a list of stars in an astronomy application. You may choose to sort those stars by their distance from the earth, or the magnitude of the star, or their perceived brightness.

In all those cases, the Sort() method does essentially the same thing: arranges the items in the list based on some comparison. The code that compares two stars is different for each of the sort orderings.

These kinds of solutions have been used in software for half a century. The C# language delegate concept provides first class language support, and type safety around the concept.

As you'll see later in this series, the C# code you write for algorithms like this is type safe. The compiler ensures that the types match for arguments and return types.

Function pointers were added to C# 9 for similar scenarios, where you need more control over the calling convention. The code associated with a delegate is invoked using a virtual method added to a delegate type. Using function pointers, you can specify different conventions.

Language Design Goals for Delegates

The language designers enumerated several goals for the feature that eventually became delegates.

The team wanted a common language construct that could be used for any late binding algorithms. Delegates enable developers to learn one concept, and use that same concept across many different software problems.

Second, the team wanted to support both single and multicast method calls. (Multicast delegates are delegates that chain together multiple method calls. You'll see examples later in this series.)

The team wanted delegates to support the same type safety that developers expect from all C# constructs.

Finally, the team recognized an event pattern is one specific pattern where delegates, or any late binding algorithm, is useful. The team wanted to ensure the code for delegates could provide the basis for the .NET event pattern.

The result of all that work was the delegate and event support in C# and .NET.

The remaining articles in this series will cover language features, library support, and common idioms used when you work with delegates and events. You'll learn about:

- The `delegate` keyword and what code it generates.
- The features in the `System.Delegate` class, and how those features are used.
- How to create type-safe delegates.
- How to create methods that can be invoked through delegates.
- How to work with delegates and events by using lambda expressions.
- How delegates become one of the building blocks for LINQ.
- How delegates are the basis for the .NET event pattern, and how they're different.

Web Services in C#

An Overview Let's think of a scenario where I am planning to show information on regional,national and international news, weather information, ongoing sports scores and other personalized content in a web site. Just think about how much effort and time it will take me to develop this application if I write the code for all these functionalities. On the other hand all these functionalities are already provided by other available sites. So, what if I can use this existing logic in my application? But, the question here is “how I can use someone else's business logic in my application?”.

For situations of this sort (& many other), we have techniques like **Web Services**.

With Web Services, you can reuse someone else's business logic instead of replicating it yourself, using just a few lines of code. This technique is similar to what programmers currently do with libraries of APIs, DLLs or plug-ins. The main difference is that Web Services can be located remotely on another server.

When HTML pages (or the HTML output generated by ASP.NET web forms) are rendered in a browser for the end user, Web Services are invoked by other applications. They are pieces of business logic that are hosted somewhere on the internet and can be accessed by other applications.

Note 1: Web Services are not limited to the .NET Framework. The standards were defined before .NET was released and they are exposed, used and supported by vendors other than Microsoft.
Note 2: Web Services are cross-platform; a service written in one language can be invoked by an application in some other language. The only requirement for accessing a service is an internet connection to make the HTTP request.

Since a web service is cross-platform, there should be some commonly understandable language for requesting a service and getting a response from the service. Such a standard common language is XML. That's why Web Services are built on XML-based standards for exchanging data.

As a result, the set of data types Web Services can use is limited to the set of data types recognized by the XML Schema standard. So you can use simple data types such as strings and numbers to communicate with a web service and you can't send proprietary .NET objects such as a FileStream, an Image or an EventLog. This restriction makes a lot of sense. Since other programming languages have no way to interpret these .NET objects, even if you could devise a way to send them over the wire, the client might not be able to interpret them, that would thwart interoperability.

Note 3: If you need to work with .NET proprietary objects, you can go for .NET remoting. It is a distributed technology that allows use of .NET objects. But non-.NET clients can't consume it.
Here's the list of supported data types

- **Built-in types (The Basics):** The built in C# data types such as short, int, long, ushort, uint, ulong, float, double, decimal, bool, string, char, byte and DateTime.
- **Objects:** You can use an object of any user defined class. If your class has methods, these methods will not be transmitted to the client.
- **Arrays:** Arrays of any supported type (built-in or custom objects). You can also use an ArrayList (that is simply converted into an array).
- **Enumerations:** Enums are supported. However, a web service uses the string name of the enumeration value, not the underlying integer.
- **XmlNode:** Objects based on System.Xml.XmlNode represent a portion of an XML document. You can use this to send arbitrary XML.
- **DataSet and DataTable :** DataSet and DataTable are allowed. But other ADO.NET data objects, such as DataColumns and DataRows, aren't supported.

Create a web service

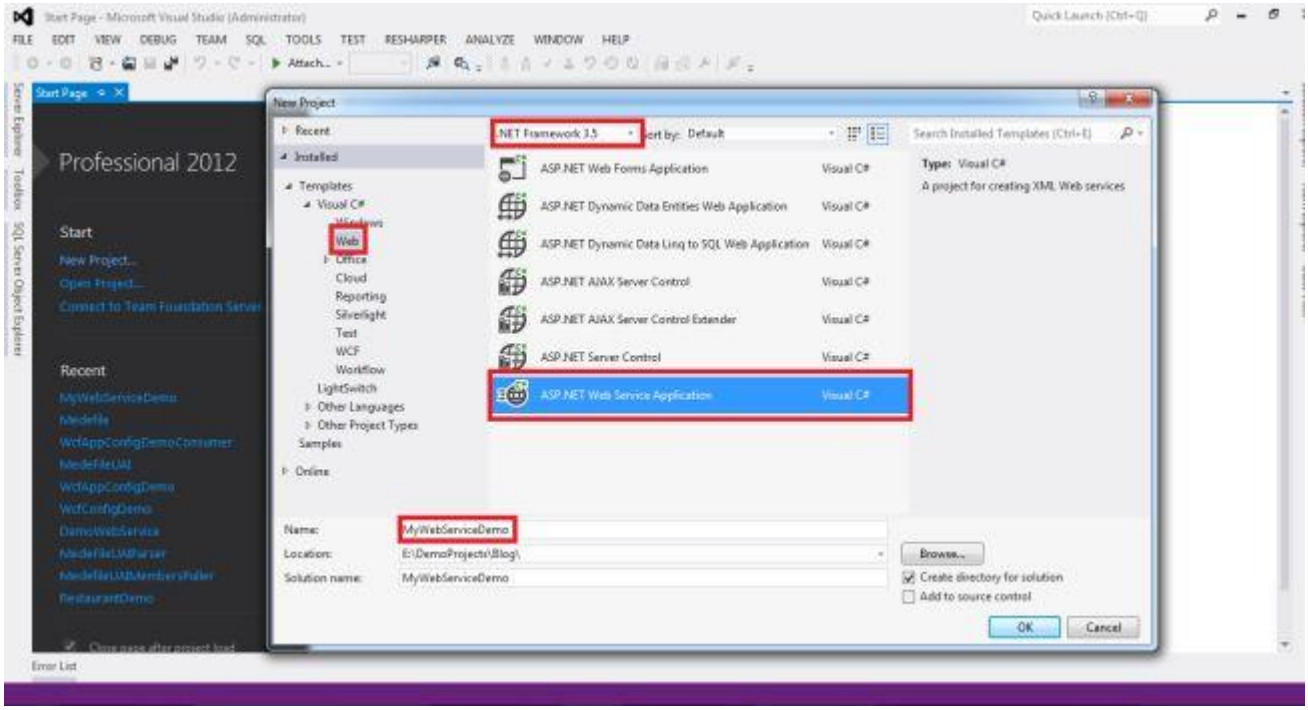
A web service is a simple asmx page.

Here I will use Visual Studio 2012 (though you can use any editor), with the .Net Framework 3.5 to create a web service.

Up to framework 3.5, Visual Studio provides a direct template for Web Services. Starting from 4, it doesn't provide any direct template, so you need to create a web application and add a web service to your application (right-click on your web application → Add → New Item → WebService).

Let's create a simple service that will return a sum of 2 integers.

Open Visual Studio in Administrator mode. File → New → Project then select .Net Framework 3.5 (on the top) then select ASP.NET web service application then name your project (I named it MyWebServiceDemo) then click OK.



Visual Studio will create a web service boilerplate (Service1.asmx and Service1.asmx.cs) for you. Now, let's analyze this template created by Visual Studio.

Note the following In Service1.asmx.cs:

1. An additional namespace "**System.Web.Services**" is included along with the 4 default namespaces that Visual Studio includes for web application.

2. The "Service1" class is inherited from "**System.Web.Services.WebService**". By inheriting a class from "**System.Web.Services.WebService**", you can access built-in ASP.NET objects such as (Application, Session, User, Context, server). If you don't need built-in objects of .Net, you don't need to inherit your service class from "**WebService**".

3. "Service1" is decorated with a "**WebService**(Namespace = "http://tempuri.org/")" attribute. If you want to expose a class as a service, you need to decorate it with the "WebService" attribute. This **WebService** attribute has several properties like:

    - **Namespace:** This property makes a service uniquely identifiable. This is an XML property. A client application may be consuming several services, so there is the probability of a naming collision. To avoid this, it's service providers responsibility to use a unique namespace.
    - **Name**: Use this property to provide a descriptive name to your service.
    - **Description:** To provide a brief description on the service.

4. "Service1" has another attribute "**WebServiceBinding**(ConformsTo = **WsiProfiles**.BasicProfile1_1)". This is to indicate the standard which service is following. If the service does not confirm to this standard, you will get an exception.

5. One more attribute service is decorated with is "[System.Web.Script.Services.**ScriptService**]", to make a service accessible from the client script, it should be decorated with this attribute.

6. The Service1 class has a method HelloWorld that is decorated with a "[WebMethod]" attribute. The methods of the service that are to be accessed by the client application should be decorated with this attribute. There may be some method that the service is using for some internal functionality, client applications don't need to access them. Don't decorate such methods with a WebMethod attribute. The WebMethod attribute also has Name and Description properties that you can use to provide a self describing name or description respectively.

Now let's see the mark up. Right-click on Service1.asmx in Solution Explorer then select view mark up. In Service1.asmx, you will see that there is only a WebService directive with some attributes, since a service will be invoked by some application not by any end user. So the asmx page has no mark up.

```
<%@ WebService Language="C#" CodeBehind="Service1.asmx.cs"
Class="MyWebServiceDemo.Service1"%>
```

ASP.NET (C#)

Copy

- The "**WebService**" directive, indicates this asmx page is a web service.
- The "**Language="C#"**", is to indicate language used for this service.
- The "**CodeBehind**" property is nothing to do with ASP.NET or web service, this is completely a Visual Studio property, that is used to map the asmx page with it's code behind page.

- The “**Class**” property holds fully qualified name of service class. This marks the entry point of service like main() in C programming language.

Now, run your application by hitting F5, http://localhost:56655/Service1.asmx will open in your browser (the port number may vary). You will find a link for Service Description, that will redirect to the WSDL document of the service, another link for HelloWorld (list for methods exposed by service) that will redirect to a page for testing this method.

Implementing a web service

Now let's implement the service. Rename the “Service1” file in Solution Explorer to something convenient like “MyService”. Change the class name from Service1 to MyService. Open the mark up (asmx) page.

As you can see here Visual Studio is unable to resolve “Service1” in the class property, since the class indicates a fully qualified name of the service and we renamed our Service1 class to MyService. So Visual Studio is unable to resolve it. So, now change the class property to “**MyWebServiceDemo.MyService**”. Change the “CodeBehind” property from “Service1.asmx.cs” to “MyService.asmx.cs” as we renamed the file also.

MyService.asmx

```
<%@ WebService Language="C#" CodeBehind="MyService.asmx.cs"
Class="MyWebServiceDemo.MyService"%>
```

ASP.NET (C#)

Copy

MyService.asmx.cs

```
using System.Web.Script.Serialization;
using System.Web.Services;

namespace MyWebServiceDemo
{
    // Use "Namespace" attribute with an unique name,to make service
uniquely discoverable
    [WebService(Namespace = "http://tempuri.org/")]
    // To indicate service confirms to "WsiProfiles.BasicProfile1_1"
standard,
    // if not, it will throw compile time error.
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    // To restrict this service from getting added as a custom tool
to toolbox
    [System.ComponentModel.ToolboxItem(false)]
    // To allow this Web Service to be called from script, using
ASP.NET AJAX
    [System.Web.Script.Services.ScriptService]
    public class MyService : WebService
    {
```

```
        [WebMethod]
        public int SumOfNums(int First, int Second)
        {
            return First + Second;
        }
    }
}
```
C#

Copy

Now, the service is ready to be used, let's compile and test it.

Test a web service

Let's run the project by hitting F5. The "**http://localhost:56655/MyService.asmx**" page will open that has a link for the Service description (the WSDL document, documentation for web service) another link for SumOfNums, which is for test page of SumOfNums method.

Let's use method overloading of the OOP concept. Add the following WebMethod in MyService class.

```
[WebMethod]
public float SumOfNums(float First, float Second)
{
    return First + Second;
}
```
C#

Copy

Hit F5 to run the application, you will get "Both Single SumOfNums(Single, Single) and Int32 SumOfNums(Int32, Int32) use the message name 'SumOfNums'. Use the MessageName property of the WebMethod custom attribute to specify unique message names for the methods." error message. We just used method overloading concept, so why this error message? This is because these methods are not unique for a client application. As the error message suggests let's use the MessageName property of the WebMethod attribute as shown below:

```
[WebMethod (MessageName = "SumOfFloats")]
public float SumOfNums(float First, float Second)
{
    return First + Second;
}
```
C#

Copy

Now, compile and run the application. Again it's showing some different error message "Service 'MyWebServiceDemo.MyService' does not conform to WS-I Basic Profile v1.1". As, **WsiProfiles**.BasicProfile1_1 doesn't support method overloading we are getting this exception. Now, either remove this

“[**WebServiceBinding**(ConformsTo = **WsiProfiles**.BasicProfile1_1)]” attribute or make it “[WebServiceBinding(ConformsTo = WsiProfiles.None)]”.

```
using System.Web.Script.Serialization;
using System.Web.Services;

namespace MyWebServiceDemo
{
    [WebService(Namespace = "http://tempuri.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.None)]
    [System.ComponentModel.ToolboxItem(false)]
    [System.Web.Script.Services.ScriptService]
    public class MyService : WebService
    {
        [WebMethod]
        public int SumOfNums(int First, int Second)
        {
            return First + Second;
        }
        [WebMethod(MessageName = "SumOfFloats")]
        public float SumOfNums(float First, float Second)
        {
            return First + Second;
        }
    }
}
```

C#

Copy

Now you can use method overloading in the service.

The Test page

Click on SumOfNums, you will be redirected to “**http://localhost:56655/MyService.asmx?op=SumOfNums**”. You will see here that just “?op=SumOfNums” is appended to the service URL. This page has 2 text boxes for 2 input values (First, Second) that the SumOfNums method takes as an input parameter and a button “Invoke”, clicking on which you'll be redirected to: “**http://localhost/WebServiceForBlog/MyService.asmx/SumOfNums**” that is has the value that the SumOfNums method returned in XML format. Similarly, by clicking on “SumOfNums MessageName="SumOfFloats"”, you will be redirected to “http://localhost:56655/MyService.asmx?op=**SumOfFloats**”. So the “SumOfNums MessageName="SumOfFloats"” method will be known as “SumOfFloats” for client applications.

.cs” as we renamed the file also.

MyService.asmx

```
<%@ WebService Language="C#" CodeBehind="MyService.asmx.cs"
Class="MyWebServiceDemo.MyService"%>
```

ASP.NET (C#)

Copy

MyService.asmx.cs

```
using System.Web.Script.Serialization;
using System.Web.Services;

namespace MyWebServiceDemo
{
    // Use "Namespace" attribute with an unique name,to make service
uniquely discoverable
    [WebService(Namespace = "http://tempuri.org/")]
    // To indicate service confirms to "WsiProfiles.BasicProfile1_1"
standard,
    // if not, it will throw compile time error.
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    // To restrict this service from getting added as a custom tool
to toolbox
    [System.ComponentModel.ToolboxItem(false)]
    // To allow this Web Service to be called from script, using
ASP.NET AJAX
    [System.Web.Script.Services.ScriptService]
    public class MyService : WebService
    {
        [WebMethod]
        public int SumOfNums(int First, int Second)
        {
            return First + Second;
        }
    }
}
```

C#

Copy

Now, the service is ready to be used, let's compile and test it.

Test a web service

Let's run the project by hitting F5. The "**http://localhost:56655/MyService.asmx**" page will open that has a link for the Service description (the WSDL document, documentation for web service) another link for SumOfNums, which is for test page of SumOfNums method.

Let's use method overloading of the OOP concept. Add the following WebMethod in MyService class.

```
[WebMethod]
public float SumOfNums(float First, float Second)
{
    return First + Second;
}
```

C#

Copy

Hit F5 to run the application, you will get “Both Single SumOfNums(Single, Single) and Int32 SumOfNums(Int32, Int32) use the message name 'SumOfNums'. Use the MessageName property of the WebMethod custom attribute to specify unique message names for the methods.” error message. We just used method overloading concept, so why this error message? This is because these methods are not unique for a client application. As the error message suggests let's use the MessageName property of the WebMethod attribute as shown below:

```
[WebMethod (MessageName = "SumOfFloats")]
public float SumOfNums(float First, float Second)
{
    return First + Second;
}
```

C#

Copy

Now, compile and run the application. Again it's showing some different error message “Service 'MyWebServiceDemo.MyService' does not conform to WS-I Basic Profile v1.1”. As, **WsiProfiles**.BasicProfile1_1 doesn't support method overloading we are getting this exception. Now, either remove this “[**WebServiceBinding**(ConformsTo = **WsiProfiles**.BasicProfile1_1)]” attribute or make it “[WebServiceBinding(ConformsTo = WsiProfiles.None)]”.

```
using System.Web.Script.Serialization;
using System.Web.Services;

namespace MyWebServiceDemo
{
    [WebService(Namespace = "http://tempuri.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.None)]
    [System.ComponentModel.ToolboxItem(false)]
    [System.Web.Script.Services.ScriptService]
    public class MyService : WebService
    {
        [WebMethod]
        public int SumOfNums(int First, int Second)
        {
            return First + Second;
        }
        [WebMethod(MessageName = "SumOfFloats")]
        public float SumOfNums(float First, float Second)
        {
            return First + Second;
        }
    }
}
```

C#

Copy

Now you can use method overloading in the service.

The Test page
Click on SumOfNums, you will be redirected to "**http://localhost:56655/MyService.asmx?op=SumOfNums**". You will see here that just "?op=SumOfNums" is appended to the service URL. This page has 2 text boxes for 2 input values (First, Second) that the SumOfNums method takes as an input parameter and a button "Invoke", clicking on which you'll be redirected to: "**http://localhost/WebServiceForBlog/MyService.asmx/SumOfNums**" that is has the value that the SumOfNums method returned in XML format. Similarly, by clicking on "SumOfNums MessageName="SumOfFloats"", you will be redirected to "http://localhost:56655/MyService.asmx?op=**SumOfFloats**". So the "SumOfNums MessageName="SumOfFloats"" method will be known as "SumOfFloats" for client applications.

Concept of COM in C#

Introduction

COM in C#

COM stands for "Common object modal**".**
COM IS A KIND OF SPECIFICATION TO MAKE THE COMPONENT REUSABLE. ALL THE ACTIVE COMPONENTS ARE BY DEFAULT COM COMPONENTS. COM IS A CONCEPT-ITS PRACTICAL IMPLEMENTATION IS ACTIVEX.

========= ABOVE CONCEPT WAS USED IN VB/VC++/C++==========

In .NET for this purpose we use assembly

Assembly is the component modal for .NET.

Assembly

1. It contains the meta data (data about data).
2. Project details: Name/version/author/creation date/creation time/ and IL (Intermediate language).

Assembly have two parts:

1. Manifest : Name, Author, Creation date, Time.
2. Body : IL
To view assembly: C:\>ILDASM A.EXE

Types of assembly

1. Private assembly:for same project
2. Public or shared assembly:for any project

Assembly comes in the form of

.DLL (Dynamic link library)

.EXE (Executable file)

Difference between .dll & .exe

DLL	**EXE**
1. Not self executed	1. Self executable
2. No Main()	2. Main()
3. In Process (Execute in client memory)	3. Out Process (Hold separate memory)
4. light weight	4. heavy weight

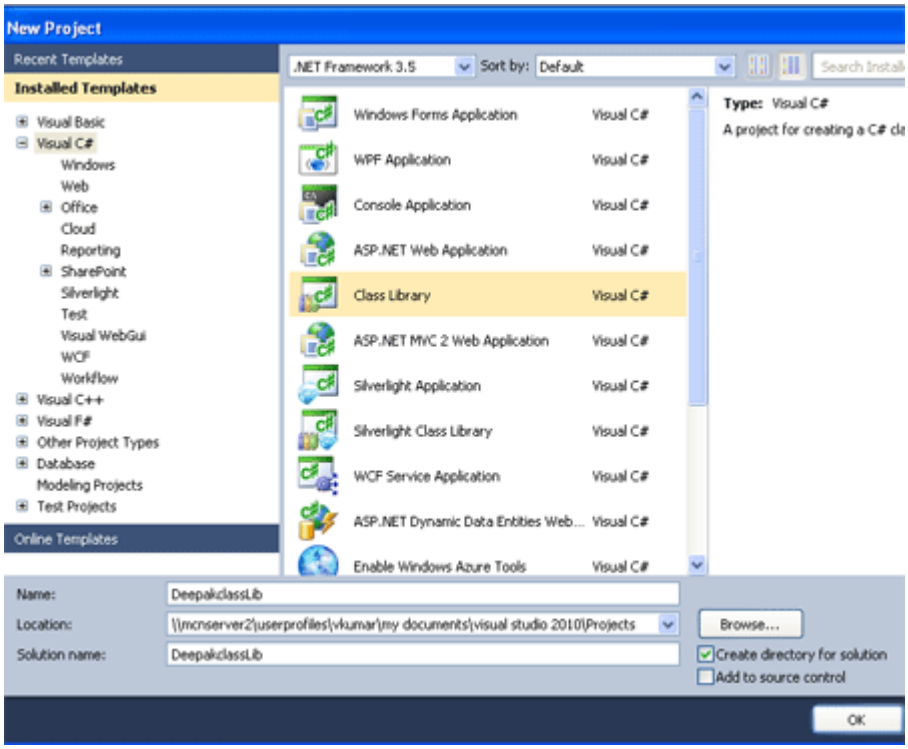
How to make a .dll :

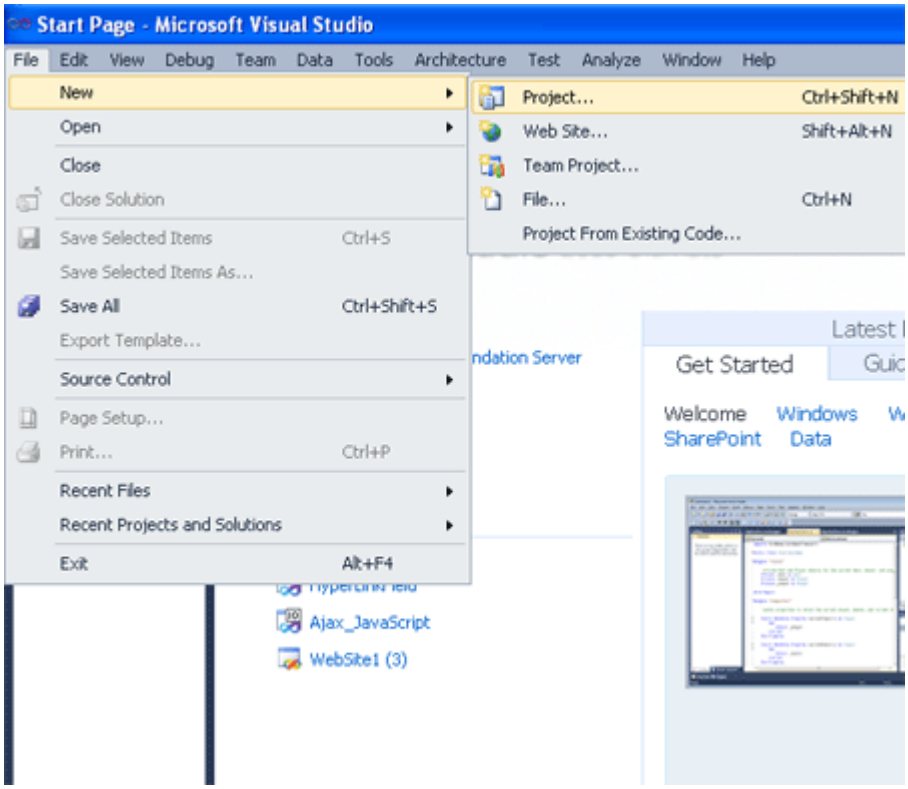
1.Class library : There will be Class(method/constructor/property)

2. Window control library : There will be user defined control.

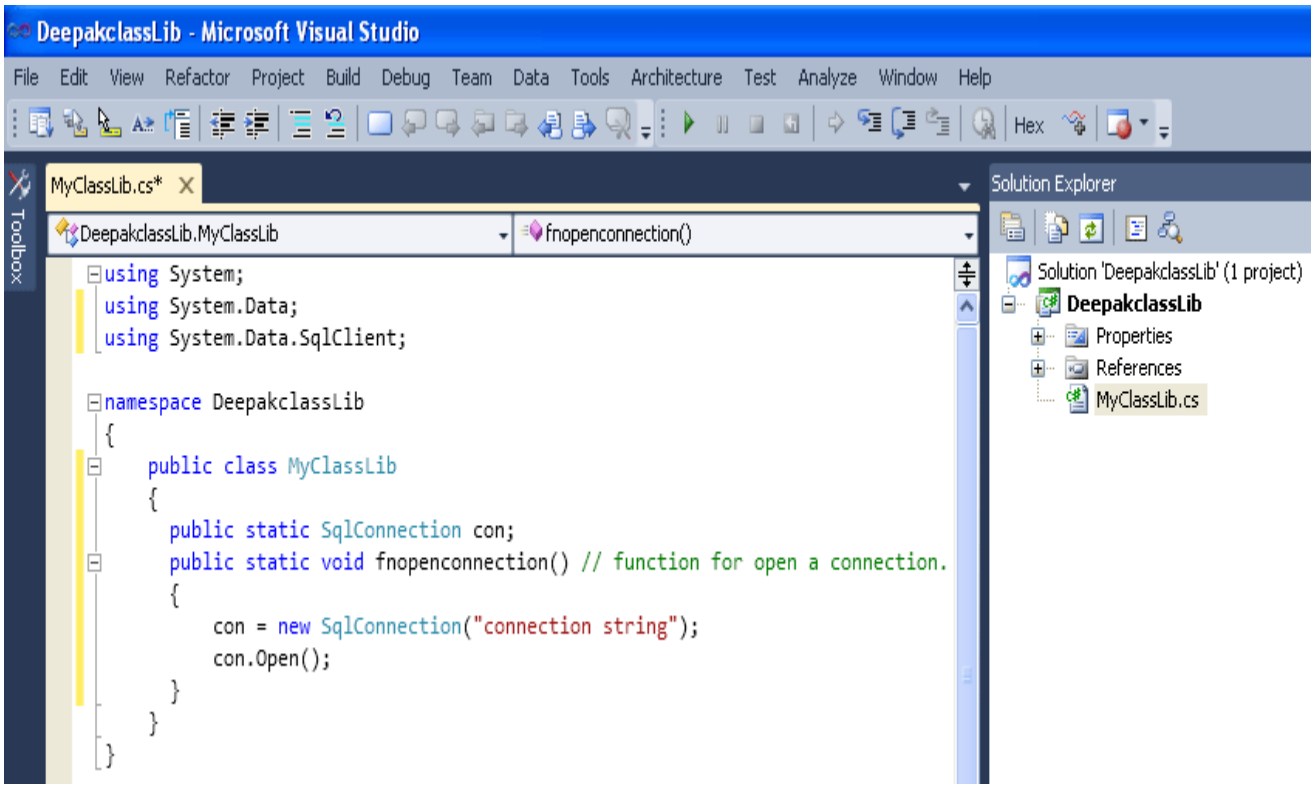
How to make a class library

1. Open visual studio-->File menu- select new project- select C# class library-name-location then OK.





2. Coding

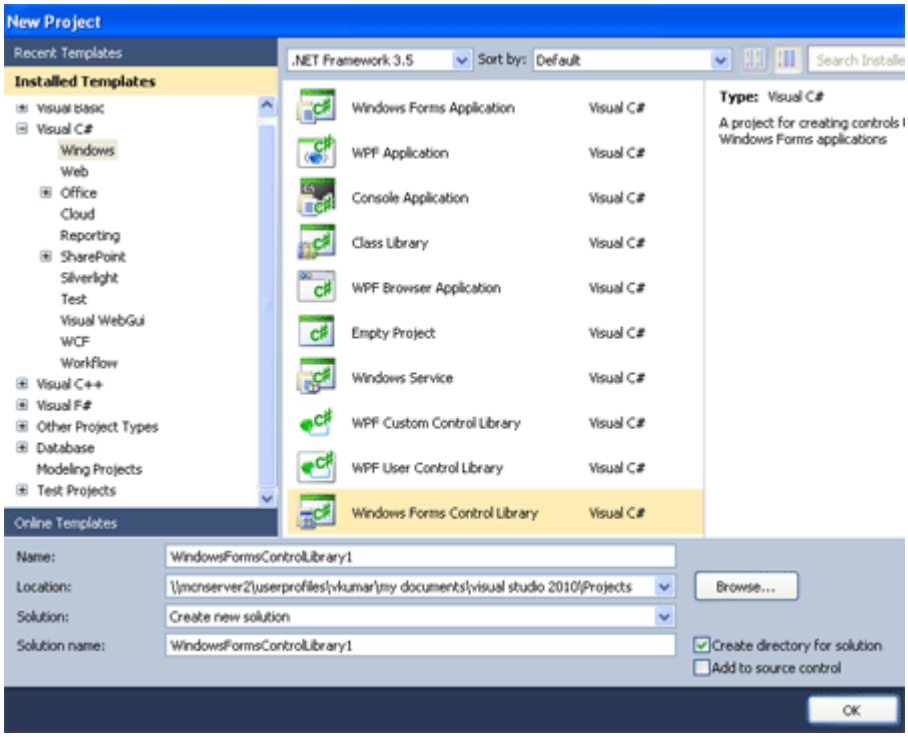


3. Build (After that .dll will create)

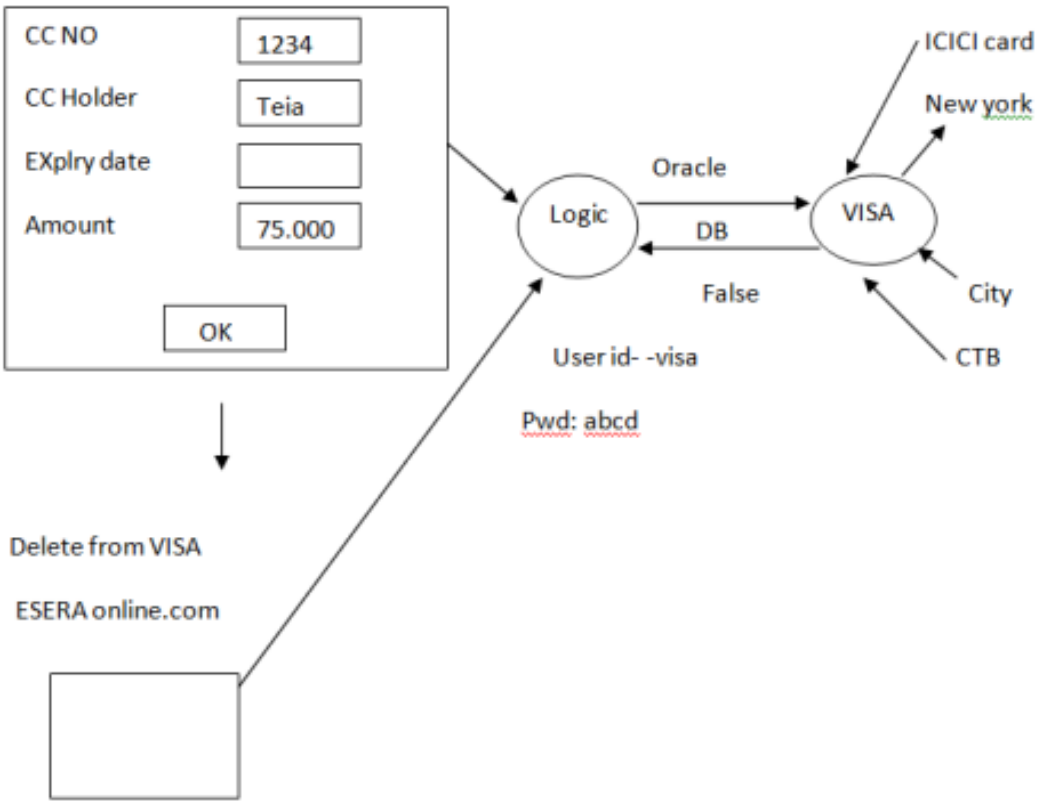
Another example

Windows control library

1. Open visual studio -File menu-new project -C#-windows form control library-name--ok



Distributed programming in C# .net



- Distributed programming is a concept of sharing application logic over the network A collection of computers, which are connected together is called as a network
- A collection of computers, which are connected together is called a network
- To establish a network, a protocol is required protocol provides a set of rules, which need to be allowed while transferring the data over the network Generally network can be established with the help of TCP or HTTP Protocol.

Generally distributed programming is required to develop3- Tier or N- Tier architectures.

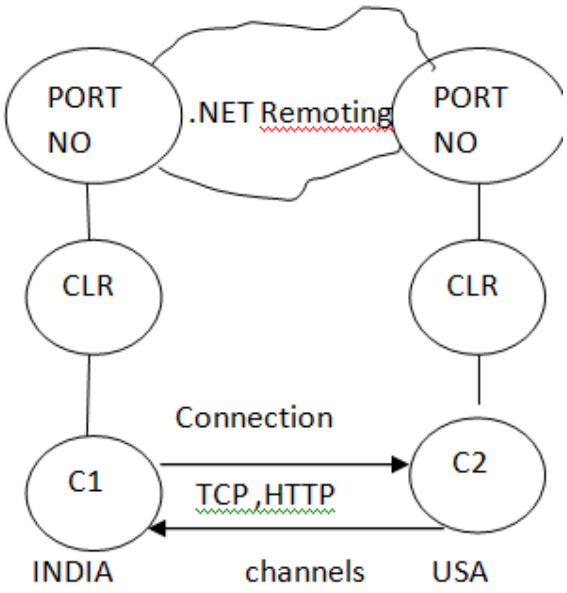
These core tutorials will help you to learn the fundamentals of .NET. For an in-depth understanding and practical experience, explore Online ".NET Training".

Examples for distributed programming techniques

1970	Unix & C	RPC
1990	JAVA	RMI
1995	VB 6.0	DCOM
1998	OMG	CORBA
	(Object management group)	
2001	.NET	Remoting

Working with .NET Remoting

1. .NET Remoting helps to develop distributed programming architecture.
2. Distributed programming allows to share the logic over the network.
3. Distributed programming allows to develop secured applications.



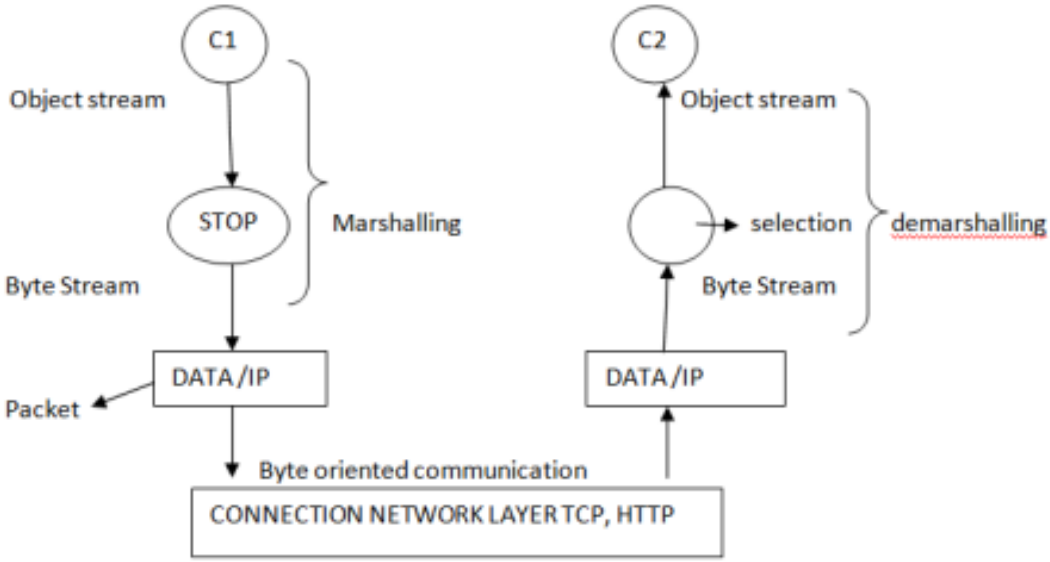
4. The connection between two computers is called a “channel”.
5. Remoting supports HTTP Channel and Tep channel
6. The communication in between two CLR’s is also called as .NET remoting
7. To start the communication a PORT need to be reserved.
8. The starting point of a communication channel is called as PORT.

9. Os supports o to 65535 ports, from which some ports are PRE-RESERVED.

Ex:-

HTTP---80 SMTP---25 FTP-----31 Web logic Servers-----7070,8080,9090

1. .NET Remoting Architecture



1. Converting object stream into byte stream is called as "Marshalling" and vice versa is called as "De- Marshalling".
2. Marshaling will be done with the help of STUB and De-Marshalling by the skeleton.
3. To work with .NET remoting, micro soft introduced system Run time. Remoting Assembly
4. Marshal By Ref object is a predefined class, which helps in Marshalling and De-Marshalling

In order share the Logic, Some Reouree files need to be copied from the server machine into the client machine.

Steps for Developing Remoting Application:-

Step1:- Write an interface with a set of abstract methods and compile into a DLL file.

Step2:- Write a server program by overriding Interface methods and Reserve a channel over a PORT.

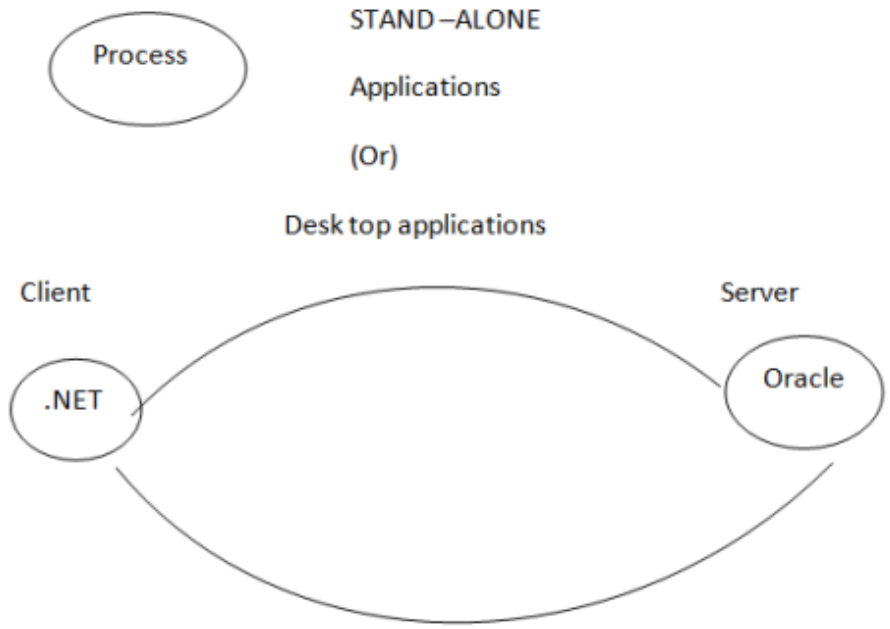
Step3:- Write a client program to create a connection with server and call interface methods.

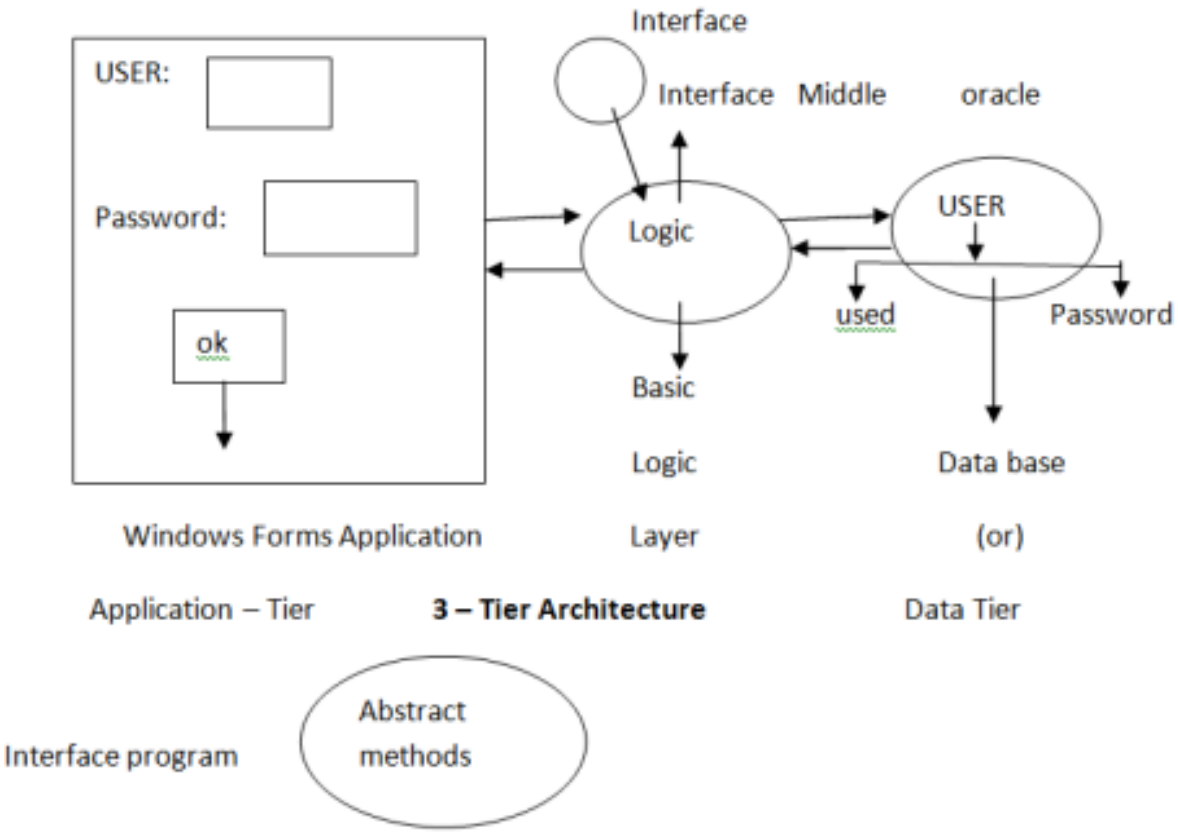
Step4:- Start server then start client programs

Obs:-

The interface DLL must be copied into both server and client machines.

Developing 3- Tier Architecture with Remoting:-





⟶Open oracle software and create a table with name users and with two columns used and pwd

⟶Create table users(Usid varchor2(20), Pwd varchar 2(10));

⟶Add two records as follows Insert into users values('teja', 'abc'); Insert into users values('Ravu', 'xyz');

⟶Save or commit;

1. **Developing Interface with a let of abstract methods**

⟶Open class Library project with the project name Middle Interface. Namespace middle Interface { Public interface MI { Bool check user(String a, String b); }//HI }Middle Inter face

⟶Build the project(build menu

⟶ Build solution)

Obs:- Middle interface.dll is created under D:/C194/ Middle interfaceBinDebug folder

2. Developing server Application

→Open console Application project with project name Middle server

→ project menuàadd referenceà System. Runtime. Remoting

→Project menuàAdd referenceàbrowseà Middle Interface.DLL

→Using system. Run time. Remoting;

→ Using system. Run time. Remoting. channels;

→Using system. Run time. Remoting channels. TCP;

→Using system. Data. oleDb;

→

```
Code in GD(Before Main)

{

Class M Server: Marsha L By Ret object, HI

{

Public bool check user(String a, string b)

{

String q=" select* from users where Usid='"+a+"' and pwd='"+"';

oleDb connection cn = new oleDb connection ("user id = scott; password=
tiger;provider=msdaora.1") Cn.open();

oleDb command cmd= new oleDb command(q.cn); object obj=cmd. Execute
scalor();

H(OBJ==NULL) Return false; Else Return true;

}
```

//check user method }//m server-class àcode for

```
main()

{
```

```
TCP channel t= new TCP channel(1600);

Channel Services. Register channel(t);

Remoting configuration. Register well known service type (type of (m
server),”abc”, well known object mode Single call);

Console. write line(“server is ready to use”); Console. Read key();

}
```

→Execute the project Then a TCP Server will be created with the following URL Tcp://Local host:1600/abc

3. Developing a client application

→open a new .NET Window(startàrunàdevenr)

→open WFAP with project name Middle client

→Project menuà Add ReferenceàBrowseà Middle Interface.DLL

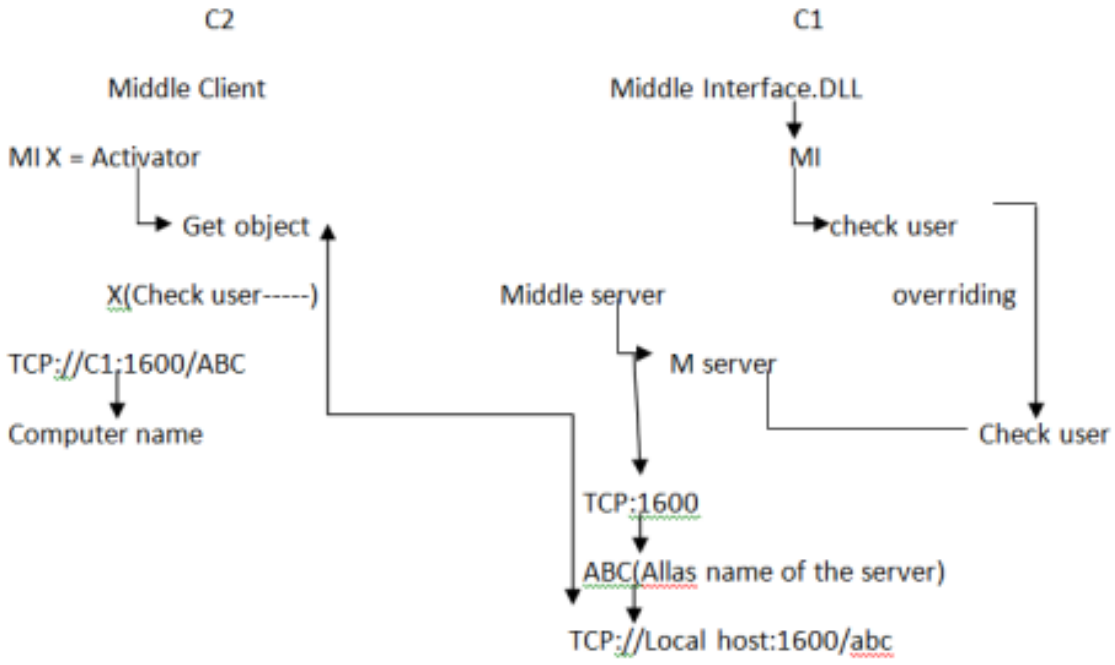
→Design the form as shown

→Using middle Interface;

→

code for button1-click

```
{

MI X=(MI)Activator. Get object(type of(MI)”tcp://Local host:1600/abc”);

BooL B= X. Check user (text box1.text,Text box2. text);

If(b==true) Message box. show(“valid user”);

Else Message box. show(“not a valid user”);

 }
```

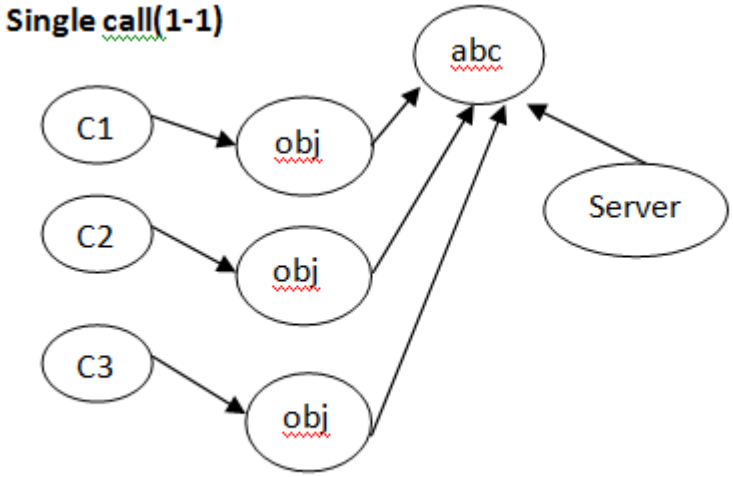


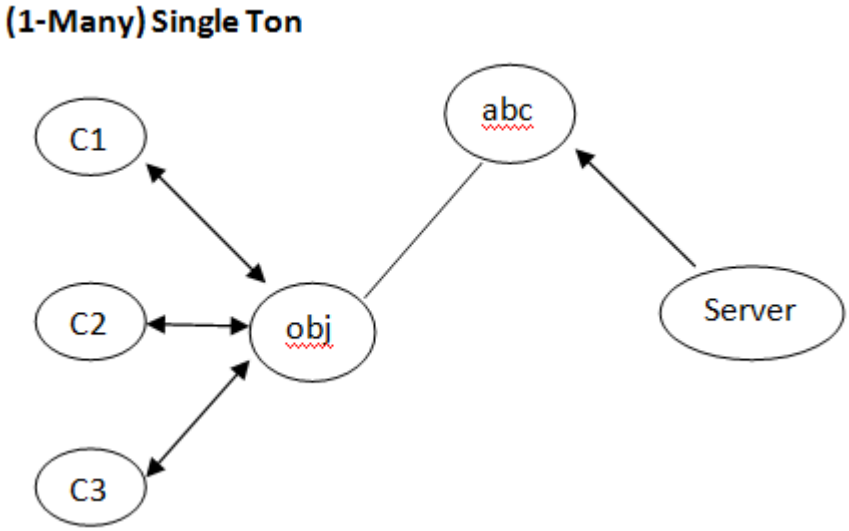
Obs:- Remoting configuration. Register well-known service type() takes3 arguments

Org 1) Type of the class, where interface methods are over rided .

Org 2) Provide an alias name to the server

Org3) Specifies the mode of the communication Remoting supports two types of communication model with the help of well-known object mode



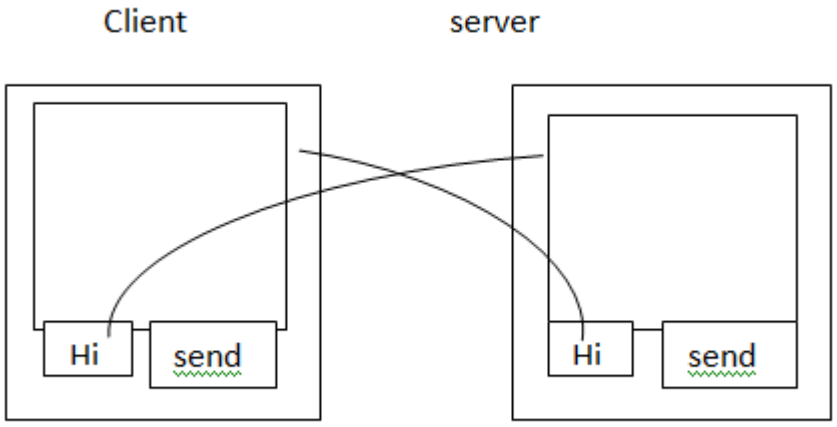


enum

→Single call is tastes in communication when compared with single ton

→In single call, a separate object will be created for every client

→In single ton, only one object will be created for all the clients When number of clients are limited then use single all other wise use single ton Activator class is used to open a connection with my specified URL 2 Developing an application like yahoo messenger with off line support and two consecutive messages quid not be same



⟶Open class library project with project name yahoo Interface

{

Public Interface YI

{

String Get TEXT();

}

}

⟶Build the project(Build menu⟶Build solution)

Obs:-

Yahoo Interface.dll is created under D:C194/Yahoo Interface/Bin/Debug folder

⟶Open windows farms Application project with project name yahoo server

⟶Design the form as follows:-

⟶Project

⟶Add Reference

→ System. Run time. Remoting

→project menu

→ Add Reference

→Browe

→ Yahoo Interface.DLL

Using system. Runtime. Remoting;

Using system. Runtime. Remoting. channels;

Using system. Runtime. Remoting. Channels.TCP;

Using Yahoo Interface;

→Code in GD

Static string S;

```
Class y server: Marshal By Ret Object.YI
{
Public string Get text()
{
Return Farm1.s;
}//Get Text
}//4 Server
```

→

```
Code for Button1-click(send)
{
Forml. s=text box2.text;
}
Code for farm1-load Event
{
TCP Channel=new TCP Channel(4829);
```

```
Channel Services. Register channel(t);

Remoting configuration. Register well known Service type(type of (4
Server),”yahoo”, well known object mode. single ton);

Text box1.text=”server is Ready”;

}
```

⟶Execute the project

OBS:-

Then a tcp server will be created with the following URL Tcp://Local host:4829/yahoo

⟶Developing client application

⟶Open a new .NET Window

⟶Open WFAP with project name yahoo client

⟶Place two text boxes, one button, and a timer control with enabled = false

⟶Project menu

⟶Add reference⟶Browse⟶Yahoo Interface.dll

⟶Using yahoo Interface; ⟶code in GD String s1,s2; YI X;

⟶code for the formal – Load event

```
{

X=(YI)Activator. Get Object(type of (yI) “tcp://Local host:4829/yahoo”);

Text box1.text=x. get text();

Timer1.Enabled=true;

}

àcode for time:-1-tick event

{

S1-X.Get Text();

If(s1 !=s2)
```

```
Text box1.text=text box1.text+ Environment.  new line+s1;

S2=s1;

}
```

Reading and Writing XML in C#

In this article, you will see how to read and write XML documents in Microsoft .NET using C# language.

First, I will discuss XML .NET Framework Library namespace and classes. Then, you will see how to read and write XML documents. In the end of this article, I will show you how to take advantage of ADO.NET and XML .NET model to read and write XML documents from relational databases and vice versa.

Introduction to Microsoft .NET XML Namespaces and Classes

Before start working with XML document in .NET Framework, it is important to know about .NET namespace and classes provided by .NET Runtime Library. .NET provides five namespace - System.Xml, System.Xml.Schema, System.Xml.Serialization, System.Xml.XPath, and System.Xml.Xsl to support XML classes.

The System.Xml namespace contains major XML classes. This namespace contains many classes to read and write XML documents. In this article, we are going to concentrate on reader and write class. These reader and writer classes are used to read and write XML documents. These classes are - XmlReader, XmlTextReader, XmlValidatingReader, XmlNodeReader, XmlWriter, and XmlTextWriter. As you can see there are four reader and two writer classes.

The XmlReader class is an abstract bases class and contains methods and properties to read a document. The Read method reads a node in the stream. Besides reading functionality, this class also contains methods to navigate through a document nodes. Some of these methods are MoveToAttribute, MoveToFirstAttribute, MoveToContent, MoveToFirstContent, MoveToElement and MoveToNextAttribute. ReadString, ReadInnerXml, ReadOuterXml, and ReadStartElement are more read methods. This class also has a method Skip to skip current node and move to next one. We'll see these methods in our sample example.

The XmlTextReader, XmlNodeReader and XmlValidatingReader classes are derived from XmlReader class. As their name explains, they are used to read text, node, and schemas.

The XmlWrite class contains functionality to write data to XML documents. This class provides many write method to write XML document items. This class is base class for XmlTextWriter class, which we'll be using in our sample example.

The **XmlNode** class plays an important role. Although, this class represents a single node of XML but that could be the root node of an XML document and could represent the entire file. This class is an abstract base class for many useful classes for inserting, removing, and replacing nodes, navigating through the document. It also contains properties to get a parent or child, name, last child, node type and more. Three major classes derived from XmlNode are XmlDocument, XmlDataDocument and XmlDocumentFragment. XmlDocument class represents an XML document and provides methods and properties to load and save a document. It also provides functionality to add XML items such as attributes, comments, spaces, elements, and new nodes. The Load and LoadXml methods can be used to load XML documents and Save method to save a document respectively. XmlDocumentFragment class represents a document fragment, which can be used to add to a document. The XmlDataDocument class provides methods and properties to work with ADO.NET data set objects.

In spite of above discussed classes, System.Xml namespace contains more classes. Few of them are XmlConvert, XmlLinkedNode, and XmlNodeList.

Next namespace in Xml series is System.Xml.Schema. It classes to work with XML schemas such XmlSchema, XmlSchemaAll, XmlSchemaXPath, XmlSchemaType.

The System.Xml.Serialization namespace contains classes that are used to serialize objects into XML format documents or streams.

The System.Xml.XPath Namespace contains XPath related classes to use XPath specifications. This namespace has following classes -XPathDocument, XPathExression, XPathNavigator, and XPathNodeIterator. With the help of XpathDocument, XpathNavigator provides a fast navigation through XML documents. This class contains many Move methods to move through a document.

The System.Xml.Xsl namespace contains classes to work with XSL/T transformations.

Reading XML Documents

In my sample application, I'm using books.xml to read and display its data through XmlTextReader. This file comes with VS.NET samples. You can search this on your machine and change the path of the file in the following line:

XmlTextReader textReader = new XmlTextReader("C:\\books.xml");

Or you can use any XML file.

The XmlTextReader, XmlNodeReader and XmlValidatingReader classes are derived from XmlReader class. Besides XmlReader methods and properties, these classes also contain members to read text, node, and schemas respectively. I am using XmlTextReader class to read an XML file. You read a file by passing file name as a parameter in constructor.

XmlTextReader textReader = new XmlTextReader("C:\\books.xml");

After creating an instance of XmlTextReader, you call Read method to start reading the document. After read method is called, you can read all information and data stored in a document. XmlReader class has properties such as Name, BaseURI, Depth, LineNumber and so on.

List 1 reads a document and displays a node information using these properties.

About Sample Example 1

In this sample example, I read an XML file using XmlTextReader and call Read method to read its node one by one until end of file and display the contents to the console output.

Sample Example 1.

```
using System;
using System.Xml;
namespace ReadXml1 {
    class Class1 {
        static void Main(string[] args) {
            // Create an isntance of XmlTextReader and call Read
method to read the file
            XmlTextReader textReader = new
XmlTextReader("C:\\books.xml");
            textReader.Read();
            // If the node has value
            while (textReader.Read()) {
                // Move to fist element
                textReader.MoveToElement();
                Console.WriteLine("XmlTextReader Properties Test");
                Console.WriteLine("==================");
                // Read this element's properties and display them
on console
                Console.WriteLine("Name:" + textReader.Name);
                Console.WriteLine("Base URI:" + textReader.BaseURI);
                Console.WriteLine("Local Name:" +
textReader.LocalName);
                Console.WriteLine("Attribute Count:" +
textReader.AttributeCount.ToString());
                Console.WriteLine("Depth:" +
textReader.Depth.ToString());
                Console.WriteLine("Line Number:" +
textReader.LineNumber.ToString());
                Console.WriteLine("Node Type:" +
textReader.NodeType.ToString());
                Console.WriteLine("Attribute Count:" +
textReader.Value.ToString());
            }
        }
    }
}
```

C#

Copy

The NodeType property of XmlTextReader is important when you want to know the content type of a document. The XmlNodeType enumeration has a member for each type of XML item such as Attribute, CDATA, Element, Comment, Document, DocumentType, Entity, ProcessInstruction, WhiteSpace and so on.

List 2 code sample reads an XML document, finds a node type and writes information at the end with how many node types a document has.

Highly Recommended

I have published a free book on XML programming using C#. Get your free copy here.

Learn more XML Programming

Still hungry for more XML programming with C# and .NET? Here is a dedicated section with hundreds of articles and code samples on XML programming using C# and .NET.

XML Programming in C#

About Sample Example 2

In this sample example, I read an XML file using XmlTextReader and call Read method to read its node one by one until end of the file. After reading a node, I check its NodeType property to find the node and write node contents to the console and keep track of number of particular type of nodes. In the end, I display total number of different types of nodes in the document.

SHORT QUESTIONS

1.Write about delegates and events?

2.Write about indexes?

3.Discuss about reflection?

4.Write about messaging?

5.Write about COM?

LONG QUESTIONS

1.Write about indexes and web services?

2.Discuss about Messaging and Reflection?

3.Discuss about localization?

4.Discuss about Distributed Application?

5.Discuss about Web services?

UNIT-4

Database Connection In C# Using ADO.NET

Background

C# console application is the simplest app to create to test our database connectivity. In this article, I'll create a console application, use ADO.NET SQL data provider classes to connect to a SQL Server database using C#, and access, update and execute SQL commands using ADO.NET.

You can use the same code in your Windows Forms or WPF application.

Working with Data in the .NET Framework

Database connectivity and functionality are defined in the ADO.NET namespaces of the .NET Framework. ADO.NET comes with several data providers including SqlClient, OleDB, and ODBC. .NET framework also provides in-memory data access using LINQ. In this article, we will use the SqlClient data provider of ADO.NET.

SQL data provider of ADO.NET is defined in the System.Data.SqlClient namespace. We'll also use the System.Data and System.Data.Sql namespace that provide additional database functionality.

If you're new to ADO.NET, here is a detailed article, What is ADO.NET.

System.Data.SqlClient

This assembly (namespace) of .NET Framework contains all of the classes required to connect to a SQL Server database and read, write, and update. The namespace provides classes to create a database connection, adapters, and SQL commands that provide the functionality to execute SQL queries.

The SqlError class is used for error and the success report returned after query execution.

In this article, we will work with the SQL Server database only.

Connecting to a database

Connection to a database requires a connection string. This string has the information about the server you're going to connect to, the database you will require, and the credentials that you can use to connect. Each database has its own properties including the server name and type.

SqlConnection class represents a database connection. The following code creates a SqlConnection object by passing a SQL Server connection string.

```
1. using(SqlConnection conn = new SqlConnection()) {
2.     conn.ConnectionString = "Server=[server_name];Database=[database_name
   ];Trusted_Connection=true";
3.     // using the code here...
4. }
```

In this connection string,

1. Server - Name of the server to connect to.
2. Database - This is the name of the database you're connecting to.

In all of the databases, there are two types of login methods. Windows Authentication and Database Authentication. In Windows Authentication, the database is authenticated using the user's credentials from Windows (the OS), and in Database Authentication you the username and word, in order to connect to the database.

In my case, the authentication was Windows, so I need to write the Trusted_Connection part inside the string. If you're using the database authentication then you will provide the username and word fields in the string.

Learn How to Generate Connection Strings using Visual Studio

Connection pools

Connecting to a database, as already said, is a long process of opening the connection, closing the connection, and so on. To repeat this process for every single user in the application is not a good approach and will slow down the processes of code execution. So, in program executions, many such connections would be opened and closed and again opened that are identical. These processes are time-consuming and are the opposite of a good UX.

In the .NET Framework, ADO.NET plays a part in this and minimizes the opening and closing process to make the program execution a bit faster by creating, what we call, a Connection Pool. This technique reduces the number of times the connection is opened, by saving the instance of the connection. For every new connection, it just looks for a connection already opened, and then if the connection exists, doesn't attempt to create a new connection, otherwise, it opens a new connection based on the connection string.

It must be remembered that only the connections with the same configuration can be pooled. Any connection with even a single dissimilarity would require a new pool for itself. Generally, it is based on the ConnectionString of the connection. You can learn how that would differ by changing the values in the connection string.

An example from the MSDN documentation would be like:

```
1. using(SqlConnection connection = new SqlConnection(
2.     "Integrated Security=SSPI;Initial Catalog=Northwind")) {
3.     connection.Open();
4.     // Pool A is created.
5. }
6. using(SqlConnection connection = new SqlConnection(
7.     "Integrated Security=SSPI;Initial Catalog=pubs")) {
8.     connection.Open();
9.     // Pool B is created because the connection strings differ.
10.}
11.using(SqlConnection connection = new SqlConnection(
12.    "Integrated Security=SSPI;Initial Catalog=Northwind")) {
13.    connection.Open();
14.    // The connection string matches pool A.
15.}
```

Learn more about Connection Pooling in ADO.NET.

Why use "using" in code

In C# there are some objects that use the resources of the system. That needs to be removed, closed, flushed and disposed of, and so on. In C# you can either write the code to create a new instance to the resource, use it, close it, flush it and dispose of it. Or on the other hand, you can simply just use this using statement block in which the object created is closed, flushed, and disposed of and the resources are then allowed to be used again by other processes. This ensures that the framework would take the best measures for each process.

We could have done it using the simple line to line code like:

```
1. SqlConnection conn = new SqlConnection();
2. conn.ConnectionString = "connection_string";
3. conn.Open();
```

```
4. // use the connection here
5.  conn.Close();
6. conn.Dipose();
7. // remember, there is no method to flush a connection.
```

Document about SqlConnection on MSDN

This was minimized as:

```
1. using(SqlConnection conn = new SqlConnection()) {
2.     conn.ConnectionString = "connection_string";
3.     conn.Open();
4.  // use the connection here
5. }
```

Once the code would step out of this block. The resources would be closed and disposed of on their own. The framework would take care of in the best way.

Executing the Commands

Once connected to the database, you can execute the set of commands that you're having and which would execute on the server (or the data provider) to execute the function you're trying to do, such as a query for data, insert the data, update records and so on and so forth.

SQL has the basic syntax for the commands and in my opinion, has the simple syntax of commands and nearly human-understandable commands in the programming world. In the namespace the class, SqlCommand does this job for us. An example of a command would be like:

```
1. SqlCommand command = new SqlCommand("SELECT * FROM TableName", conn);
```

Each and every command on the SQL Server would be executed like this. The first parameter is the command and the second one is the connection on which the command would execute. You can any command into it and the underlying code would convert it back to the command that would execute on the server where the data is present and then will return the result to you, whether an error or a success report.

Sometimes you might want to use the command of the INSERT INTO clause. To work that out, you will need to add parameters to the command, so that your database is safe from SQL Injections. Using parameters would reduce the chances of your database being attacked, by throwing an error if the user tries to add some commands (using the form) into the database server.

Parameterizing the data

Parameterizing the query is done using the SqlParameter ed into the command. For example, you might want to search for the records where the criteria match. You can denote that criteria, by ing the variable name into the query and then adding the value to it using the SqlParameter object. For instance, the following is your SqlCommand to be ed on to the server:

```
1. // Create the command
2. SqlCommand command = new SqlCommand("SELECT * FROM TableName WHERE FirstC
   olumn = @0", conn);
3. // Add the parameters.
4. command.Parameters.Add(new SqlParameter("0", 1));
```

In the preceding code, the variable added is 0 and the value to it is ed. You can use any variable but it must start with a @ sign. Once that has been done, you can then add the parameters to that name. In this case, the value 1 has been hardcoded and you can add a variable value here too.

Remember, the connection must be opened in order to run this code, you can use conn.Open() to open the connection if asked.

As explained in the code, I have used the parameter as a number (0) that can also be a name. For example, you can also write the code as:

```
1. // Create the command
2. SqlCommand command = new SqlCommand("SELECT * FROM TableName WHERE FirstC
   olumn = @firstColumnValue", conn);
3. // Add the parameters.
4. command.Parameters.Add(new SqlParameter("firstColumnValue", 1));
```

This way, it will be easier for you to keep them in mind. I am better at working with numbers and indexes like in an array so I used 0, you can use a name, a combination of alphanumeric characters, and so on. Just the name in the SqlParameter object and you'll be good to go!

Reading the data returned

In SQL you usually use the SELECT statement to get the data from the database to show, CodeProject would do so to show the recent articles from their database, Google would do so to index the results, and so on. But how to show those data results in the application using C#? That is the question here. Well, in the namespace we're talking about, there is the class SqlDataReader present for the SqlCommand that returns the Reader object for the data. You can use this to read through the data and for each of the columns provide the results on the screen.

The following code would get the results from the command once executed:

```
1. // Create a new SqlDataReader object and read data from the command.
2. using(SqlDataReader reader = command.ExecuteReader()) {
```

```
    while there is another record present
    while (reader.Read()) {
        // write the data on to the screen
        Console.WriteLine(String.Format("{0} \t | {1} \t | {2} \t | {3}",

            // call the objects from their index
            reader[0], reader[1], reader[2], reader[3]));
    }
}
```

This is the same code, it will execute, and once done executing it will let the framework handle the job and close the resources depending on the best method.

Adding data to the SQL Server

A similar method is implemented for adding the data to the database. Only the command would change and we know in the database, the INSERT INTO clause is used to add the data. So, the command would become:

```
SqlCommand insertCommand = new SqlCommand("INSERT INTO TableName (FirstCo
lumn, SecondColumn, ThirdColumn, ForthColumn) VALUES (@0, @1, @2, @3)", c
onn);
```

You can then use the SqlParameter objects to add the values to the parameters. This way, when the command is executed the data would be added to the table that you've specified.

Catching the errors from SQL Server

SQL Server generates the errors for you to catch and work on them. In the namespace we're working on there are two classes that work with the errors and exceptions thrown by SQL Server.

1. SqlError
2. SqlException

These are used to get the error details or to catch the exceptions in the code and print the data results respectively. If you're going to use a try-catch block you're more likely to use the SqlException thing in your code.

For this to work, we will a command that we know will throw an error.

```
SqlCommand errorCommand = new SqlCommand("SELECT * FROM someErrorColumn",
 conn);
```

Now we know that this is faulty, but this won't generate any error, untill we execute it. To do so, we will try to execute it like this:

```
1. errorCommand.ExecuteNonQuery();
```

Once this is executed, SQL Server would complain, saying there is no such table present. To catch it you can simply use the try-catch block with the SqlException in the catch block to be caught. For a working code, you can see the following code block in the live example of my article. That explains the usage of the try-catch block with the SqlException here.

Working example

In the article, there is an associated example for you to download if you want to work it out. You must use a SQL Server, database, and the relevant tables to ensure the program works. If the server name does not match the database name or the tables then the program won't run. There was no way for me to attach a database in the example. Since the databases require a SQL Server database that will be always available using a connection, I won't use this database again, so I have not provided the database connection string.

Database table

The database table in my system was like the following,

	FirstColumn	SecondColumn	ThirdColumn	ForthColumn
1	1	First Column!	2014-09-28	0
2	2	Second Column!	2014-09-28	1
3	3	Third Column!	2014-09-28	1
4	4	Forth Column!	2014-09-28	0
5	5	Fifth Column!	2014-09-28	0
6	6	Sixth Column!	2014-09-28	0
7	7	Seventh Column!	2014-09-28	0

SELECT with a WHERE

We can at the same time add a few more parameters to the SELECT query so that only the data we want would be extracted from the database. For example, if we add a WHERE clause to the SELECT query, the following result would be generated.

XML Integration with Relational Data and ADO.NET

The **XmlDataDocument** class is a derived class of the **XmlDocument**, and contains XML data. The advantage of the **XmlDataDocument** is that it provides a bridge between relational and hierarchical data. It is an **XmlDocument** that can be bound to a **DataSet** and both classes can synchronize changes made to data contained in the two classes. An **XmlDocument** that is bound to a **DataSet** allows XML to integrate with relational data, and you do not have to have your data represented as either XML or in a relational format. You can do both and not be constrained to a single representation of the data.

The benefits of having data available in two views are:

- The structured portion of an XML document can be mapped to a dataset, and be efficiently stored, indexed, and searched.
- Transformations, validation, and navigation can be done efficiently through a cursor model over the XML data that is stored relationally. At times, it can be done more efficiently against relational structures than if the XML is stored in an **XmlDocument** model.
- The **DataSet** can store a portion of the XML. That is, you can use **XPath** or **XslTransform** to store to a **DataSet** only those elements and attributes of interest. From there, changes can be made to the smaller, filtered subset of data, with the changes propagating to the larger data in the **XmlDataDocument**.

You can also run a transform over data that was loaded into the **DataSet** from SQL Server. Another option is to bind .NET Framework classes-style-managed WinForm and WebForm controls to a **DataSet** that was populated from an XML input stream.

In addition to supporting **XslTransform**, an **XmlDataDocument** exposes relational data to **XPath** queries and validation. Basically, all XML services are available over relational data, and relational facilities, such as control binding, codegen, and so on, are available over a structured projection of XML without compromising XML fidelity.

Because **XmlDataDocument** is inherited from an **XmlDocument**, it provides an implementation of the W3C DOM. The fact that the **XmlDataDocument** is associated with, and stores a subset of its data within, a **DataSet** does not restrict or alter its use as an **XmlDocument** in any way. Code written to consume an **XmlDocument** works unaltered against an **XmlDataDocument**.
The **DataSet** provides the relational view of the same data by defining tables, columns, relations, and constraints, and is a stand-alone, in-memory user data store.

The following illustration shows the different associations that XML data has with the **DataSet** and **XmlDataDocument**:

The illustration shows that XML data can be loaded directly into a **DataSet**, which allows direct manipulation with XML in the relational manner. Or, the XML can be loaded into a derived class of the DOM, which is the **XmlDataDocument**, and subsequently loaded and synchronized with the **DataSet**. Because the **DataSet** and **XmlDataDocument** are synchronized over a single set of data, changes made to the data in one store are reflected in the other store.

The **XmlDataDocument** inherits all the editing and navigational features from the **XmlDocument**. There are times when using the **XmlDataDocument** and its inherited features, synchronized with a **DataSet**, is a more appropriate option than loading XML directly into the **DataSet**. The following table shows the items to be considered when choosing which method to use to load the **DataSet**.

When to load XML directly into a DataSet	When to synchronize an XmlDataDocument with a DataSet
Queries of data in the **DataSet** are easier using SQL than XPath.	XPath queries are needed over data in the **DataSet**.
Preservation of element ordering in the source XML is not critical.	Preservation of element ordering in the source XML is critical.
White space between elements and formatting does not need to be preserved in the source XML.	White space and formatting preservation in the source XML is critical.

If loading and writing XML directly into and out of a **DataSet** addresses your needs, see Loading a DataSet from XML and Writing a DataSet as XML Data.

If loading the **DataSet** from an **XmlDataDocument** addresses your needs, see Synchronizing a Dataset with an XML Document.

Using XML in a DataSet

With ADO.NET you can fill a DataSet from an XML stream or document. You can use the XML stream or document to supply to the DataSet either data, schema information, or both. The information supplied from the XML stream or document can be combined with existing data or schema information already present in the DataSet.

ADO.NET also allows you to create an XML representation of a DataSet, with or without its schema, in order to transport the DataSet across HTTP for use by another application or XML-enabled platform. In an XML representation of a DataSet, the data is written in XML and the schema, if it is included inline in the representation, is

written using the XML Schema definition language (XSD). XML and XML Schema provide a convenient format for transferring the contents of a DataSet to and from remote clients.

In This Section

DiffGrams
Provides details on the DiffGram, an XML format used to read and write the contents of a DataSet.

Loading a DataSet from XML
Discusses different options to consider when loading the contents of a DataSet from an XML document.

Writing DataSet Contents as XML Data
Discusses how to generate the contents of a DataSet as XML data, and the different XML format options you can use.

Loading DataSet Schema Information from XML
Discusses the DataSet methods used to load the schema of a DataSet from XML.

Writing DataSet Schema Information as XSD
Discusses the uses for an XML Schema and how to generate one from a DataSet.

DataSet and XmlDataDocument Synchronization
Discusses the capability available in the .NET Framework of synchronous access to both relational and hierarchical views of a single set of data, and shows how to create a synchronous relationship between a DataSet and an XmlDataDocument.

Nesting DataRelations
Discusses the importance of nested DataRelation objects when representing the contents of a DataSet as XML data, and describes how to create them.

Deriving DataSet Relational Structure from XML Schema (XSD)
Describes the relational structure, or schema, of a DataSet that is created from XML Schema.

Inferring DataSet Relational Structure from XML
Describes the resulting relational structure, or schema, of a DataSet that is created when inferred from XML elements.

Related Sections

ADO.NET Overview
Describes the ADO.NET architecture and components, and how to use them to access existing data sources as well as to manage application data.

See also

- DataSets, DataTables, and DataViews
- ADO.NET Overview

Recommended content

- **Read XML data into a dataset - Visual Studio (Windows)**

  Read XML data into a dataset. In this walkthrough, you create a Windows application that loads XML data into a dataset.

- **Writing DataSet Contents as XML Data - ADO.NET**

  Learn more about: Writing DataSet Contents as XML Data

- **Save a dataset as XML - Visual Studio (Windows)**

  Save a dataset as XML. Access the XML data in a dataset by calling the available XML methods on the dataset, such as GetXml or WriteXml.

- **Loading a DataSet from XML - ADO.NET**

  Learn how to add contents to an ADO.NET DataSet from XML. The .NET Framework offers flexibility for what to load and the structure of the DataSet.
  Show more

Pretty much any programming language can be used to create versatile and highly useful image processing libraries that can and are used to edit, change, distort, and process images. C# is no exception to this rule. C# is actually a great language to use for image processing because it's so powerful.

With these image processors, you can change and manipulate colors, image orientations, add the filter and blend effects, change image size, and much more. If you're looking to use an image processor built using C#, take a look at the list below to see what options are available to you:

1. ImageProcessor

ImageProcessor is a collection of lightweight image processing libraries built using C#. ImageProcessor gives the user the ability to edit and manipulate images quickly and easily on both desktop and mobile applications. The project is open source and completely free to download, install, and use.

2. CSharp Image Library

irely in C#. It supports jpg, png, BMP, TGA, and gif image formats. It's speedy, lightweight, and has some interesting color-related and color-splitting features.

3. Emgu CV

Emgu CV is an open source library that uses the OpenCV library as a basis for its image processing features. It works by allowing OpenCV core functions and functionalities to be called by any .NET language (including C#). It's written in C# and is highly portable — it's supported on Windows, Mac, and Linux operating systems.

Introduction to Azure and .NET

Azure is a cloud platform designed to simplify the process of building modern applications. Whether you choose to host your applications entirely in Azure or extend your on-premises applications with Azure services, Azure helps you create applications that are scalable, reliable, and maintainable. With extensive support in tools you already use like Visual Studio and Visual Studio Code and a comprehensive SDK library, Azure is designed to make you, the .NET developer, productive right from the start.

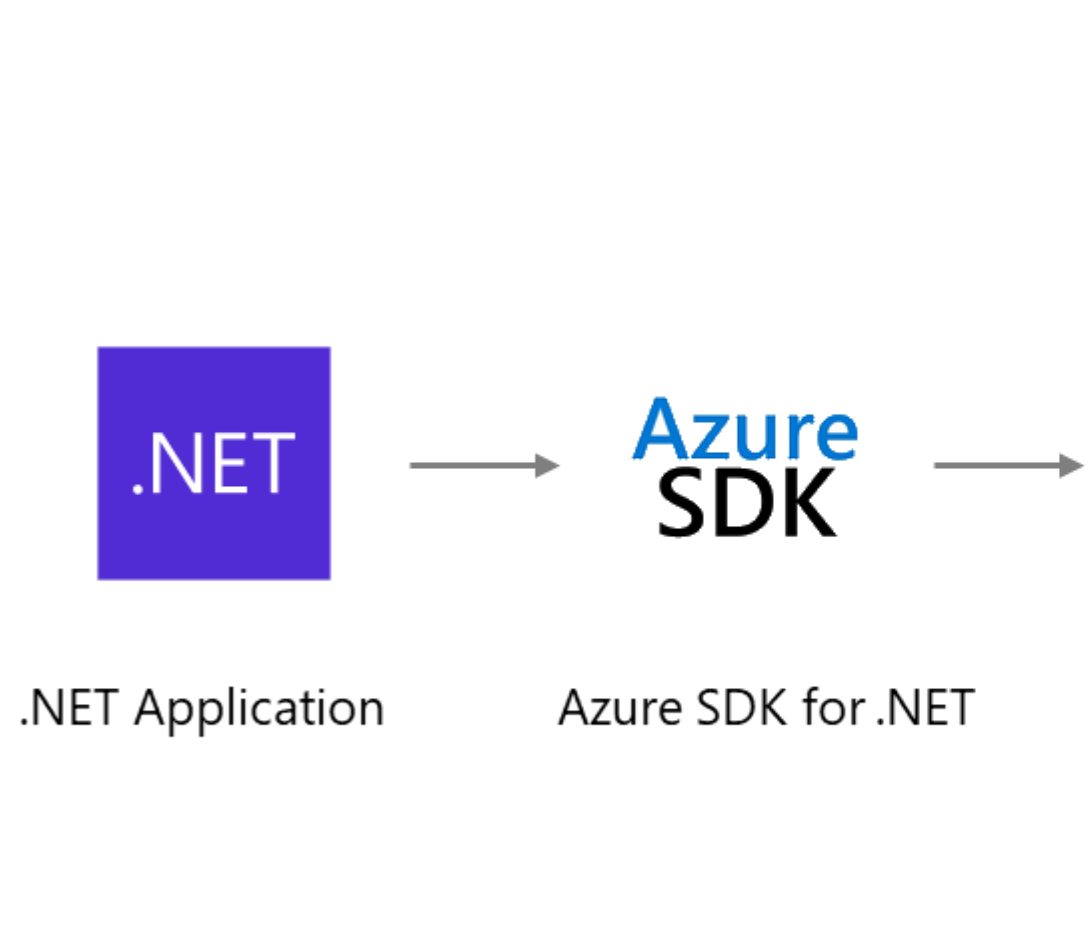
Application development scenarios on Azure

You can incorporate Azure into your application in different ways depending on your needs.

- **Application hosting on Azure -** Azure can host your entire application stack from web applications and APIs to databases to storage services. Azure supports a variety of hosting models from fully managed services to containers to virtual machines. When using fully managed Azure services, your applications can take advantage of the scalability, high-availability, and security built in to Azure.
- **Consuming cloud services from applications -** Existing apps can incorporate Azure services to extend their capabilities. This could include adding full-text searching capability with Azure Cognitive Search, securely storing application secrets in Azure Key Vault or adding vision, speech and language understanding capabilities with Azure Cognitive Services. These services are fully managed by Azure and can be easily added to your application without changing your current application architecture or deployment model.
- **Modern serverless architectures -** Azure Functions simplify building solutions to handle event-driven workflows, whether responding to HTTP requests, handling file uploads in Blob storage, or processing events in a queue. You write only the code necessary to handle your event without worrying about servers or framework code. Further, you can take advantage of over 250 connectors to other Azure and third-party services to tackle your toughest integration problems.

Access Azure services from .NET applications

Whether your app is hosted in Azure or on-premises, access to most Azure services is provided through the **Azure SDK for .NET**. The Azure SDK for .NET is provided as a series of NuGet packages and can be used in both .NET Core (2.1 and higher) and .NET Framework (4.6.1 and higher) applications. The Azure SDK for .NET makes incorporating Azure services into your application as easy as installing the correct NuGet package, instantiating a client object and calling the appropriate methods. More information on the Azure SDK for .NET can be found in the Azure SDK for .NET Overview.



Next steps

Next, learn about the most commonly used Azure services for .NET development.

Recommended content

- **Key Azure Services for .NET developers**

  Azure has over 100 services, but this article focuses on the ~8 or so services used by .NET developers most frequently

- **.NET Development on Azure Configuration Checklist**

  Provides a quick summary of all the tools you should have installed to do .net development with Azure

- **Configure Visual Studio for Azure Development with .NET**

This article helps you configure Visual Studio for Azure development including getting the right workloads installed and connecting Visual Studio to your Azure account.

- **Azure SDK for .NET Overview**

Provides an overview of what the Azure SDK for .NET is and the basic steps to use the SDK in a .NET application
Show more

SHORT QUESTIONS

1.write a short note on ado.net?

2.Define database connectivity?

3.Explain about web applications?

4.Explain image processing?

5.Define graphics?

LONG QUESTIONS

1.write detail about database connectivity with ADO.NET?

2.Define graphics and printing?

3.Discuss about XML and ADO.NET?

4.Explain c# libraries for image processing?

5.Define .NET applications to azure platform?

UNIT-5

Creating Web Sites with ASP.NET 2.0

“Web application development has come a long way in a fairly short period of time.” A quote like that surely won’t send anyone into shock anytime soon because it’s accepted as fact.

From basic, static HTML pages to totally data-driven and data-centric Web applications, the demands on a Web developer are much more complex and demanding than they were just a few years ago. The advent of social networking sites like MySpace, which is written in ASP.NET 2.0, interactive mapping sites, and sites streaming full motion video has required the Web developer to adapt and change with the times. One of the best tools to use to build these types of Web applications is Microsoft’s ASP.NET 2.0. In this article I am going to delve into some of the more interesting features of ASP.NET 2.0 and show you how you can begin using ASP.NET 2.0 on your next Web project.

ASP.NET 2.0 offers features that every ASP.NET developer will find useful, including:

- Multiple project types
- A wide variety of Web controls
- Project folders to help organize files
- Enhanced code-behind model
- Master pages and themes
- Site navigation features
- Enhanced data access features
- Membership and personalization
- Portal architecture including Web parts
- Site administration tools

In this article I’ll take a detailed look at a number of these features including Visual Studio 2005 enhancements, the code-behind model, the Web controls, the master pages feature, the themes and skins feature, and the site navigation features.

Creating a Web Site

With the release of Visual Studio 2005 SP1, Microsoft offers Web site developers a choice between two different project types: the Web Site and the Web Application. The Web Site project really is not a project at all since Visual Studio 2005 solution and project files are not created. In the Web Site model all pages and code are stored in a directory, much like the

traditional ASP model. When it comes to deployment, the easiest approach is to copy all of your files to your Web server and everything compiles on demand. You could also use the aspnet_compiler.exe utility to precompile your site into a binary file and deploy to your Web site.

The Web Application project is an implementation of the type of Web project available in Visual Studio 2003 and consists of a solution (.sln) and project files (.vbproj). It is the result of Microsoft listening to the ASP.NET 2.0 developer community and responding to the needs of the community. It provides the benefit of generating a single assembly in the local /bin folder ready for deployment. This model also makes it easier to incrementally deploy changes made to your site. If you are converting a Web application from Visual Studio 2003, this is the project model you are going to want to adopt. I use this model with most of the Web applications I develop.

To create a Web site with the Web Application project model, launch Visual Studio 2005 and from the File menu option choose **New Project…** and the New Project dialog box will appear **(Figure 1)**. Choose **ASP.NET Web Application**, the name of the project, the location, and the name of the solution if you select to have one created or specify if you want the new project added to the current solution. Click **OK** and Visual Studio will create the solution (.sln), the project (.vbproj since I selected Visual Basic), a default.aspx page, and a Web.config file.

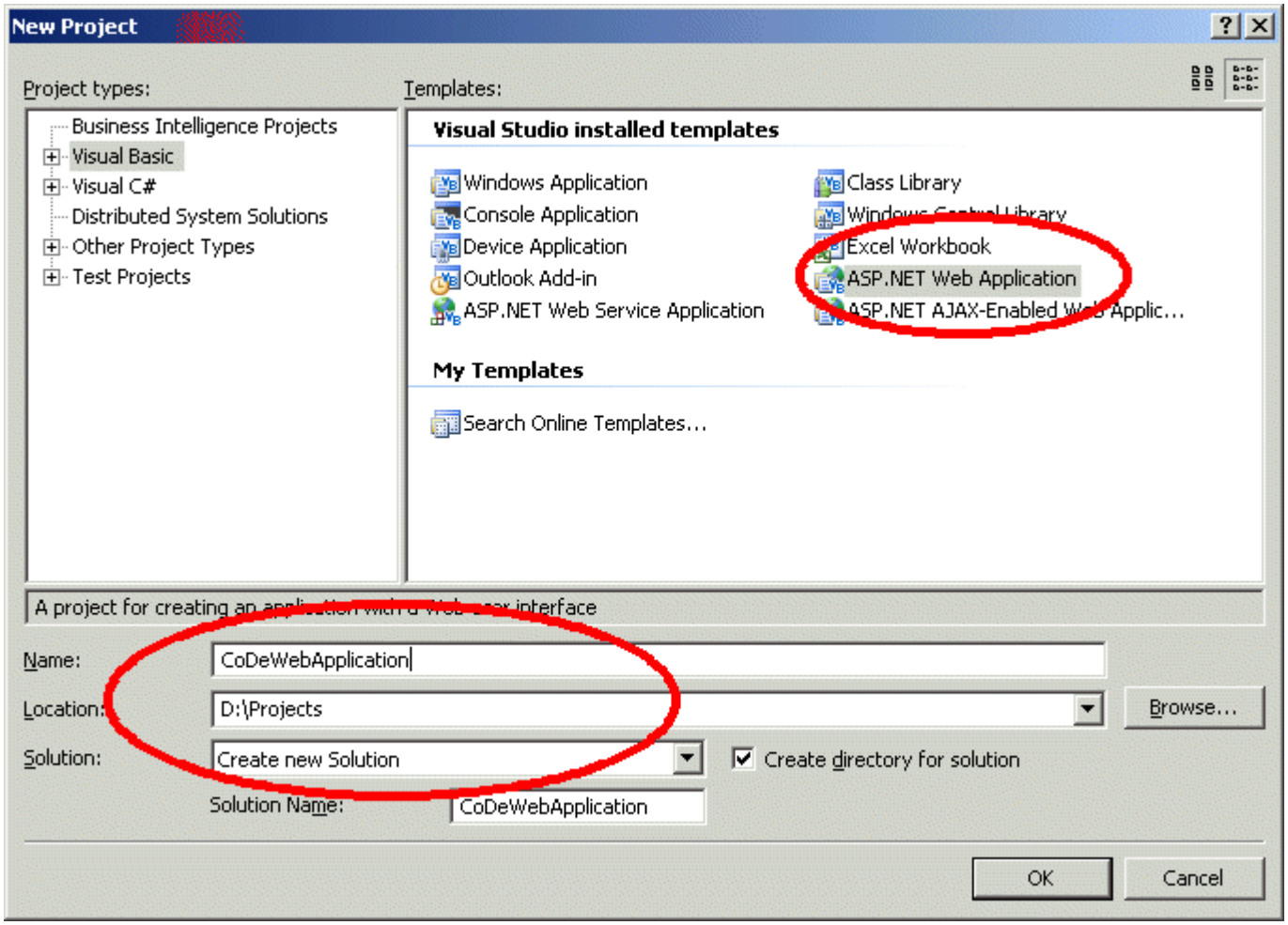


Figure 1: The New Project dialog box specifies the name, location, and language for your Web Application.

To create a Web site with the Web Site project model, launch Visual Studio 2005 and from the **File** menu option choose **New Web Site** and the New Web Site dialog box will appear **(Figure 2)**. Choose **ASP.NET Web Site**, the location of the Web site, the language used, and lastly, the name of the Web site.

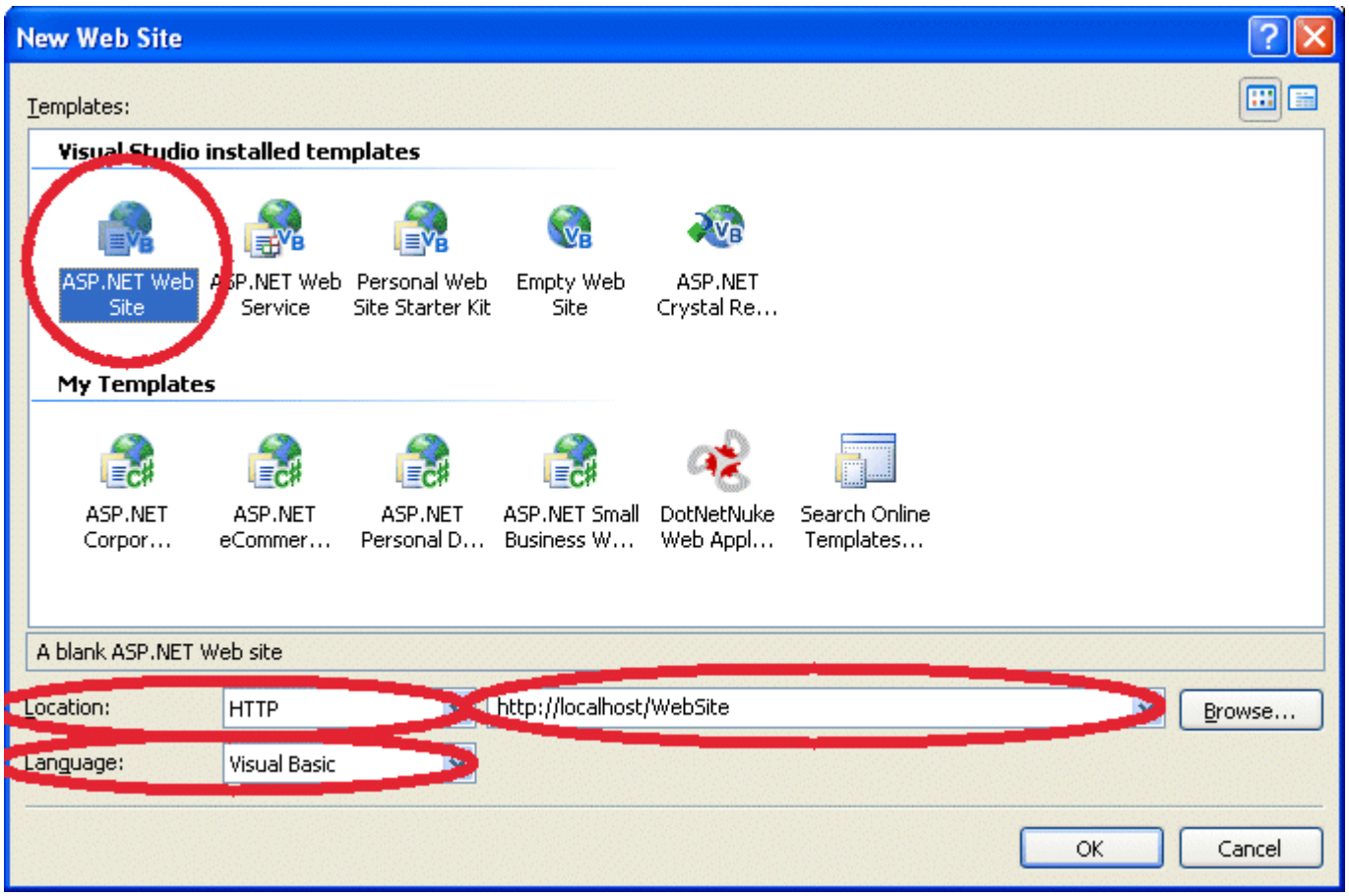


Figure 2: The New Web Site dialog box specifies the name, location, and language for your Web site.

I don't need to explain the name of the Web site and the language you want to use but I'll discuss other options. The location dropdown offers three options: File System, HTTP, and FTP. Creating a File System Web site is the simplest approach because your Web site can reside in any directory on your hard drive or shared network drive. The advantage of a File System Web site is that you can run it using Visual Studio 2005's built in development Web server. You don't need IIS though you can use IIS to run a File System Web site by creating a virtual directory for the directory you are storing the Web site in.

The second option, choosing HTTP as the location, will direct Visual Studio 2005 to automatically create a Web site in an IIS virtual directory. This saves you the step of creating the virtual directory in IIS but obviously you need to have access to an IIS server.

Choosing FTP as the location allows you to actually store and even code your application on a remote Web server somewhere. If you've selected this option, click the Browse button to display the Choose Location dialog box and then select a directory, an IIS virtual directory, and then enter the FTP parameters required to log on to an FTP location **(Figure 3)**.

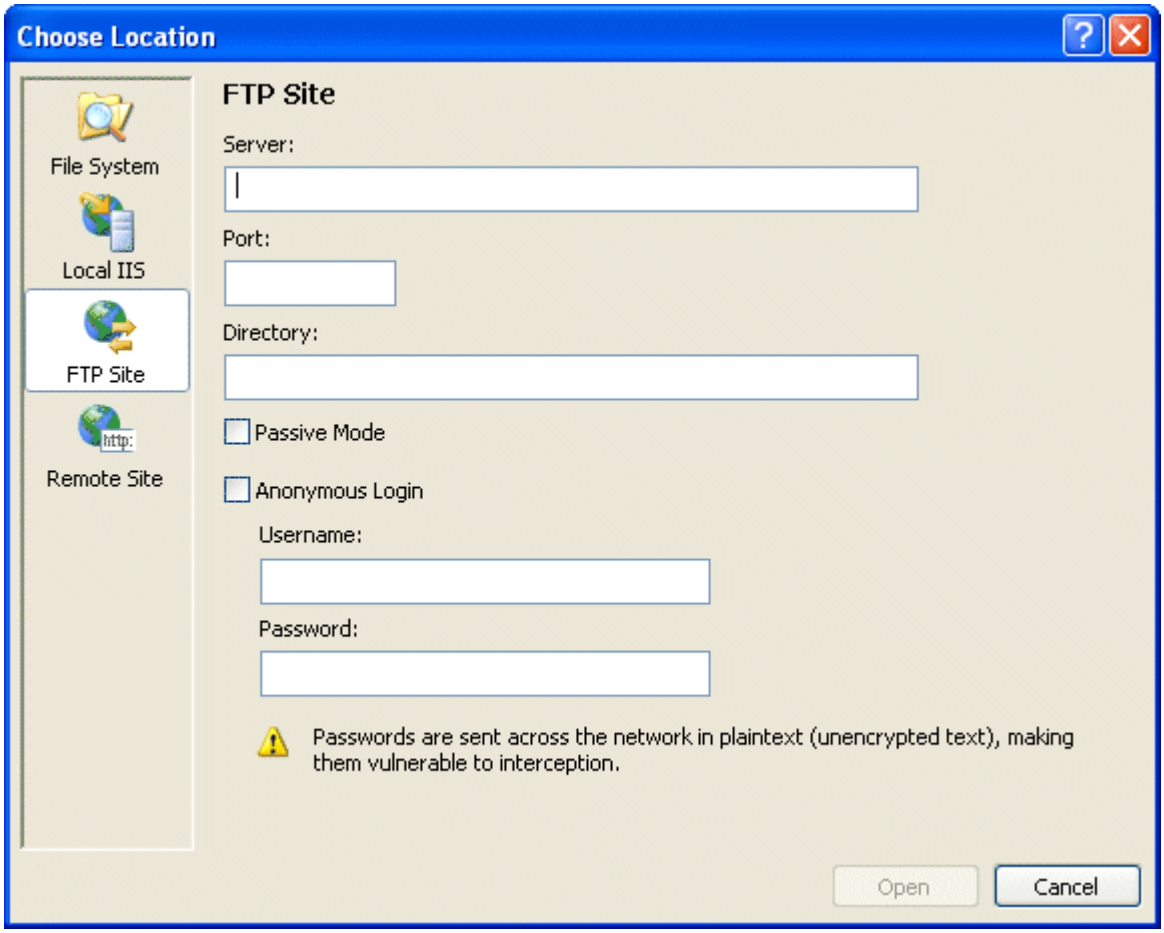


Figure 3: Site location can be FTP folder on a distant server.

Once you have decided on the name, location, and language for your new Web site, click **OK** to create the site. Visual Studio will create a Default.aspx page, Web.config file, and an App_Data folder. One primary difference between the Web Site project model and the Web Application project model is that in the Web Site model settings are stored in a Web.config file as opposed to project file.

Controls, Partial Classes, and Smart Tags

To add controls to a page you can drag and drop them from the Toolbox (Design mode) or you can manually edit the HTML (Source mode). The Design and Source tabs along the bottom of the page designer are used to jump back and forth between the two modes. The ASP.NET 2.0 Toolbox has new controls since ASP.NET 1.0/1.1 and existing controls have been better categorized **(Figure 4)**.

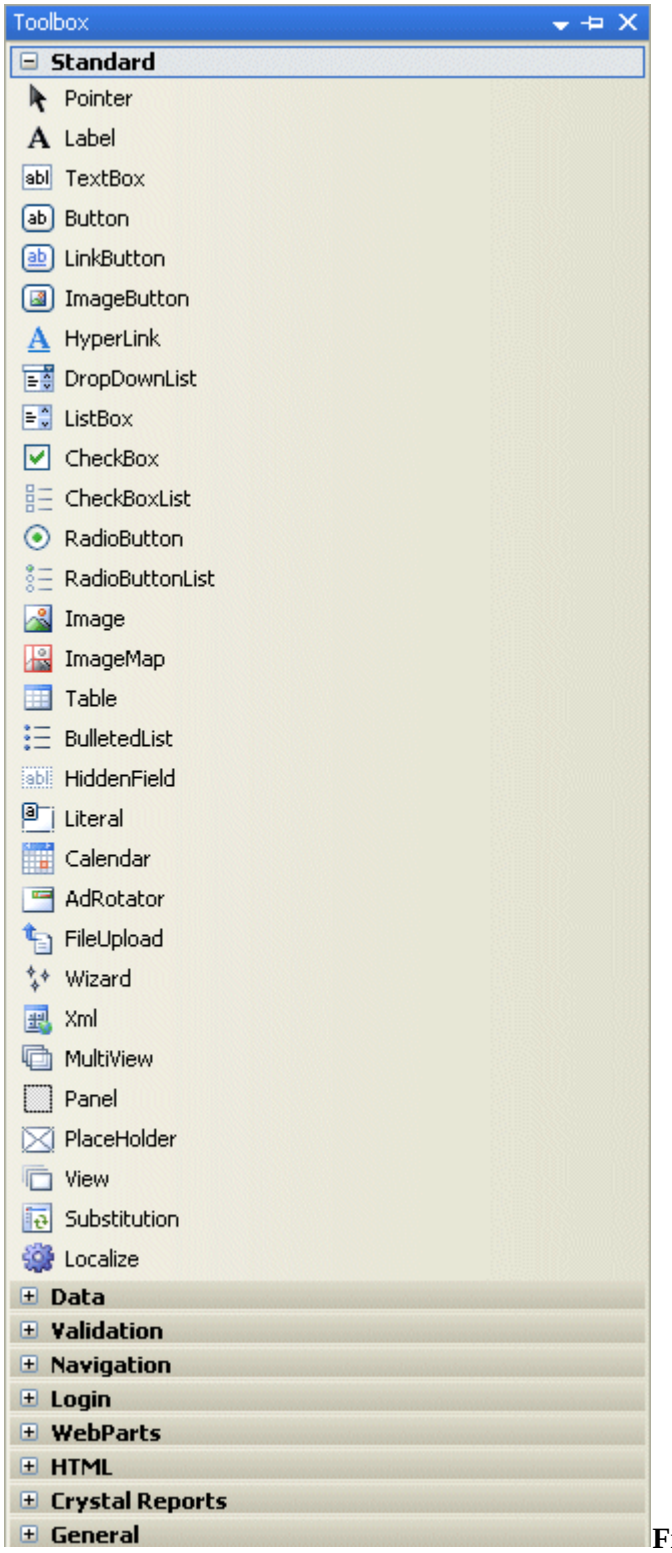


Figure 4: The Toolbox contains all the controls categorized by function.

To add items to the Web site, right-click on the Web site name and select **Add New Item** from the shortcut menu. You can see in the Add New Item dialog box **(Figure 5)** the type of things that you can add to your Web site. Note the "Place code in separate file" check box on the Add New Item dialog box. It represents the code-behind model implemented in ASP.NET 2.0.

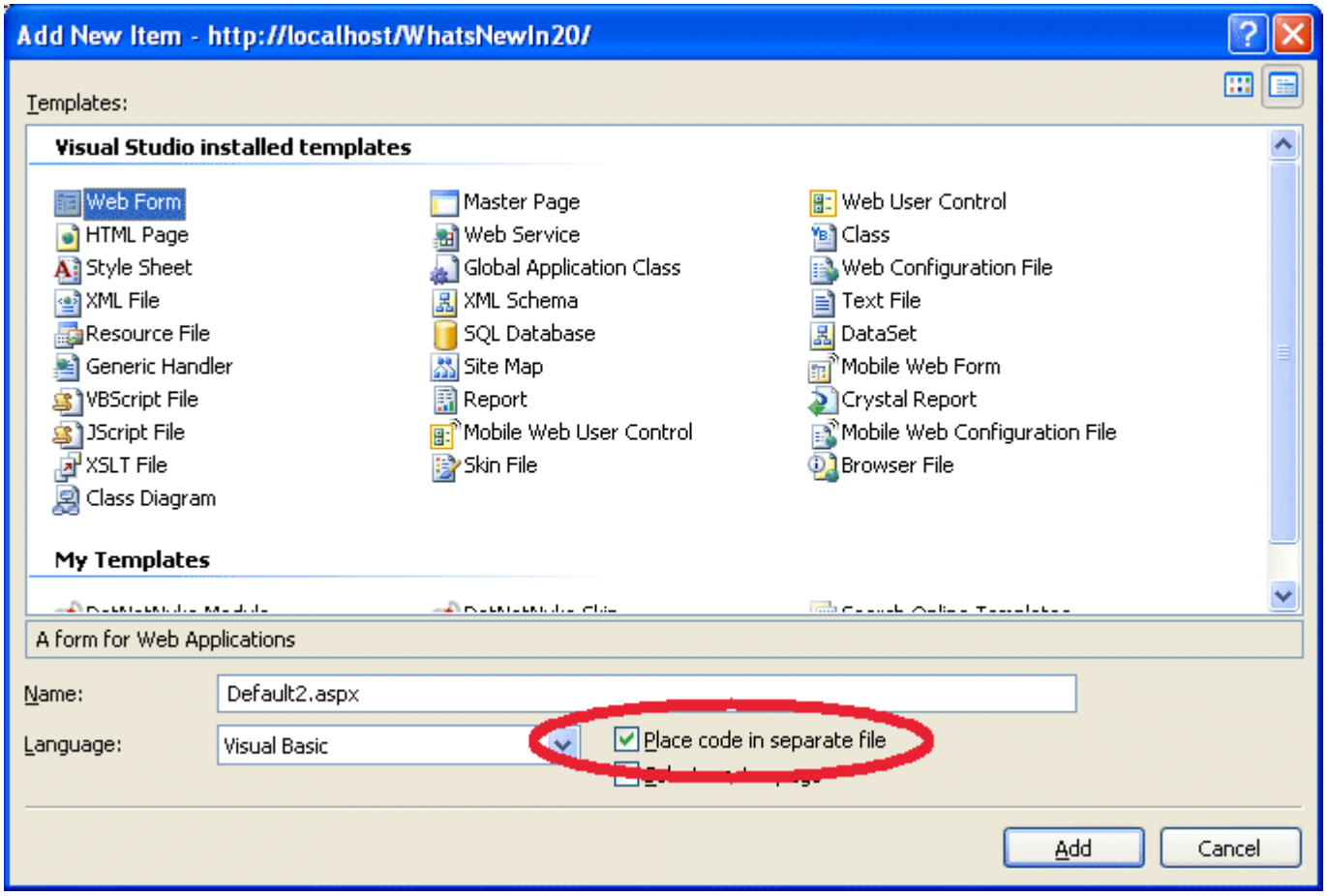


Figure 5: Use the Add New Item dialog box to add items to your Web project.

The code-behind model uses a feature in the .NET Framework 2.0 called partial classes. Partial classes are classes defined in two or more source files and combined at compilation time to create a single class. Prior to ASP.NET 2.0, control declarations were stored in the Form Designer-generated section of code-behind files. ASP.NET 2.0 stores the control declarations in a separate file from the file where you create your own code-behind classes. To make sure that the control declarations do not get out of sync, the declarations for the Web controls are not generated by Visual Studio 2005 at design time but are created by the ASP.NET runtime during compilation.

Throughout this article I'll make references to smart tags, which you can think of as a list of the most commonly used properties or tasks for a control. You access the smart tags for a control by clicking the glyph (small black triangle) in the upper right-hand corner of the control.

System-Defined Folders

Table 1 illustrates some special system-defined folders.
You store all of your non-page classes such as utility classes, business objects, or any other classes you may have created in the App_Code folder. To add an App_Code folder to your project, right-click on the project name and select **Add ASP.NET Folder** and then select **App_Code**. ASP.NET monitors the folder and whenever you change a class in the App_Code folder, ASP.NET will dynamically recompile it and automatically make it

available to the entire site. You can even place classes coded in different languages in subdirectories under the App_Code directory. Each subdirectory needs to be registered with a particular language in Web.config.
The App_Data folder makes database integration easy. The App_Data folder holds file-based data stores including SQL Server Express 2005 database files (.mdf files), Access database files (.mdb files), Excel worksheets, XML files, and so on. The primary advantage of using App_Data is that files placed there are not downloadable if a request is made for the file over the wire.

ASP.NET's compilation model offers a number of benefits including the ability to run the current form each time you run the site. A project's Start Options, available on the project's Property Pages dialog box, specifies, among other options, the ability to launch the current page or a specific page each time the site is launched **(Figure 6)**. See the article by Rick Strahl referred to in the sidebar, Compilation and Deployment.

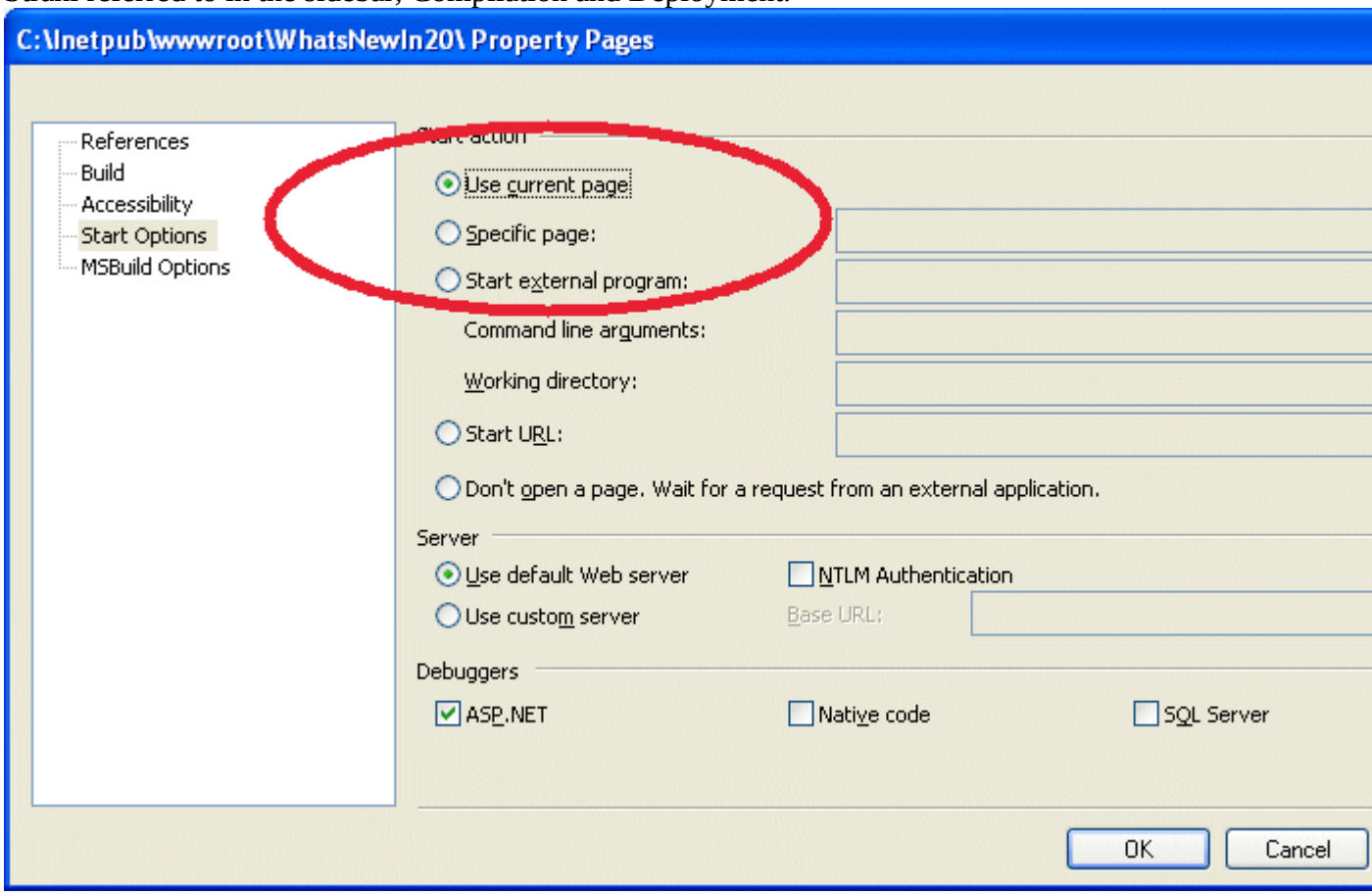


Figure 6: You can specify the launch the current page or a specific page start page in the Start Options.

The ASP.NET Web Site Administration Tool page **(Figure 7)** makes it fairly easy to administer your Web site. You can launch it by selecting the ASP.NET Administration icon on the top of the Solution Explorer **(Figure 8)**. You can manage all aspects of your site with this tool including security, application settings, and providers. The security features include being able to add and delete users, assign folder permissions, add and delete user roles, and assign users to specific roles. The application settings include being able to add and remove application variables, configure e-mail settings, take the site offline, configure the debugging and tracing options, and specify which page to use as the default error.

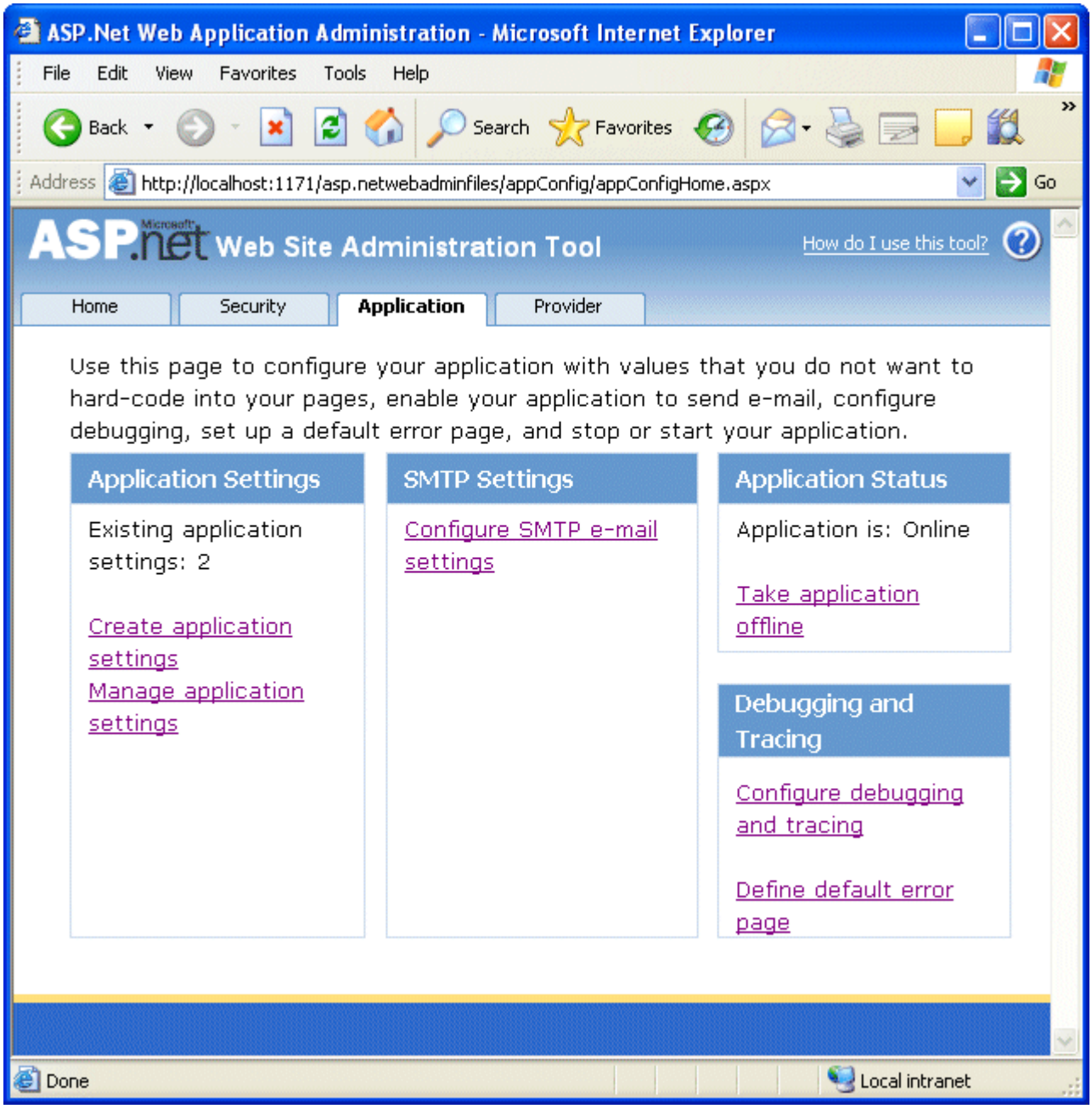


Figure 7: The ASP.NET Web Site Administration Tool provides site configuration features.

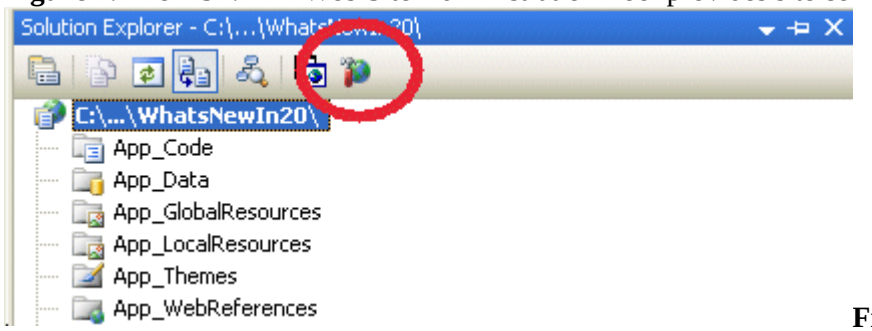


Figure 8: The Web Site Administration tool is accessed from the Solution Explorer.

ASP.NET 2.0 Web Controls

I'd like to now spend a little time discussing some of the important Web controls in ASP.NET 2.0, including the **BulletedList, HiddenField, Multiview, Wizard**, and **ImageMap** controls.

BulletedList Control

The **BulletedList** control provides a quick and easy way to develop both bulleted and numbered lists. You can manually enter a list **(Figure 9)** or you can use data binding to populate the items in the list. The **BulletStyle** property offers a number of options for bulleted and numbered lists.

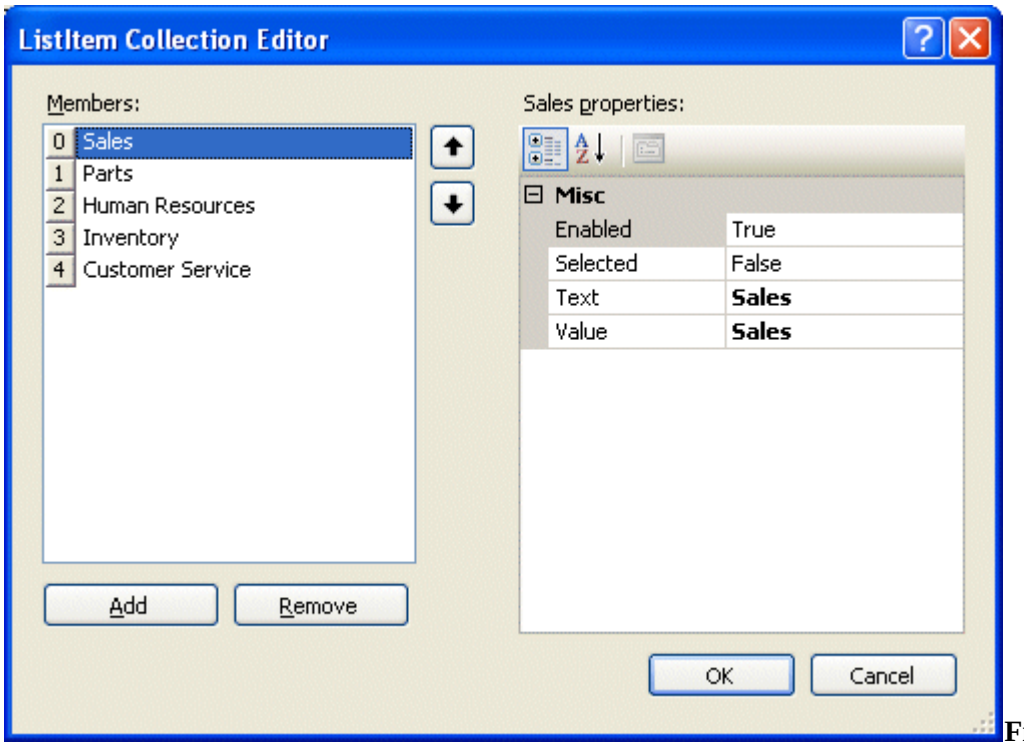


Figure 9: The BulletedList control makes creating manually maintained and data bound lists easy.

HiddenField Control

Storing data in hidden fields on a page is nothing new and is in fact a very common technique to save state data across posts back to the server. ASP.NET 2.0's **HiddenField** control provides the ability to create a hidden field on a page and use it to store data to be sent back to the server on a post back.

FileUpload Control

Many Web sites provide the ability for users to upload files. For example, an employment recruiting firm may allow people to upload their resume or a printing company might allow customers to upload their print jobs. Microsoft's ASP.NET team designed the **FileUpload** control to make this an easy feature to add to your site. **The FileUpload** control is comprised of a textbox to allow the user to type the file information and a Browse button that allows the user to go search for the file. To get the file uploaded you must add a separate button that results in a postback and executes the code to upload the file **(Listing 1)**. In this code, the FilePath variable is going to contain where the file will be uploaded to. If there is a value in the textbox portion of the **FileUpload** control, indicated by

the HasFile property, then the file name is combined with the file upload path. Next, the **SaveAs** method for the control is executed thereby actually uploading the file.

MultiView and Wizard Controls

The **MultiView** and **Wizard** controls both allow you to create multiple sections of controls on the same page. The Wizard control has features built in to facilitate its operation such as built-in Next and Prev buttons. With the **Multiview** control the responsibility to add and code the navigation falls to you.

MultiView

As the name implies, a **MultiView** control is a container control hosting a number of **View** controls. Each **View** control is an independent section which in turn hosts its own controls **(Figure 10)**. Use **MulitView** controls to replace multiple forms related to a specific function, such as you'd find in an online shopping cart. Instead of having multiple forms for each action in a shopping cart, you could use a single **MultiView** control and simply control which views a page displays at specific times. The **ActiveViewIndex** property holds the index of the currently active view. Assigning **ActiveViewIndex** to 0 displays the first view in the control. Programmatically assigning a value to the **ActiveViewIndex** property, or calling the **SetActiveView** method and passing a view object reference, is one technique for displaying a specific View control.

You can also specify which view to display using a technique that doesn't require writing any code at all. For example, you can add a **CommandButton** and set its **CommandName** property to **NextView** or **PrevView** thereby causing the **MultiView** control to automatically switch views when a user clicks the button **(Figure 11)**. If you prefer jumping to a specific view instead of moving forward or backward one view at a time, you can set the CommandButton's **CommandName** property to **SwitchViewByIndex** and its **CommandAgrument** property to the index of the view to switch to. To switch to a view by its ID instead of index position you can set the CommandButton's CommandName property to **SwitchViewByID** and its **CommandArguement** property to the ID of the view.

ASP.NET - Server Controls

Controls are small building blocks of the graphical user interface, which include text boxes, buttons, check boxes, list boxes, labels, and numerous other tools. Using these tools, the users can enter data, make selections and indicate their preferences.

Controls are also used for structural jobs, like validation, data access, security, creating master pages, and data manipulation.

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

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

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

- **Validation controls** - These are used to validate user input and they work by running client-side script.
- **Data source controls** - These controls provides data binding to different data sources.
- **Data view controls** - These are various lists and tables, which can bind to data from data sources for displaying.
- **Personalization controls** - These are used for personalization of a page according to the user preferences, based on user information.
- **Login and security controls** - These controls provide user authentication.
- **Master pages** - These controls provide consistent layout and interface throughout the application.
- **Navigation controls** - These controls help in navigation. For example, menus, tree view etc.
- **Rich controls** - These controls implement special features. For example, AdRotator, FileUpload, and Calendar control.

The syntax for using server controls is:

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

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

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

Properties of the Server Controls

ASP.NET server controls with a visual aspect are derived from the WebControl class and inherit all the properties, events, and methods of this class.

The WebControl class itself and some other server controls that are not visually rendered are derived from the System.Web.UI.Control class. For example, PlaceHolder control or XML control.

ASP.Net server controls inherit all properties, events, and methods of the WebControl and System.Web.UI.Control class.

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

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

	surface.
DisabledCssClass	Gets or sets the CSS class to apply to the rendered HTML element when the control is disabled.
Enabled	Indicates whether the control is grayed out.
EnableTheming	Indicates whether theming applies to the control.
EnableViewState	Indicates whether the view state of the control is maintained.
Events	Gets a list of event handler delegates for the control.
Font	Font.
Forecolor	Foreground color.
HasAttributes	Indicates whether the control has attributes set.
HasChildViewState	Indicates whether the current server control's child controls have any saved view-state settings.
Height	Height in pixels or %.
ID	Identifier for the control.
IsChildControlStateCleared	Indicates whether controls contained within this control have control state.
IsEnabled	Gets a value indicating whether the control is enabled.
IsTrackingViewState	It indicates whether the server control is saving changes to its view state.
IsViewStateEnabled	It indicates whether view state is enabled for this control.
LoadViewStateById	It indicates whether the control participates in loading its view state by ID instead of index.

Page	Page containing the control.
Parent	Parent control.
RenderingCompatibility	It specifies the ASP.NET version that the rendered HTML will be compatible with.
Site	The container that hosts the current control when rendered on a design surface.
SkinID	Gets or sets the skin to apply to the control.
Style	Gets a collection of text attributes that will be rendered as a style attribute on the outer tag of the Web server control.
TabIndex	Gets or sets the tab index of the Web server control.
TagKey	Gets the HtmlTextWriterTag value that corresponds to this Web server control.
TagName	Gets the name of the control tag.
TemplateControl	The template that contains this control.
TemplateSourceDirectory	Gets the virtual directory of the page or control containing this control.
ToolTip	Gets or sets the text displayed when the mouse pointer hovers over the web server control.
UniqueID	Unique identifier.
ViewState	Gets a dictionary of state information that saves and restores the view state of a server control across multiple requests for the same page.
ViewStateIgnoreCase	It indicates whether the StateBag object is case-insensitive.
ViewStateMode	Gets or sets the view-state mode of this control.

Visible	It indicates whether a server control is visible.
Width	Gets or sets the width of the Web server control.

Methods of the Server Controls

The following table provides the methods of the server controls:

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

	related properties.
DataBind	Binds a data source to the server control and all its child controls.
DataBind(Boolean)	Binds a data source to the server control and all its child controls with an option to raise the DataBinding event.
DataBindChildren	Binds a data source to the server control's child controls.
Dispose	Enables a server control to perform final clean up before it is released from memory.
EnsureChildControls	Determines whether the server control contains child controls. If it does not, it creates child controls.
EnsureID	Creates an identifier for controls that do not have an identifier.
Equals(Object)	Determines whether the specified object is equal to the current object.
Finalize	Allows an object to attempt to free resources and perform other cleanup operations before the object is reclaimed by garbage collection.
FindControl(String)	Searches the current naming container for a server control with the specified id parameter.
FindControl(String, Int32)	Searches the current naming container for a server control with the specified id and an integer.
Focus	Sets input focus to a control.
GetDesignModeState	Gets design-time data for a control.
GetType	Gets the type of the current instance.
GetUniqueIDRelativeTo	Returns the prefixed portion of the UniqueID property of the specified control.

HasControls	Determines if the server control contains any child controls.
HasEvents	Indicates whether events are registered for the control or any child controls.
IsLiteralContent	Determines if the server control holds only literal content.
LoadControlState	Restores control-state information.
LoadViewState	Restores view-state information.
MapPathSecure	Retrieves the physical path that a virtual path, either absolute or relative, maps to.

HTML Input Controls

The following controls, which are based on the HTML INPUT element, are available on the HTML tab of the Toolbox:

- **Input (Button)** control: INPUT type="button" element
- **Input (Checkbox)** control: INPUT type="checkbox" element
- **Input (File)** control: INPUT type="file" element
- **Input (Hidden)** control: INPUT type="hidden" element
- **Input (Password)** control: INPUT type="password" element
- **Input (Radio)** control: INPUT type="radio" element
- **Input (Reset)** control: INPUT type="reset" element
- **Input (Submit)** control: INPUT type="submit" element
- **Input (Text)** control: INPUT type="text" element

HTML server controls added from the Toolbox to a page in Visual Studio are simply HTML elements with certain attributes already set. You can also create HTML elements in Source view by typing markup.

By default, HTML elements on a Web Forms page are not available to the server; they are treated as markup that is passed through to the browser. However, if you add an id attribute and the attribute runat="server", ASP.NET recognizes the element as a control on the page and you can program it with server-based code.

Unlike other HTML elements, if you convert an HTML INPUT element to an ASP.NET server control, it is not created as an instance of the HtmlInputControl class. You cannot

create an instance of the HtmlInputControl class directly. Instead, this class is inherited by the classes listed in the table below.

The following table lists the type that is used to instantiate INPUT elements as ASP.NET server controls if the markup contains the attribute runat="server" and an id attribute.

Server control	Type
Button control	HtmlInputButton
CheckBox control	HtmlInputCheckBox
File Field control	HtmlInputFile
Hidden control	HtmlInputHidden
Password control	HtmlInputPassword
Radio Button control	HtmlInputRadioButton
Reset Button control	HtmlInputReset
Submit Button control	HtmlInputSubmit
Text Field control	HtmlInputText

For more information, see ASP.NET Web Server Controls Overview. For a list of HTML controls, see HTML Server Controls.

Security Note:

User input in a Web Forms page can include potentially malicious client script. By default, the Web Forms page validates that user input does not include script or HTML elements. For more information, see Script Exploits Overview and How to: Protect Against Script Exploits in a Web Application by Applying HTML Encoding to Strings.

ASP.NET - Data Binding

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

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

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

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

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

- DataBoundControl
- HierarchicalDataBoundControl

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

- ListControl
- CompositeDataBoundControl

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

- BulletedList
- CheckBoxList
- DropDownList
- ListBox
- RadioButtonList

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

- DetailsView
- FormView
- GridView
- RecordList

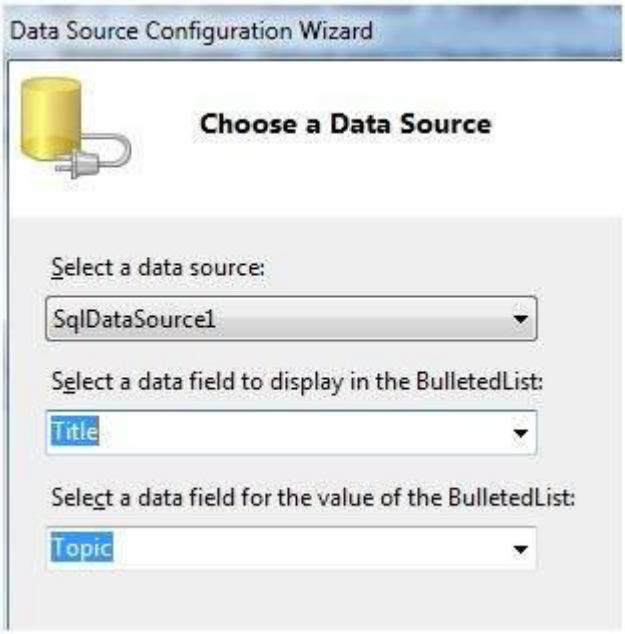
Simple Data Binding

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

Let us take up a small example to understand the concept. Create a web site with a bulleted list and a SqlDataSource control on it. Configure the data source control to retrieve two values from your database (we use the same DotNetReferences table as in the previous chapter).

Choosing a data source for the bulleted list control involves:

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



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

Declarative Data Binding

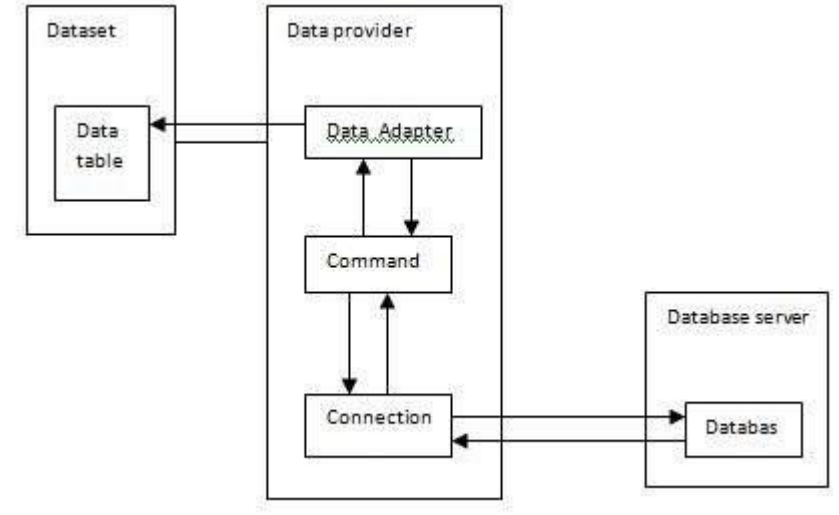
We have already used declarative data binding in the previous tutorial using GridView control. The other composite data bound controls capable of displaying and manipulating data in a tabular manner are the DetailsView, FormView, and RecordList control.

In the next tutorial, we will look into the technology for handling database, i.e, ADO.NET.

However, the data binding involves the following objects:

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

Relation between the data binding objects:



Example

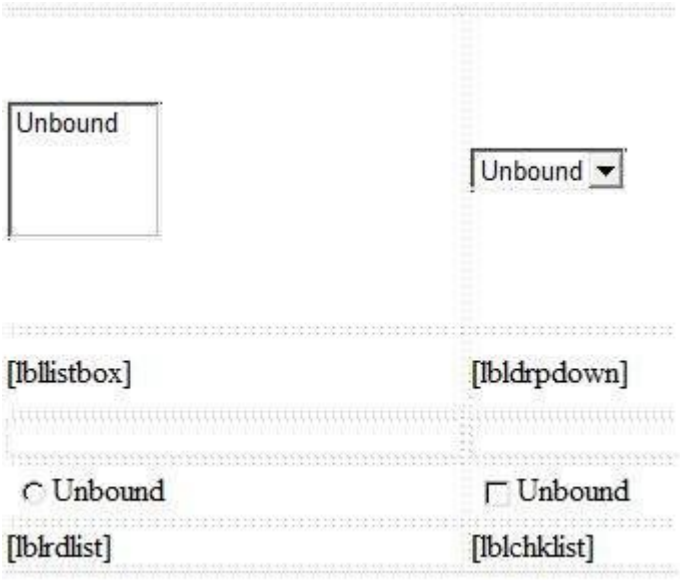
Let us take the following steps:

Step (1) : Create a new website. Add a class named booklist by right clicking on the solution name in the Solution Explorer and choosing the item 'Class' from the 'Add Item' dialog box. Name it as booklist.cs.

```
using System;
using System.Data;
using System.Configuration;
using System.Linq;

using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

using System.Xml.Linq;

namespace databinding
{
  public class booklist
  {
    protected String bookname;
    protected String authorname;
    public booklist(String bname, String aname)
    {
      this.bookname = bname;
      this.authorname = aname;

    }

    public String Book
```

```
    {
      get
      {
        return this.bookname;
      }
      set
      {
        this.bookname = value;
      }
    }

    public String Author
    {
      get
      {
        return this.authorname;
      }
      set
      {
        this.authorname = value;
      }
    }
  }
}
```

Step (2) : Add four list controls on the page a list box control, a radio button list, a check box list, and a drop down list and four labels along with these list controls. The page should look like this in design view:



The source file should look as the following:

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

    <table style="width: 559px">
```

```
        <tr>
          <td style="width: 228px; height: 157px;">
            <asp:ListBox ID="ListBox1" runat="server" AutoPostBack="True"
              OnSelectedIndexChanged="ListBox1_SelectedIndexChanged">
            </asp:ListBox>
          </td>

          <td style="height: 157px">
            <asp:DropDownList ID="DropDownList1" runat="server"
              AutoPostBack="True"
OnSelectedIndexChanged="DropDownList1_SelectedIndexChanged">
            </asp:DropDownList>
          </td>
        </tr>

        <tr>
          <td style="width: 228px; height: 40px;">
            <asp:Label ID="lbllistbox" runat="server"></asp:Label>
          </td>

          <td style="height: 40px">
            <asp:Label ID="lbldrpdown" runat="server">
            </asp:Label>
          </td>
        </tr>

        <tr>
          <td style="width: 228px; height: 21px">
          </td>

          <td style="height: 21px">
          </td>
        </tr>

        <tr>
          <td style="width: 228px; height: 21px">
            <asp:RadioButtonList ID="RadioButtonList1" runat="server"
              AutoPostBack="True"
OnSelectedIndexChanged="RadioButtonList1_SelectedIndexChanged">
            </asp:RadioButtonList>
          </td>

          <td style="height: 21px">
            <asp:CheckBoxList ID="CheckBoxList1" runat="server"
              AutoPostBack="True"
OnSelectedIndexChanged="CheckBoxList1_SelectedIndexChanged">
            </asp:CheckBoxList>
          </td>
        </tr>

        <tr>
          <td style="width: 228px; height: 21px">
```

```
            <asp:Label ID="lblrdlist" runat="server">
            </asp:Label>
          </td>

          <td style="height: 21px">
            <asp:Label ID="lblchklist" runat="server">
            </asp:Label>
          </td>
        </tr>
      </table>

    </div>
</form>
```

Step (3) : Finally, write the following code behind routines of the application:

```
public partial class _Default : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
    IList bklist = createbooklist();

    if (!this.IsPostBack)
    {
      this.ListBox1.DataSource = bklist;
      this.ListBox1.DataTextField = "Book";
      this.ListBox1.DataValueField = "Author";

      this.DropDownList1.DataSource = bklist;
      this.DropDownList1.DataTextField = "Book";
      this.DropDownList1.DataValueField = "Author";

      this.RadioButtonList1.DataSource = bklist;
      this.RadioButtonList1.DataTextField = "Book";
      this.RadioButtonList1.DataValueField = "Author";

      this.CheckBoxList1.DataSource = bklist;
      this.CheckBoxList1.DataTextField = "Book";
      this.CheckBoxList1.DataValueField = "Author";

      this.DataBind();
    }
  }

  protected IList createbooklist()
  {
    ArrayList allbooks = new ArrayList();
    booklist bl;

    bl = new booklist("UNIX CONCEPTS", "SUMITABHA DAS");
    allbooks.Add(bl);

    bl = new booklist("PROGRAMMING IN C", "RICHI KERNIGHAN");
```

```
    allbooks.Add(bl);

    bl = new booklist("DATA STRUCTURE", "TANENBAUM");
    allbooks.Add(bl);

    bl = new booklist("NETWORKING CONCEPTS", "FOROUZAN");
    allbooks.Add(bl);

    bl = new booklist("PROGRAMMING IN C++", "B. STROUSTROUP");
    allbooks.Add(bl);

    bl = new booklist("ADVANCED JAVA", "SUMITABHA DAS");
    allbooks.Add(bl);

    return allbooks;
  }

  protected void ListBox1_SelectedIndexChanged(object sender, EventArgs e)
  {
    this.lbllistbox.Text = this.ListBox1.SelectedValue;
  }

  protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
  {
    this.lbldrpdown.Text = this.DropDownList1.SelectedValue;
  }

  protected void RadioButtonList1_SelectedIndexChanged(object sender, EventArgs e)
  {
    this.lblrdlist.Text = this.RadioButtonList1.SelectedValue;
  }

  protected void CheckBoxList1_SelectedIndexChanged(object sender, EventArgs e)
  {
    this.lblchklist.Text = this.CheckBoxList1.SelectedValue;
  }
}
```

Observe the following:

- The booklist class has two properties: bookname and authorname.
- The createbooklist method is a user defined method that creates an array of booklist objects named allbooks.
- The Page_Load event handler ensures that a list of books is created. The list is of IList type, which implements the IEnumerable interface and capable of being bound to the list controls. The page load event handler binds the IList object 'bklist' with the list controls. The bookname property is to be displayed and the authorname property is considered as the value.

SHORT QUESTIONS

1.Define features of asp.net 2.o?

2.Write about server controls?

3.Define user controls?

4.Define master detail forms?

5.Explain html controls?

LONG QUESTIONS

1.Discuss about web forms processing?

2.Discuss about html controls and validation controls?

3.Give a brief note on data binding controls?

4.Discuss about server controls?

5.Explain about database connectity with ado.net?

Internal Examinations-1

Section-A

Answer any five of the following Questions. 5*2=10m

1.Explain about .NET framework?

2.Define garbage collection?

3.Define Object oriented concept in c#?

4.Explain about Regular Expressions in c#?

5.Define multithreading?

6.Explain about Error Handling?

7.Define Data handling?

8.Explain about Delegates and events in c#?

Section-B

Answer any one Question from each unit. 10*2=20m

Unit-1

9a).Explain about Justin Time compilation and garbage collection?

b). Explain about variables and C# statements?

10a). Explain about Arrays and Strings in C#?

b). Define System collections and Regular Expressions in c#?

Unit-2

11a). What are the input output libraries?

b). Explain about Networking and Sockets?

12a). Explain about Web applications and Error handling?

b). What are the Index Attributes and explain in detail?

Internal Examinations-2

Section-A

Answer any five of the following Questions. 5*2=10m

1.define Error Handling?

2. define C# Using Libraries?

3.what is data handling?

4.define features of .NET?

5.What is Multithreading?

6.what are the libraries for Image Processing?

7.define about Regular Expressions?

8 define data types?

Section-B

Answer any one Question from each unit. 10*2=20m

Unit-1

9.a)Explain about .NET Framework in detail?

b)describe about System Collections?

10.a)define about Libraries used in C#?

b) Explain about Justin Time compilation and garbage collection?

Unit-2

11.a)define Namespace-System and also explain libraries in C#?

b)define Data Handling and Windows Forms?

12.a)Explain about Input and Output Concepts in Detail?

b)deine about Networking and Sockets?

Semester Paper

MASTER OF COMPUTER APLLICATIONS DEGREE EXAMINATION

FOURTH SEMESTER

PAPER MCA 401B: Dot Net Technologies

(Under C.B.S.C Revised Regulations w.e.f.2021-2023)

(Common paper to University and all Affliated Colleges)

Time:3 hours Max.Marks:70

PART-A

Answer any five of the following questions each question carries 2 marks(5*2=10)

1.a) define features of .NET?

b).What is Multithreading?

c). 1.Define features of asp.net 2.o?

d).Write about server controls?

e)Define user controls?

f)Define master detail forms?

g)Explain html controls?

h) define Error Handling?

i) define C# Using Libraries?

j).what is data handling?

PART-B

Answer any one full question from each unit

Each question carries 12 marks(5*12=60)

Unit-1

2.Explain about .NET Framework in detail?

(or)

3.describe about System Collections?

Unit-2

4.define about Libraries used in C#?

(or)

5. Explain about Justin Time compilation and garbage collection?

Unit-3

6.Discuss about web forms processing?

(or)

7.Discuss about html controls and validation controls?

Unit-4

8.Give a brief note on data binding controls?

(or)

9.Discuss about server controls?

Unit-5

10.Explain about database connectity with ado.net?

(or)

11.describe briefly on the data types and data frameworks?