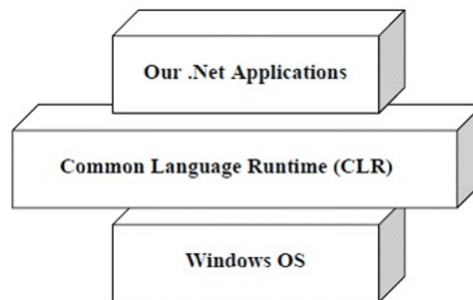
# .Net Architecture and .Net Framework basics:

1. **Common Language Runtime (CLR):** The heart of the .Net Framework. It is also called the .Net runtime. It resides above the operating system and handles all .Net applications. It handles garbage collection, Code Access Security (CAS) etc.

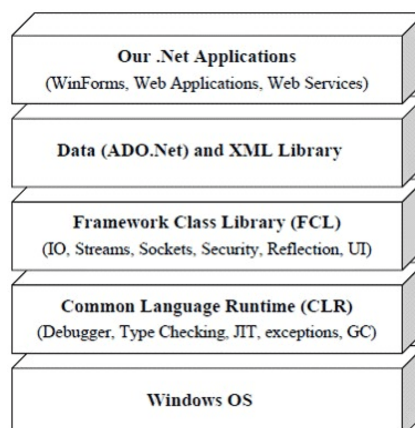



2. **Microsoft Intermediate Language (MSIL) Code:** When we compile our .Net code then it is not directly converted to native/binary code; it is first converted into intermediate code known as MSIL code which is then interpreted by the CLR. MSIL is independent of hardware and the operating system. Cross language relationships are possible since MSIL is the same for all .Net languages. MSIL is further converted into native code.

3. **Just in Time Compilers (JIT):** It compiles IL code into native executable code (exe or dlls). Once code is converted to IL then it can be called again by JIT instead of recompiling that code.
4. **Framework class library:** The .Net Framework provides a huge class library called FCL for common tasks. It contains thousands of classes to access Windows APIs and common functions like string manipulations, Data structures, stream, IO, thread, security etc.
5. **Common Language Specification (CLS):** What makes a language to be .Net compliant? Answer is CLS. Microsoft has defined some specifications that each .Net language has to follow. For e.g.: no pointer, no multiple inheritances etc.

6. **Common Type System (CTS):** CTS defines some basic data types that IL can understand. Each .Net compliant language should map its data types to these standard data types. This makes it possible for two .Net compliant languages to communicate by ing/receiving parameters to and from each other. For example CTS defines Int32 for C# int and VB integer data types.

7. **The .Net Framework:** Is a combination of CLR, FCL, ADO.Net and XML classes, Web/Window applications and Web services.



# C# - Overview

C# is a modern, general-purpose, object-oriented programming language developed by Microsoft and approved by European Computer Manufacturers Association (ECMA) and International Standards Organization (ISO).

C# was developed by Anders Hejlsberg and his team during the development of .Net Framework.

C# is designed for Common Language Infrastructure (CLI), which consists of the executable code and runtime environment that allows use of various high-level languages on different computer platforms and architectures.

The following reasons make C# a widely used professional language:

- It is a modern, general-purpose programming language
- It is object oriented.
- It is component oriented.
- It is easy to learn.
- It is a structured language.
- It produces efficient programs.
- It can be compiled on a variety of computer platforms.
- It is a part of .Net Framework.

## Strong Programming Features of C#

Although C# constructs closely follow traditional high-level languages, C and C++ and being an object-oriented programming language. It has strong resemblance with Java, it has numerous strong programming features that make it endearing to a number of programmers worldwide.

Following is the list of few important features of C#:

- Boolean Conditions
- Automatic Garbage Collection
- Standard Library
- Assembly Versioning
- Properties and Events
- Delegates and Events Management
- Easy-to-use Generics
- Indexers
- Conditional Compilation
- Simple Multithreading
- LINQ and Lambda Expressions
- Integration with Windows

## Microsoft .Net Metadata

Metadata in .Net is binary information which describes the characteristics of a resource . This information include Description of the Assembly , Data Types and members with their declarations and implementations, references to other types and members , Security permissions etc. A module's metadata contains everything that needed to interact with another module.

During the compile time Metadata created with Microsoft Intermediate Language (MSIL) and stored in a file called a Manifest . Both Metadata and Microsoft Intermediate Language (MSIL) together wrapped in a Portable Executable (PE) file. During the runtime of a program Just In Time (JIT) compiler of the Common Language Runtime (CLR) uses the Metadata and converts Microsoft Intermediate Language (MSIL) into native code. When code is executed, the runtime loads metadata into memory and references it to discover information about your code's classes, members, inheritance, and so on. Moreover Metadata eliminating the need for Interface Definition Language (IDL) files, header files, or any external method of component reference.

## Microsoft .Net Assembly

Microsoft **.Net Assembly** is a logical unit of code, that contains code which the Common Language Runtime (CLR) executes. It is the smallest unit of deployment of a .net application and it can be a **.dll** or an **exe** . Assembly is really a collection of types and resource information that are built to work together and form a logical unit of functionality. It include both executable application files that you can run directly from Windows without the need for any other programs (.exe files), and libraries (.dll files) for use by other applications.

Assemblies are the building blocks of .NET Framework applications. During the compile time Metadata is created with Microsoft Intermediate Language (MSIL) and stored in a file called Assembly Manifest . Both Metadata and Microsoft Intermediate Language (MSIL) together wrapped in a Portable Executable (PE) file. Assembly Manifest contains information about itself. This information is called Assembly Manifest, it contains information about the members, types, references and all the other data that the runtime needs for execution.

Every Assembly you create contains one or more program files and a Manifest. There are two types program files : Process Assemblies (EXE) and Library Assemblies (DLL). Each Assembly can have only one entry point (that is, DllMain, WinMain, or Main).

We can create two types of Assembly:

1. Private Assembly

2. Shared Assembly

A private Assembly is used only by a single application, and usually it is stored in that application's install directory. A shared Assembly is one that can be referenced by more than one application. If multiple applications need to access an Assembly, we should add the Assembly to the Global Assembly Cache (GAC). There is also a third and least known type of an assembly: Satellite Assembly . A Satellite Assembly contains only static objects like images and other non-executable files required by the application.

## .Net Assembly Manifest

An Assembly Manifest is a file that containing Metadata about .NET Assemblies. Assembly Manifest contains a collection of data that describes how the elements in the assembly relate to each other. It describes the relationship and dependencies of the components in the Assembly, versioning information, scope information and the security permissions required by the Assembly.

The Assembly Manifest can be stored in Portable Executable (PE) file with Microsoft Intermediate Language (MSIL) code. You can add or change some information in the Assembly Manifest by using assembly attributes in your code. The Assembly Manifest can be stored in either a PE file (an .exe or .dll) with Microsoft Intermediate Language (MSIL) code or in a standalone PE file that contains only assembly manifest information. Using ILDasm, you can view the manifest information for any managed DLL.

## Common Language Specification - CLS

Common Language Specification (CLS) is a set of basic language features that .Net Languages needed to develop Applications and Services , which are compatible with the .Net Framework. When there is a situation to communicate Objects written in different .Net Complaint languages , those objects must expose the features that are common to all the languages . Common Language Specification (CLS) ensures complete interoperability among applications, regardless of the language used to create the application.

Common Language Specification (CLS) defines a subset of Common Type System (CTS) . Common Type System (CTS) describes a set of types that can use different .Net languages have in common , which ensure that objects written in different languages can interact with each other. Most of the members defined by types in the .NET Framework Class Library (FCL) are Common Language Specification (CLS) compliant Types. Moreover Common Language Specification (CLS) standardized by ECMA .

## Common Type System - CTS

Common Type System (CTS) describes a set of types that can be used in different .Net languages in common . That is , the Common Type System (CTS) ensure that objects written in different .Net languages can interact with each other. For Communicating between programs written in any .NET complaint language, the types have to be compatible on the basic level .

These types can be Value Types or Reference Types . The Value Types are passed by values and stored in the stack. The Reference Types are passed by references and stored in the heap. Common Type System (CTS) provides base set of Data Types which is responsible for cross language integration. The Common Language Runtime (CLR) can load and execute the source code written in any .Net language, only if the type is described in the Common Type System (CTS) .Most of the members defined by types in the .NETFramework Class Library (FCL) are Common Language Specification(CLS) compliant Types.

## Difference between managed and unmanaged code

### What is Managed Code -

Managed code is the code that is written to target the services of the managed runtime execution environment such as Common Language Runtime in .Net Technology.

The Managed Code running under a Common Language Runtime cannot be accessed outside the runtime environment as well as cannot call directly from outside the runtime environment. It refers to a contract of cooperation between natively executing code and the runtime. It offers services like garbage collection, run-time type checking, reference checking etc. By using managed code you can avoid many typical programming mistakes that lead to security holes and unstable applications, also, many unproductive programming tasks are automatically taken care of, such as type safety checking, memory management, destruction of unused Objects etc.

What is Unmanaged Code –

Unmanaged code compiles straight to machine code and directly executed by the Operating System. The generated code runs natively on the host processor and the processor directly executes the code generated by the compiler. It is always compiled to target a specific architecture and will only run on the intended platform. So, if you want to run the same code on different architecture then you will have to recompile the code using that particular architecture.

Unmanaged executable files are basically a binary image, x86 code, directly loaded into memory. This approach typically results in fastest code execution, but diagnosing and recovery from errors might difficult and time consuming in most cases. The memory allocation, type safety, security, etc needs to be taken care of by the programmer and this will lead unmanaged code prone to memory leaks like buffer overruns, pointer overrides etc.

All code compiled by traditional C/C++ compilers are Unmanaged Code. COM components, ActiveX interfaces, and Win32 API functions are examples of unmanaged code. Managed code is code written in many high-level programming languages that are available for use with the Microsoft .NET Framework, including VB.NET, C#, J#, JScript.NET etc. Since Visual C++ can be compiled to either managed or unmanaged code it is possible to mix the two in the same application.

## C# Entry Point (Main) Method

In C# programming the Main method is where program starts execution. It is the main entry point of program that executes all the objects and invokes method to execute. There can be only one Main method in C#. However, the C# Main method can be void or int return type. It must be inside a class or struct and must be declared with static modifier. It is the main place where a program starts the execution and end. The Main method can have a parameter and these parameters are known as **zero-indexed** command line argument.

C#

```
class TestClass
{
    static void Main(string[] args)
    {
        // Display the number of command line arguments:
        System.Console.WriteLine(args.Length);
    }
}
```

## C# command line arguments

We can pass command line arguments to C# programs. The program accept arguments in the order of args[0], args[1] etc. The following program shows how to pass command line arguments to the c# program. Open a new text document and copy and paste the following source code and save the file as "NewProg.cs"

```
using System;

class NewProg
{
        static void Main(string[] args)
        {
Console.WriteLine("Arguments-1 "+ args[0]+ "Argument-2
"+ args[1]);
                Console.ReadKey();
        }
}
```

Go to the command prompt and issue the following command for compilation. Csc NewProg.cs

After the successful compilation you will get NewProg.exe file

When you execute this C# program you have to pass two arguments with the filename.

NewProg test1 test2

you will get the output like Arguments-1 test1 Argument-2 test2

## Differences between Stack and Heap

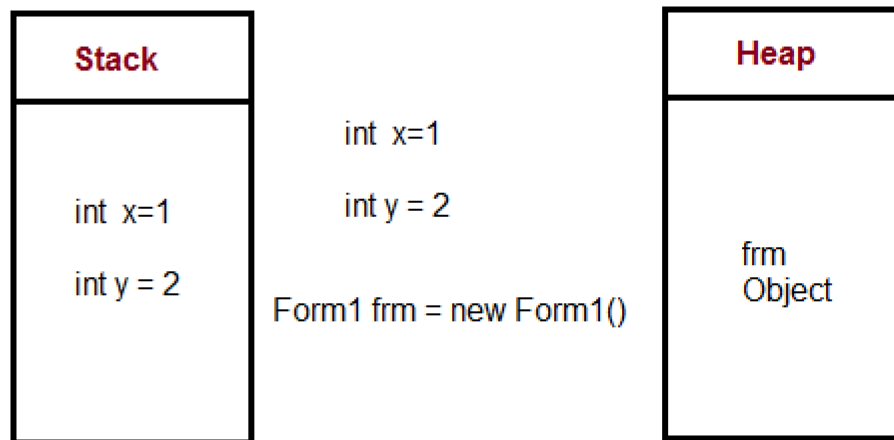
### Stack and a Heap ?

Stack is used for static memory allocation and Heap for dynamic memory allocation, both stored in the computer's RAM .

Variables allocated on the stack are stored directly to the memory and access to this memory is very fast, and it's allocation is dealt with when the program is compiled. When a function or a method calls another function which in turns calls another function etc., the execution of all those functions remains suspended until the very last function returns its value. The stack is always reserved in a LIFO order, the most recently reserved block is always the next block to be freed. This makes it really simple to keep track of the stack, freeing a block from the stack is nothing more than adjusting one pointer.

Variables allocated on the heap have their memory allocated at run time and accessing this memory is a bit slower, but the heap size is only limited by the size of virtual memory . Element of the heap have no dependencies with each other and can always be accessed randomly at any time. You can allocate a block at any time and free it

at any time. This makes it much more complex to keep track of which parts of the heap are allocated or free at any given time.

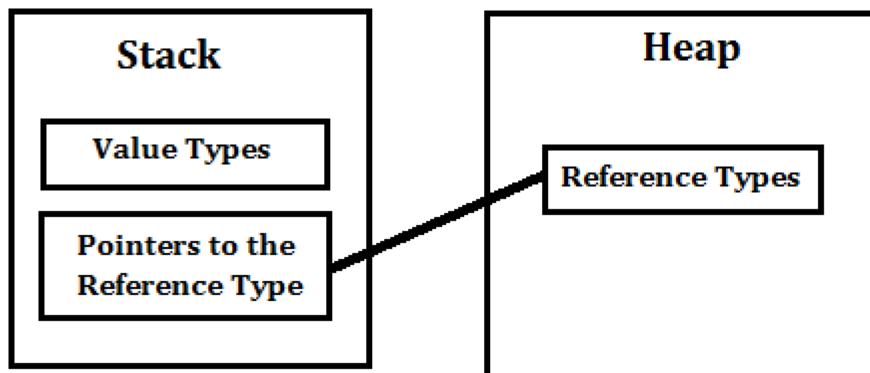



You can use the stack if you know exactly how much data you need to allocate before compile time and it is not too big. You can use heap if you don't know exactly how much data you will need at runtime or if you need to allocate a lot of data.

In a multi-threaded situation each thread will have its own completely independent stack but they will share the heap. Stack is thread specific and Heap is application specific. The stack is important to consider in exception handling and thread executions.

## Value Type and a Reference Type

The Types in .NET Framework are either treated by Value Type or by Reference Type. A Value Type holds the data within its own memory allocation and a Reference Type contains a pointer to another memory location that holds the real data. Reference Type variables are stored in the heap while Value Type variables are stored in the stack.




### Value Type:

A Value Type stores its contents in memory allocated on the stack. When you created a Value Type, a single space in memory is allocated to store the value and that variable directly holds a value. If you assign it to another variable, the value is copied directly and both variables work independently. Predefined datatypes, structures, enums are also value types, and work in the same way. Value types

can be created at compile time and Stored in stack memory, because of this, Garbage collector can't access the stack.

e.g.

```
int x = 10;
```

Here the value 10 is stored in an area of memory called the stack.

## Reference Type:

Reference Types are used by a reference which holds a reference (address) to the object but not the object itself. Because reference types represent the address of the variable rather than the data itself, assigning a reference variable to another doesn't copy the data. Instead it creates a second copy of the reference, which refers to the same location of the heap as the original value. Reference Type variables are stored in a different area of memory called the heap. This means that when a reference type variable is no longer used, it can be marked for garbage collection. Examples of reference types are Classes, Objects, Arrays, Indexers, Interfaces etc.

e.g. `int[] iArray = new int[20];`

In the above code the space required for the 20 integers that make up the array is allocated on the heap.

# NET C# Arrays

An array is a variable that can store more than one value of same data type. A normal variable can store only one value and when you want to store more than one value in a variable then declare that variable as an array.

In C# array element index starts with zero and ends with size -1 same as in case of an array in C.

## One Dimensional Arrays in C#

**Syntax** : DataType[] ArrayName = new DataType[Size];

**Example**

```
int[] digits = new int[10];
        String[] names = new string[10];
        object[] objects = new Object[10];
```

In C#, Array can be declared and initialized at a time as below

```
//Int Array
        int[] digits = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
        //String Array
        string[] names = { "John", "Bill", "Harry" };
        //Char Array
        char[] chararray = { 'a', 'b', 'c', 'd', 'e', 'f' };
```

# C# Switch Example

```
using System;

namespace switchincs
{
    class switchexample
    {
        static void Main(string[] args)
        {
            int choice = 0;
            Console.WriteLine(" 1 - I like Foot ball\n 2 - I like
Cricket\n 3 - I like Tennis");

            Console.WriteLine("Please select your choice between 1
and 3");
            choice = int.Parse(Console.ReadLine());

            switch (choice)
            {
                case 1:
                    Console.WriteLine("You like Foot ball");
                    break;
                case 2:
                    Console.WriteLine("You like Cricket");
                    break;
                case 3:
                    Console.WriteLine("You like Tennis");
                    break;
                default:
                    Console.WriteLine("Please select between 1 and
3");

                    break;
            }
            Console.Read(); //To make console visible after
processing.
        }
    }
}
```

**Output**

1 - I like Foot ball
 2 - I like Cricket
 3 - I like Tennis
Please select your choice between 1 and 3
2
You like Cricket

# Do While in C#

The do - while loop is similar to the while loop in that the loop continues as long as the specified loop condition remains true .

The main difference is that the condition is checked at the end of the loop — which contrasts with the while loop and the for loop, where the condition is checked at the beginning of the loop.

The do - while loop statement is always executed at least once.

## C# Do While Example

```
using System;

namespace ProgramCall
{

   class Sample
   {
      static void Main()
      {

         int A, B, Ch;
         string Continue;

         do
         {

            Console.WriteLine("Enter Two Integers");
```

```
            A = int.Parse(Console.ReadLine());

            B = int.Parse(Console.ReadLine());

            Console.Write("Enter Your Choice (1 - Add/2 - Sub/3 - Mul/ 4 - Div)
: ");

            Ch = int.Parse(Console.ReadLine());

            switch (Ch)
            {
               case 1:

                  Console.WriteLine("Sum Is {0}", A + B);
                  break;
               case 2:

                  Console.WriteLine("Difference Is {0}", A - B);
                  break;
               case 3:

                  Console.WriteLine("Product Is {0}", A * B);
                  break;

               case 4:

                  Console.WriteLine("Ratio Is {0}", A / B);
                  break;

               default:

                  Console.WriteLine("Wrong Choice");
                  break;
            }

            Console.Write("Do You Want To Continue? (Y/N) : ");
            Continue = Console.ReadLine();
         } while (Continue != "N" && Continue != "n");
      }
   }
}
```

```
OUTPUT
----------
Enter Two Integers
65
86
Enter Your Choice (1 - Add/2 - Sub/3 - Mul/ 4 - Div) : 1
Sum Is 151
Do You Want To Continue? (Y/N) : y
Enter Two Integers
45
21
Enter Your Choice (1 - Add/2 - Sub/3 - Mul/ 4 - Div) : 4
Ratio Is 2
Do You Want To Continue? (Y/N) :
```

## NET Programming - C# DataTypes

For Programming using C# , the following datatypes are used. All the data types in .net framework are available within the namespace System.

| Category | Class/Structre Name | Data Type in C#.NET | NO. of Bytes | Range |
|---|---|---|---|---|
| **Integer** | System.Byte | Byte | 1(Unsigned) | 0 to 255 |
| | System.SByte | Sbyte | 1(Signed) | -128 to 127 |
| | System.Int16 | Short | 2(Signed) | -32,768 to 32,767 |
| | System.Uint16 | Ushort | 2(Unsigned) | 0 to 65,535 |
| | System.Int32 | Int | 4(Signed) | -2,147,483,648 to 2,147,483,647 |
| | System.Uint32 | Uint | 4(Unsigned) | 0 to 4,294,967,295 |
| | System.Int64 | Long | 8(Signed) | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| | System.Uint64 | Ulong | 8(Unsigned) | 0 to 18,446,744,073,709,551,615 |

| **Float** | | | | |
|---|---|---|---|---|
| | Single | Float | 4 | ±1.5e−45 to ±3.4e38 (Precision:7 digits) |
| | Double | Double | 8 | ±5.0e−324 to ±1.7e308 (Precision:15-16 digits) |
| | Decimal | Decimal | 16 | (-7.9 x 1028 to 7.9 x 1028) / (100 to 28) (Precision:28-29 digits) |
| **Character** | | | | |
| | Char | Char | 2 | |
| | String | String | size varies | |
| **Other** | | | | |
| | DateTime | DateTime | 8 | |
| | Boolean | Bool | 1 | |
| | Object | Object | size varies | Can store any type of value |
| | System.IntPtr | | Platform dependent | Pointer to a memory address |

In the .NET framework, all the types are derived from System.Object. This relationship helps to establish common type system used throughout the .NET Framework.

**Note:** Of all the C# datatypes, string and object are reference types while all other are value types.

## C# boxing and unboxing

C# Type System contains three Types , they are Value Types , Reference Types and Pointer Types. C# allows us to convert a Value Type to a Reference Type, and back again to Value Types . The operation of Converting a Value Type to a Reference Type is called Boxing and the reverse operation is called Unboxing.

### Boxing

```
1:      int Val = 1;

2:      Object Obj = Val; //Boxing
```

The first line we created a Value Type Val and assigned a value to Val. The second line , we created an instance of Object Obj and assign the value of Val to Obj. From the above operation (Object Obj = i ) we saw converting a value of a Value Type into a value of a corresponding Reference Type . These types of operation is called Boxing.

### UnBoxing

```
1:      int Val = 1;
2:      Object Obj = Val; //Boxing

3:      int i = (int)Obj; //Unboxing
```

The first two line shows how to Box a Value Type . The next line (int i = (int) Obj) shows extracts the Value Type from the Object . That is converting a value of a Reference Type into a value of a Value Type. This operation is called UnBoxing.

Boxing and UnBoxing are computationally expensive processes. When a value type is boxed, an entirely new object must be allocated and constructed , also the cast required for UnBoxing is also expensive computationally.

```
using System;
using System.Windows.Forms;

namespace WindowsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            int Val = 1;
            Object Obj = Val;           //Boxing
            int i = (int)Obj;           //Unboxing
            MessageBox.Show("The value is   : " + i);
        }
    }
}
```

## DIFFERENCE BETWEEN EXE AND DLL

The terms EXE and DLL are very common in programming. When coding, you can either export your final project to either a DLL or an EXE. The term EXE is a shortened version of the word executable as it identifies the file as a program. On the other hand, DLL stands for Dynamic Link Library, which commonly contains functions and procedures that can be used by other programs.

In the basest application package, you would find at least a single EXE file that may or may not be accompanied with one or more DLL files. An EXE file contains the entry point or the part in the code where the operating system is supposed to begin the execution of the application. DLL files do not have this entry point and cannot be executed on their own.

The most major advantage of DLL files is in its reusability. A DLL file can be used in other applications as long as the coder knows the names and parameters of the functions and procedures in the DLL file. Because of this capability, DLL files are ideal for distributing device drivers. The DLL would facilitate the communication

between the hardware and the application that wishes to use it. The application would not need to know the intricacies of accessing the hardware just as long as it is capable of calling the functions on the DLL.

Launching an EXE would mean creating a process for it to run on and a memory space. This is necessary in order for the program to run properly. Since a DLL is not launched by itself and is called by another application, it does not have its own memory space and process. It simply shares the process and memory space of the application that is calling it. Because of this, a DLL might have limited access to resources as it might be taken up by the application itself or by other DLLs.

Summary:
1.EXE is an extension used for executable files while DLL is the extension for a dynamic link library.
2.An EXE file can be run independently while a DLL is used by other applications.
3.An EXE file defines an entry point while a DLL does not.
4.A DLL file can be reused by other applications while an EXE cannot.
5.A DLL would share the same process and memory space of the calling application while an EXE creates its separate process and memory space.

## Understanding C# Pass by Reference and Pass by Value

C# is an object oriented language architected by Microsoft for the development of a variety of applications that run via .NET framework. Like every object oriented language, C# contains objects, methods, variables, properties and events. And similar to other object oriented languages, C# methods take parameters. There are two ways in which parameters can be passed to a method in C#: Pass by value and pass by reference. This article describes basic difference between the two. The article contains basic examples that demonstrate how a parameter is passed by reference and by value, along with their implications.

## C# Pass by Value

Passing parameters to a method by value is simple. That is why this is explained before explaining passing parameters by reference, which is more complex. C# pass by reference has been explained in the next section.

When a simple variable is passed as the parameter to any method, it is passed as a value. This means that the value contained by the variable that is passed as the parameter is copied to the variables of the method, and if inside the method these values are changed or modified, the change is not reflected in the actual passed variable.

Passing variable by value is useful in cases where the actual value of the variable should not be modified by the method and the change is only limited to the called method whereas the value of the variables in the calling method remain unchanged.

The following example demonstrates the concept of passing variable by value in C#.

```
class Program

{

static void Square(int a, int b)

{

a = a * a;

b = b * b;
```

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

}



static void Main(string[] args)

{



int num1 = 5;

int num2 = 10;



Console.WriteLine(num1 +" "+num2);
```

```
Square(num1, num2);

Console.WriteLine(num1 + " " + num2);

Console.ReadLine();

}

}
```

In the above example, the class Program contains two static methods. One is the Main method that is the entry point of the application and the other method is the Square method that takes two parameters and displays the squares of these values.

Inside the main method, two integer type variables num1 and num2 have been declared and assigned values of 5 and 10 respectively. These values have been displayed on the console output.

Next, num1 and num2 have been passed as parameter to Square method. The square method copies the values of num1 and num2 variable into variables in the parameter body i.e ‘a’ and ‘b’ respectively. Now ‘a’ and ‘b’ variables only contain copies of the values of num1 and num2 variables passed to the method. Other than that variables ‘a’ and ‘b’ have no connection with num1 and num2 respectively and if the value of variable ‘a’ and ‘b’ are changed it will have no impact on num1 and num2.

Inside the square method, the square of ‘a’ and ‘b’ values have been squared and result is stored in ‘a’ and ‘b’ variables respectively. These values have then been displayed on the console.

Now, if inside the main method, after calling the Square method, if num1 and num2 variables are displayed, they would contain their original values of 5 and 10.

The output on console would look like this:

5 10

25 10

5 10

**C# Pass by Reference**

Passing a variable to a method by reference is a little bit trickier. The concept of C# pass by reference is that when a parameter is passed to a method by reference, instead of passing value contained by a variable, the reference of the variable is passed to method. The method operates on the references of the variables passed in the parameters rather than operating on their values. This results in the modification of variables in the calling function when they are modified in the called function.

Normally, all the objects are passed by reference as parameter to the method. On the other hand most of the primitive data types such as integer, double, Boolean etc. are passed by value.

The following example demonstrates the concept of C# pass by reference.

```
class Person

{

public int age;

}
```

```
class Program

{

static void Square(Person a, Person b)

{

a.age = a.age * a.age;

b.age = b.age * b.age;

Console.WriteLine(a.age+" "+b.age);

}

static void Main(string[] args)

{

Person p1 = new Person();



Person p2 = new Person();

p1.age = 5;

p2.age = 10;

Console.WriteLine(p1.age +" "+p2.age);

Square(p1, p2);

Console.WriteLine(p1.age + " " + p2.age);

Console.ReadLine();

}
```

```
}
```

In the above code, a class named Person has been declared and it only contains one public member of type integer named age.

Inside the Main method of the Program class, two objects of this Person class have been created and have been named p1 and p2. The member variable age of p1 is assigned a value of 5 and the member variable age of p2 is assigned value 10. These two values have then been displayed on the console.

Next, the two person objects p1 and p2 have been passed as parameters to Square method. This is where real magic begins. What happens here is that instead of passing the values of the members of the p1 and p2 objects, the reference of these objects is passed to the Square method. Those references would be copied to ‘a’ and ‘b’ person type objects in the parameters of the Square method.

Inside the Square method, the age variable of both the a and b objects would be accessed and squared. The resultant values would be again stored in the age variable of both objects and would be displayed on console.

Now, after passing p1 and p2 objects to Square method, if the value of age member variable of these two objects is displayed in the Main method, they would contain the updated value. This is due to the reason that Inside the Square method the member variable ‘age’ has been updated using the reference of the p1 and p2 object that was passed to it.

The output would look like this:

5 10

25 100

25 100

**Conclusion**

Passing by reference is an extremely important feature of C#.NET. It allows modifying multiple values inside a function, which otherwise would return only one value. Apart from that, a complex application might contain multiple classes and methods, it is never advisable to create and destroy objects again and again. The better approach is to create the object of a class and pass it as reference between multiple parts of the application.

## **Partial** classes

**Partial** classes span multiple files. How can you use the partial modifier on a C# class declaration? With partial, you can physically separate a class into multiple files. This is often done by code generators.

**Example.** With normal C# classes, you cannot declare a class in two separate files in the same project. But with the partial modifier, you can. This is useful if one file is commonly edited and the other is machine-generated or rarely edited.

**C# program that uses partial class**

```
class Program
{
    static void Main()
```

```
    {
      A.A1();
      A.A2();
    }
}
```

**Contents of file A1.cs: C#**

```
using System;

partial class A
{
    public static void A1()
    {
      Console.WriteLine("A1");
    }
}
```

**Contents of file A2.cs: C#**

```
using System;

partial class A
{
    public static void A2()
    {
      Console.WriteLine("A2");
    }
}
```

**Output**

```
A1
A2
```

**Partial is required here.** If you remove the partial modifier, you will get an error containing this text: [The namespace '<global

namespace>' already contains a definition for 'A'].

[Namespace]

> **Tip:** To fix this, you can either use the partial keyword, or change one of the class names.

**Compiler.** How does the C# compiler deal with partial classes? If you disassemble the above program, you will see that the files A1.cs and A2.cs are eliminated. You will find that the class A is present.

[IL Disassembler]

> **So:** Class A will contain the methods A1 and A2 in the same code block. The two classes were merged into one.

[Class]

> **Tip:** Partial classes are precisely equivalent to a single class with all the members.

**Compiled result of A1.cs and A2.cs: C#**

```
internal class A
{
    // Methods
```

```
    public static void A1()
    {
      Console.WriteLine("A1");
    }

    public static void A2()
    {
      Console.WriteLine("A2");
    }
}
```

**Summary.** Partial classes can simplify certain C# programming situations. They are often used in Visual Studio when creating Windows Forms programs. The machine-generated C# code is separate.

> **Note:**Partial classes are sometimes used to separate commonly-edited code from rarely-edited code.
>
> > **And:**This can reduce confusion and the possibility that code that isn't supposed to be edited is changed.

UNIT NO-III **Introduction to Web Programming**

## WEB server

A Web server is a program that uses HTTP (Hypertext Transfer Protocol) to serve the files that form Web pages to users, in response to their requests, which are forwarded by their computers' HTTP clients. Dedicated computers and appliances may be referred to as Web servers as well.

The process is an example of the client/server model. All computers that host Web sites must have Web server programs. Leading Web servers include Apache (the most widely-installed Web server), Microsoft's Internet Information Server (IIS) and nginx (pronounced*engine X*) from NGNIX. Other Web servers include Novell's NetWare server, Google Web Server (GWS) and IBM's family of Domino servers.

Web servers often come as part of a larger package of Internet- and intranet-related programs for serving email, downloading requests for File Transfer Protocol (FTP) files, and building and publishing Web pages. Considerations in choosing a Web server include how well it works with the operating system and other servers, its ability to handle server-side programming, security characteristics,

and  the particular publishing, search engine and site building tools that come with it.

# web browser

A browser is software that is used to access the internet. A browser lets you visit websites and do activities within them like login, view multimedia, link from one site to another, visit one page from another, print, send and receive email, among many other activities. The most common browser software titles on the market are: Microsoft Internet Explorer, Google's Chrome, Mozilla Firefox, Apple's Safari, and Opera. Browser availability depends on the operating system your computer is using (for example: Microsoft Windows, Linux, Ubuntu, Mac OS, among others).

## What does browser do

When you type a web page address such as www.allaboutcookies.org into your browser, that web page in its entirety is not actually stored on a server ready and waiting to be delivered. In fact each web page that you request is individually created in response to your request.

You are actually calling up a list of requests to get content from various resource directories or servers on which the content for that page is stored. It is rather like a recipe for a cake - you have a shopping list of ingredients (requests for content) that when

combined in the correct order bakes a cake (the web page).The page maybe made up from content from different sources.Images may come from one server, text content from another, scripts such as date scripts from another and ads from another. As soon as you move to another page, the page that you have just viewed disappears. This is the dynamic nature of websites.

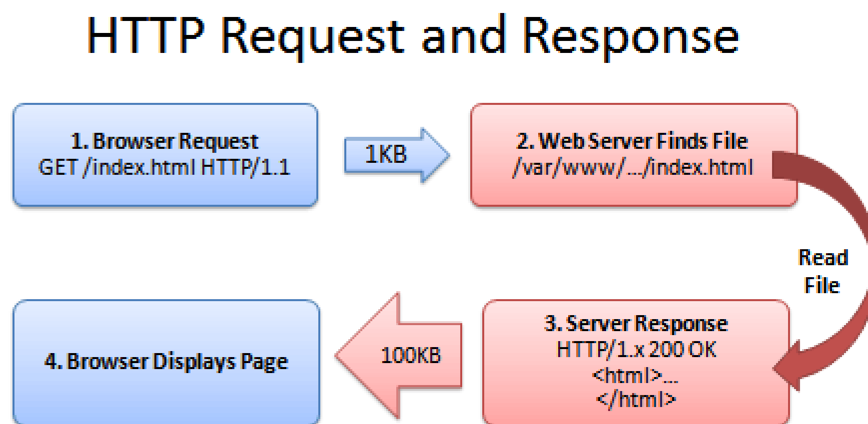
Web browsers and servers communicate via TCP/IP. Hypertext Transfer Protocol (HTTP)is the standard application protocol on top of TCP/IP supporting Web browser requests and server responses. Web browsers also rely on the DNS to work with URLs like 'http://wireless.about.com/'. These protocol standards enable different brands of Web browsers to communicate with different brands of Web servers without requiring special logic for each combination.

A basic Web browsing session works as follows:

- the user specifies a URL in their browser (either from a bookmark or by typing)
- the browser initiates a TCP connection to the Web server (or server pool) via its IP address as published in DNS. (Web servers by default use TCP port 80 to service incoming requests.). As part of this process, the browser also makes DNS lookup requests to convert the URL to an IP address

- after the server completes acknowledgement of its side of the TCP connection, the browser sends HTTP requests to the server to retrieve content for the URL.
- after the server replies with content for the Web page, the browser retrieves the content from the HTTP packets and display it accordingly. Content can include embedded URLs for advertising banners or other third-party content, that in turn triggers the browser to issue new TCP connection requests to those locations. The browser may also save temporary information about its connections to local files on the client computer called*cookies*.

## HTTP REQUEST AND RESPONSE STRUCTURE



**(HTTP - HyperText Transfer Protocol)**

It's a stateless request-response based communication protocol. It's used to send and receive data on the Web i.e., over the Internet. This protocol uses reliable TCP connections either for the transfer of data to and from clients which are Web Browsers in this case. HTTP is a stateless protocol means the HTTP Server doesn't maintain the contextual information about the clients communicating with it and hence we need to maintain sessions in case we need that feature for our Web-applications.

This protocol has three well-known versions so far: HTTP/0.9 being the first version,HTTP/1.0 came next, and now we normally use the HTTP/1.1 version..

As we just saw that HTTP is a request-response based protocol. That means the client will initiate the communication by sending a request (normally called an HTTP Request) and the HTTP Server (or Web Server) will respond back by sending a response (usually called an HTTP Response). Everytime a client needs to send the request, it first establishes a TCP reliable connection with the Web Server and then transfer the request via this connection. The same happens in case a Web Server needs to send back an HTTP Response to a client. Any of the two parties - the client or the server can prematurely stop the transfer by terminating the TCP connection. How a client can terminate the connection is pretty easy to visualize, isn't it? It can be done simply by clicking the 'Stop' button of the browser window (or by closing the browser window itself :-)).

Let's move on to discussing how an HTTP Request or an HTTP Response does look like? Both the Request and the Response have a pre-defined format and it should comply with that so that both the client (the Web Browser) and the server

(HTTP/Web Server) can understand and communicate properly with each other.

## Format of an HTTP Request

It has three main components, which are:-

- **HTTP Request Method, URI, and Protocol Version** - this should always be the first line of an HTTP Request. As it's quite evident from the name itself, it contains the HTTP Request method being used for that particular request, the URI, and the HTTP protocol name with the version being used. It may look like 'GET /servlet/jspName.jsp HTTP/1.1' where the request method being used is 'GET', the URI is '/servlet/jspName.jsp', and the protocol (with version) is 'HTTP/1.1'.
- **HTTP Request Headers** - this section of an HTTP Request contains the request headers, which are used to communicate information about the client environment. Few of these headers are: Content-Type, User-Agent, Accept-Encoding,Content-Length, Accept-Language, Host, etc. Very obvious to understand what info do these headers carry, isn't it? The names are quite self-explanatory.
- **HTTP Request Body** - this part contains the actual request being sent to the HTTP Server. The HTTP Request Header and Body are separated by a blank line (CRLF sequence, where CR means Carriage Return and LF means Line Feed). This blank line is a mandatory part of a valid HTTP Request.

## Format of an HTTP Response

Similar to an HTTP Request, an HTTP Response also has three main

components, which are:-

- **Protocol/Version, Status Code, and its Description** - the very first line of a valid HTTP Response is consists of the protocol name, it's version, status code of the request, and a short description of the status code. A status code of 200means the processing of request was successful and the description in this case will be 'OK'. Similarly, a status code of '404' means the file requested was not found at the HTTP Server at the expected location and the description in this case is 'File Not Found'.
- **HTTP Response Headers** - similar to HTTP Request Headers, HTTP Response Headers also contain useful information. The only difference is that HTTP Request Headers contain information about the environment of the client machine whereas HTTP Response Headers contain information about the environment of the server machine. This is easy to understand as HTTP Requests are formed at the client machine whereas HTTP Responses are formed at the server machine. Few of these HTTP Response headers are: Server, Content-Type, Last-Modified, Content-Length, etc.
- **HTTP Response Body** - this the actual response which is rendered in the client window (the browser window). The content of the body will be HTML code. Similar to HTTP Request, in this case also the Body and the Headers components are separated by a mandatory blank line (CRLF sequence).

# ASP.NET – Introduction

ASP.NET is a web development platform, which provides a programming model, a comprehensive software infrastructure and various services required to build up robust web applications for PC, as well as mobile devices.

ASP.NET works on top of the HTTP protocol, and uses the HTTP commands and policies to set a browser-to-server bilateral communication and cooperation.

ASP.NET is a part of Microsoft .Net platform. ASP.NET applications are compiled codes, written using the extensible and reusable components or objects present in .Net framework. These codes can use the entire hierarchy of classes in .Net framework.

The ASP.NET application codes can be written in any of the following languages:

- C#
- Visual Basic.Net
- Jscript
- J#

ASP.NET is used to produce interactive, data-driven web applications over the internet. It consists of a large number of controls such as text boxes, buttons, and labels for assembling, configuring, and manipulating code to create HTML pages.

## ASP.NET Web Forms Model

ASP.NET web forms extend the event-driven model of interaction to the web applications. The browser submits a web form to the web server and the server returns a full markup page or HTML page in response.

All client side user activities are forwarded to the server for stateful processing. The server processes the output of the client actions and triggers the reactions.

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

- Page state
- Session state

The page state is the state of the client, i.e., the content of various input fields in the web form. The session state is the collective information obtained from various pages the user visited and worked with, i.e., the overall session state. To clear the concept, let us take an example of a shopping cart.

User adds items to a shopping cart. Items are selected from a page, say the items page, and the total collected items and price are shown on a different page, say the cart page. Only HTTP cannot keep track of all the information coming from various pages. ASP.NET session state and server side infrastructure keeps track of the information collected globally over a session.

The ASP.NET runtime carries the page state to and from the server across page requests while generating ASP.NET runtime codes, and incorporates the state of the server side components in hidden fields.

This way, the server becomes aware of the overall application state and operates in a two-tiered connected way.

### The ASP.NET Component Model

The ASP.NET component model provides various building blocks of ASP.NET pages. Basically it is an object model, which describes:

- Server side counterparts of almost all HTML elements or tags, such as <form> and <input>.
- Server controls, which help in developing complex user-interface. For example, the Calendar control or the Gridview control.

ASP.NET is a technology, which works on the .Net framework that contains all web-related functionalities. The .Net framework is made of an object-oriented hierarchy. An ASP.NET web application is made of pages. When a user requests an ASP.NET page, the IIS delegates the processing of the page to the ASP.NET runtime system.

The ASP.NET runtime transforms the .aspx page into an instance of a class, which inherits from the base class page of the .Net framework. Therefore, each ASP.NET page is an object and all its components i.e., the server-side controls are also objects.

# Types of ASP.NET paths

ASP.NET is primarily concerned with "virtual paths", the portion of the path following the hostname or port number. When working with ASP.NET, you must understand the following types of URIs thoroughly, and know how they are handled by ASP.NET and the browser.

- **Absolute paths.** Ex. http://mycomputer/Web1/Test/images/companylogo.png
  - ASP.NET leaves this type of path alone – it's already in the least ambiguous form possible. Browsers understand absolute paths very well.
  - Only use these for referencing external websites. They're expensive to maintain.
- **Root-relative virtual paths.** Ex. /Web1/Test/images/companylogo.png
  - ASP.NET leaves these alone too. Browsers resolve the path client-side by combining it with the domain of the parent document.
  - I don't ever recommend hard-coding these into a website - use application-relative paths or relative paths instead.
  - **Note: These are also called "absolute virtual paths" and "domain-relative paths".**
- **Application-relative paths.** Ex. ~/images/companylogo.png
  - Browsers don't have a clue what the tilde(~) means, so server-side path resolution is required. Server-side, the tilde is shorthand for HttpRuntime.AppDomainAppVirtualPath.
  - ASP.NET rebases these as client-side relative paths on some control attributes, but you must remember to use **runat="server"**.
  - This is the type of path you should use if a relative path doesn't make sense.
- **Relative paths.** Ex: ../images/logo.png
  - There are two types of relative paths: server-side and client-side. They aren't syntactically different, but server-side paths are relative to the containing source file, and client-side paths are relative to the address bar or parent markup file.
  - Server-side relative paths are assumed to be relative to the containing .master, .ascx, or .aspx file location. These must be rebased into client-side relative paths when rendered using ResolveClientUrl(). Most ASP.NET controls do this for you. You should use this type of path

whenever you are referencing a related file that won't move in relation to the current file.

    - Client-side relative paths are relative to the parent URL, usually the address bar. If you want to reference an image on an html page, you must use a path that is relative to the address bar location of the html page. If you want to reference a image from within a .css file, you must use a path that is **relative to the .css file.** Paths inside javascript files are**not** relative to the javascript source location, though. They must be relative to the document the script is executing in, the address bar.

- **Fragment and Javascript paths.** Ex. #section2 or javascript:OpenPopup();
    - ASP.NET leaves these alone. The browser is not supposed to create a new request when one of these is clicked, but to simply perform the action or navigation within the current document.
    - Fragments never appear in a HTTP request. They are only for the browser's benefit, and are stripped off before the path is sent to ASP.NET.

# validation controls

### Why we use validation controls?

Validation is important part of any web application. User's input must always be validated before sending across different layers of the application.

Validation controls are used to:

- Implement presentation logic.
- To validate user input data.
- Data format, data type and data range is used for validation.

### Validation is of two types:

1. Client Side

2. Serve Side

Client side validation is good but we have to be dependent on browser and scripting language support.

Client side validation is considered convenient for users as they get instant feedback. The main advantage is that it prevents a page from being postback to the server until the client validation is executed successfully.

For developer point of view serve side is preferable because it will not fail, it is not dependent on browser and scripting language.

You can use ASP.NET validation, which will ensure client, and server validation. It work on both end; first it will work on client validation and than on server validation. At any cost server validation will work always whether client validation is executed or not. So you have a safety of validation check.

For client script .NET used JavaScript. WebUIValidation.js file is used for client validation by .NET

## Validation Controls in ASP.NET

An important aspect of creating ASP.NET Web pages for user input is to be able to check that the information users enter is valid. ASP.NET provides a set of validation controls that provide an easy-to-use but powerful way to check for errors and, if necessary, display messages to the user.

There are six types of validation controls in ASP.NET

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

The below table describes the controls and their work:

| Validation Control | Description |
| --- | --- |
| RequiredFieldValidation | Makes an input control a required field |
| CompareValidator | Compares the value of one input control to the value of another input control or to a fixed value |
| RangeValidator | Checks that the user enters a value that falls between two values |
| RegularExpressionValidator | Ensures that the value of an input control matches a specified pattern |
| CustomValidator | Allows you to write a method to handle the validation of the value entered |
| ValidationSummary | Displays a report of all validation errors occurred in a Web page |

All validation controls are rendered in form as <span> (label are referred as <span> on client by server)

### Important points for validation controls

- ControlToValidate property is mandatory to all validate controls.
- One validation control will validate only one input control but multiple validate control can be assigned to a input control.

### Validation Properties

Usually, Validation is invoked in response to user actions like clicking submit button or entering data. Suppose, you wish to perform validation on page when user clicks submit button.

Server validation will only performed when CauseValidation is set to true.

When the value of the CausesValidation property is set to true, you can also use the ValidationGroup property to specify the name of the validation group for which the Button control causes validation.

Page has a Validate() method. If it is true this methods is executed. Validate() executes each validation control.

To make this happen, simply set the CauseValidation property to true for submit button as shown below:

```
<asp:Button ID="Button2" runat="server" Text="Submit" CausesValidation=true
/>
```

Lets understand validation controls one by one with practical demonstration:

**RequiredFieldValidation Control**

The RequiredFieldValidator control is simple validation control, which checks to see if the data is entered for the input control. You can have a RequiredFieldValidator control for each form element on which you wish to enforce Mandatory Field rule.

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator3" runat="server" Style
="top: 98px;
left: 367px; position: absolute; height: 26px; width: 162px" ErrorMes
sage="password required"
ControlToValidate="TextBox2"></asp:RequiredFieldValidator>
```

## CompareValidator Control

The CompareValidator control allows you to make comparison to compare data entered in an input control with a constant value or a value in a different control.

It can most commonly be used when you need to confirm password entered by the user at the registration time. The data is always case sensitive.

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator2" runat="server" Style
="top: 145px;
left: 367px; position: absolute; height: 26px; width: 162px" ErrorMes
sage="password required"
ControlToValidate="TextBox3"></asp:RequiredFieldValidator>
```

## RangeValidator Control

The RangeValidator Server Control is another validator control, which checks to see if a control value is within a valid range. The attributes that are necessary to this control are: MaximumValue, MinimumValue, and Type.

```
<asp:RangeValidator ID="RangeValidator1" runat="server" Style="top: 194px; l
eft: 365px;
position: absolute; height: 22px; width: 105px"
ErrorMessage="RangeValidator" ControlToValidate="TextBox4" Maxim
umValue="100"
MinimumValue="18" Type="Integer"></asp:RangeValidator>
```

RegularExpressionValidator Control

A regular expression is a powerful pattern matching language that can be used to identify simple and complex characters sequence that would otherwise require writing code to perform.

Using RegularExpressionValidator server control, you can check a user's input based on a pattern that you define using a regular expression.

It is used to validate complex expressions. These expressions can be phone number, email address, zip code and many more. Using Regular Expression Validator is very simple. Simply set the ValidationExpression property to any type of expression you want and it will validate it.

If you don't find your desired regular expression, you can create your custom one.

In the example I have checked the email id format:

```
<asp:RegularExpressionValidator ID="RegularExpressionValidator1" runat="server" Style="top: 234px;
left: 366px; position: absolute; height: 22px; width: 177px"
ErrorMessage="RegularExpressionValidator" ControlToValidate="TextBox5"
ValidationExpression="₩w+([-+.']₩w+)*@₩w+([-.]₩w+)*₩.₩w+([-.]₩w+)*"></asp:RegularExpressionValidator>
```

The complete code for the above 4 controls is as:

Default.aspx Design

| | | | |
|---|---|---|---|
| Enter your name: | | name is mandatory | |
| Password | | password required | |
| Confirm Password | | password required | CompareValidator |
| Enter your age: | | RangeValidator | |
| Enter your email id: | | RegularExpressionValidator | |

Submit

# Web Form Life Cycle

## Web Form Life Cycle

- Every request for a page made from a web server causes a chain of events at the server. These events, from beginning to end, constitute the life cycle of the page and all its components.
- The life cycle begins with a request for the page, which causes the server to load it. When the request is complete, the page is unloaded.
- From one end of the life cycle to the other, the goal is to render appropriate HTML output back to the requesting browser.
- The life cycle of a page is marked by the following events, each of which you can handle yourself or leave to default handling by the ASP.NET server:

### Initialize

- Initialize is the first phase in the life cycle for any page or control.
- It is here that any settings needed for the duration of the incoming request are initialized.

### Load ViewState

- The ViewState property of the control is populated.
- The ViewState information comes from a hidden variable on the control, used to persist the state across round trips to the server.
- The input string from this hidden variable is parsed by the page framework, and the ViewState property is set.
- This can be modified via the LoadViewState( ) method. This allows ASP.NET to manage the state of your control across page loads so that each control is not reset to its default state each time the page is posted.

### Process Postback Data

- During this phase, the data sent to the server in the posting is processed.
- If any of this data results in a requirement to update the ViewState, that update is performed via the LoadPostData( ) method.

### Load

- CreateChildControls( ) is called, if necessary, to create and initialize server controls in the control tree.
- State is restored, and the form controls show client-side data. The load phase can be modified by handling the Load event with the OnLoad method.

### Send Postback Change Modifications

- If there are any state changes between the current state and the previous state, change events are raised via the RaisePostDataChangedEvent( ) method.

### Handle Postback Events

- The client-side event that caused the postback is handled.

### PreRender

- This is the phase just before the output is rendered to the browser.
- It is essentially the last chance to modify the output prior to rendering using the OnPreRender( ) method.

### Save State

- Near the beginning of the life cycle, the persisted view state was loaded from the hidden variable.
- Now it is saved back to the hidden variable, persisting as a string object that will complete the round trip to the client.

- This can be overridden by using the SaveViewState( ) method.

Render

- This is where the output to be sent back to the client browser is generated.
- This can be overridden by using the Render method.
- CreateChildControls( ) is called, if necessary, to create and initialize server controls in the control tree.

Dispose

- This is the last phase of the life cycle. It gives you an opportunity to do any final cleanup and release references to any expensive resources, such as database connections.
- This can be modified by using the Dispose( ) method.

## Response.Redirect, Server.Response

Both Response.Redirectand Server.Transfermethods are used to transfer a user from one web page to another web page. Both methods are used for the same purpose, but still there are some differences as follows.

The Response.Redirectmethod redirects a request to a new URL and specifies the new URL while theServer.Transfermethod for the current request, terminates execution of the current page and starts execution of a new page using the specified URL path of the page.

Both Response.Redirectand Server.Transferhave the same syntax like:

```
Response.Redirect("UserDetail.aspx");
Server.Transfer("UserDetail.aspx");
```

ASP.Net Technology both "Server" and "Response" are objects of ASP.Net. Server.Transfer and Response.Redirect both are used to transfer a user from one page to another. But there is some remarkable differences between both the objects which are as follow.

### Response.Redirect

1. Response.Redirect() will send you to a new page, update the address bar and add it to the Browser History. On your browser you can click back.
2. It redirects the request to some plain HTML pages on our server or to some other web server.
3. It causes additional roundtrips to the server on each request.
4. It doesn't preserve Query String and Form Variables from the original request.
5. It enables to see the new redirected URL where it is redirected in the browser (and be able to bookmark it if it's necessary).
6. Response. Redirect simply sends a message down to the (HTTP 302) browser.

### Server.Transfer

1. Server.Transfer() does not change the address bar, we cannot hit back.One should use Server.Transfer() when he/she doesn't want the user to see where he is going. Sometime on a "loading" type page.
2. It transfers current page request to another .aspx page on the same server.
3. It preserves server resources and avoids the unnecessary roundtrips to the server.
4. It preserves Query String and Form Variables (optionally).
5. It doesn't show the real URL where it redirects the request in the users Web Browser.
6. Server.Transfer happens without the browser knowing anything, the browser request a page, but the server returns the content of another.

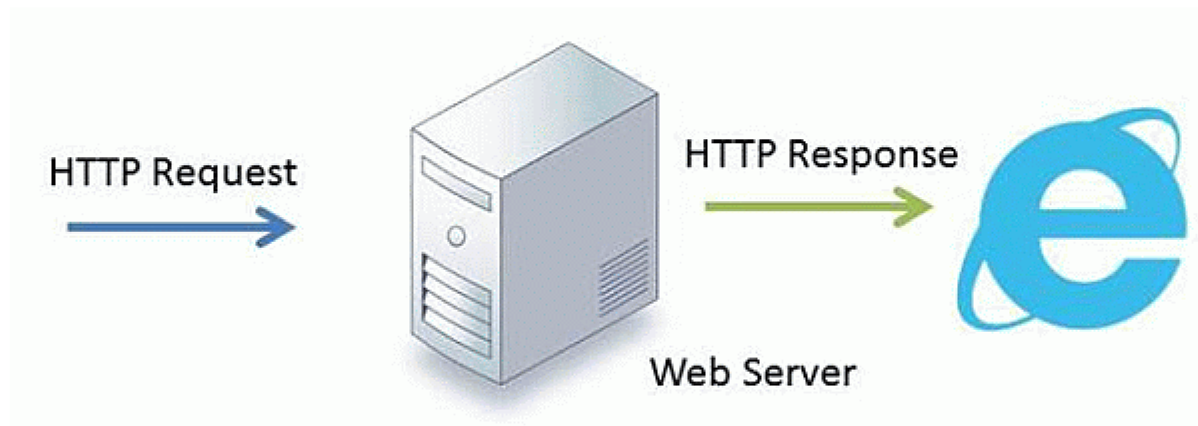



***Figure 1.4 Server.Transfer method request and response***

# postback property of button

Postback is actually sending all the information from client to web server, then web server process all those contents and returns back to the client. Most of the time ASP control will cause a post back (e. g. buttonclick) but some don't unless you tell them to do In certain events ( Listbox Index Changed,RadioButton Checked etc..) in an ASP.NET page upon which a PostBack might be needed.

How to ispostback in asp.net
IsPostBack is a property of the Asp.Net page that tells whether or not the page is on its initial load or if a user has perform a button on your web page that has caused the page to post back to itself. The value of the Page.IsPostBack property will be set to true when the page is executing after a postback, and false otherwise. We can check the value of this property based on the value and we can populate the controls on the page.

Is Postback is normally used on page _load event to detect if the web page is getting generated due to postback requested by a control on the page or if the page is getting loaded for the first time.

# ASP.NET state management

Web Pages developed in ASP.Net are HTTP based and HTTP protocol is a stateless protocol. It means that web server does not have any idea about the requests from where they coming i.e from same client or new clients. On each request web pages are created and destroyed.

So, how do we make web pages in ASP.Net which will remember about the user, would be able to distinguish b/w old clients(requests) and new clients(requests) and users previous filled information while navigating to other web pages in web site?

Solution of the above problem lies in State Management.

ASP.Net technology offers following state management techniques.

**Client side State Management**

- Cookies
- Hidden Fields
- View State
- Query String

**Server side State Management**

- Session State
- Application State

These state management techniques can be understood and by following simple examples and illustrations of the each techniques.

## Client Side State Management

### Cookies

A cookie is a small amount of data which is either stored at client side in text file or in memory of the client browser session. Cookies are always sent with the request to the web server and information can be retrieved from the cookies at the web server. In ASP.Net, HttpRequest object contains cookies collection which is nothing but list of HttpCookie objects. Cookies are generally used for tracking the user/request in ASP.Net for example, ASP.Net internally uses cookie to store session identifier to know whether request is coming from same client or not. We can also store some information like user identifier (UserName/Nick Name etc) in the cookies and retrieve them when any request is made to the web server as described in following example. It should be noted that cookies are generally used for storing only small amount of data(i.e 1-10 KB).

### Hidden Fields

A Hidden control is the control which does not render anything on the web page at client browser but can be used to store some information on the web page which can be used on the page.

HTML input control offers hidden type of control by specifying type as "hidden". Hidden control behaves like a normal control except that it is not rendered on the page. Its properties can be specified in a similar manner as you specify properties for other controls. This control will be posted to server in HttpControl collection whenever web form/page is posted to server. Any page specific information can be stored in the hidden field by specifying value property of the control.

## View State/Control State

ASP.Net technology provides View State/Control State feature to the web forms. View State is used to remember controls state when page is posted back to server. ASP.Net stores view state on client site in hidden field __ViewState in encrypted form. When page is created on web sever this hidden control is populate with state of the controls and when page is posted back to server this information is retrieved and assigned to controls. You can look at this field by looking at the source of the page (i.e by right clicking on page and selecting view source option.)

You do not need to worry about this as this is automatically handled by ASP.Net. You can enable and disable view state behaviour of page and its control by specifying 'enableViewState' property to true and false. You can also store custom information in the view state as described in following code sample. This information can be used in round trips to the web server.

## Query String

Query string is the limited way to pass information to the web server while navigating from one page to another page. This information is passed in url of the request.  Following is an example of retrieving information from the query strings.

## Server Side State Management

## Session State

Session state is used to store and retrieve information about the user as user navigates from one page to another page in ASP.Net web application. Session state is maintained per user basis in ASPNet runtime. It can be of two types in-memory and out of memory. In most of the cases small web applications in-

memory session state is used. Out of process session state management technique is used for the high traffic web applications or large applications. It can be configured  with some configuration settings in web.conig file to store state information in ASPNetState.exe (windows service exposed in .Net or on SQL server.

## Application State

Application State is used to store information which is shared among users of the ASP.Net web application. Application state is stored in the memory of the windows process which is processing user requests on the web server. Application state is useful in storing small amount of often-used data. If application state is used for such data instead of frequent trips to database, then it increases the response time/performance of the web application.

In ASP.Net, application state is an instance of HttpApplicationState class and it exposes key-value pairs to store information. Its instance is automatically created when a first request is made to web application by any user and same state object is being shared across all subsequent users.

Application state can be used in similar manner as session state but it should be noted that many user might be accessing application state simultaneously so any call to application state object needs to be thread safe. This can be easily achieved in ASP.Net by using lock keyword on the statements which are accessing application state object. This lock keyword places a mutually exclusive lock on the statements and only allows a single thread to access the application state at a time. Following is an example of using application state in an application.

### What is Web.Config File?

Configuration file is used to manage various settings that define a website. The settings are stored in XML files that are separate from your application code. In this way you can configure settings independently from your code. Generally a website contains a single Web.config file stored inside the application root directory. However there can be many configuration files that manage settings at various levels within an application.

## What Web.config file contains?

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

- Database connections
- Caching settings
- Session States
- Error Handling
- Security

### Different types of Configuration files

**Machine.config:** Server or machine-wide configuration file

**Web.config:** Application configuration files which deal with a single application

### Machine.config File

Configuration files are applied to an executing site based on a hierarchy. There is a global configuration file for all sites in a given machine which is called Machine.config. This file is typically found in the C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\CONFIG directory.

The Machine.config file contains settings for all sites running on the machine provided another .config further up the chain does not override any of these

settings. Although Machine.config provides a global configuration option, you can use .config files inside individual website directories to provide more granular control. Between these two poles you can set a number of other .config files with varying degree of applicable scope.

**Application Configuration file (Web.config)**

Each and Every ASP.NET application has its own copy of configuration settings stored in a file called Web.config. If the web application spans multiple folders, each sub folder has its own Web.config file that inherits or overrides the parent's file settings.

# Application Domains in C#

In .NET, each application runs in an application domain under the control of a host. The host creates the application domain and loads assemblies into it. The host has access to information about the code via evidence. This information can include the zone in which the code originates or the digital signatures of the assemblies in the application domain. The System.AppDomain class provides the application domain functionality and is used by hosts. A host can be trusted if it provides the CLR with all the evidence the security policy requires.

There are several types of application hosts:

- Browser host-includes applications hosted by Microsoft Internet Explorer; runs code within the context of a Web site.
- Server host-regarding ASP.NET, refers to the host that runs the code that handles requests submitted to a server.
- Shell host-refers to a host that launches applications, namely .exe files, from the operating system shell.
- Custom-designed host-a host that creates domains or loads assemblies into domains (e.g., dynamic assemblies).


- Browser host-includes applications hosted by Microsoft Internet Explorer; runs code within the context of a Web site.
- Server host-regarding ASP.NET, refers to the host that runs the code that handles requests submitted to a server.
- Shell host-refers to a host that launches applications, namely .exe files, from the operating system shell.
- Custom-designed host-a host that creates domains or loads assemblies into domains (e.g., dynamic assemblies).

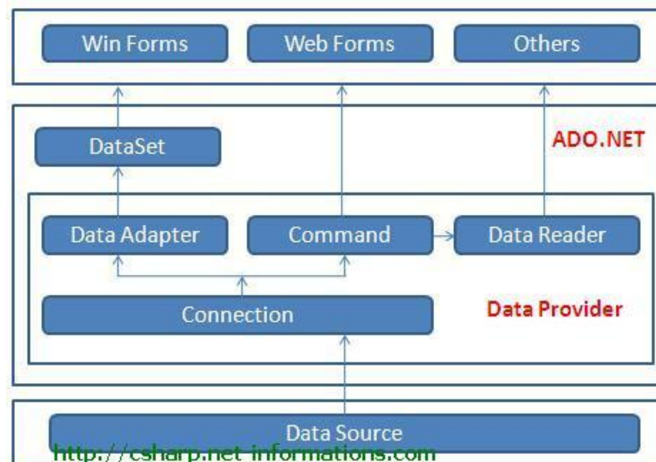
# ADO.NET

## INTRODUCTION TO ADO.NET

ADO.NET is a set of computer software components that programmers can use to access data and data services based on disconnected DataSets and XML. It is a part of the base class library that is included with the Microsoft .NET Framework. It is commonly used by programmers to access and modify data stored in relational database systems, though it can also access data in non-relational sources. ADO.NET is sometimes considered an evolution of ActiveX Data Objects (ADO) technology, but was changed so extensively that it can be considered an entirely new product.

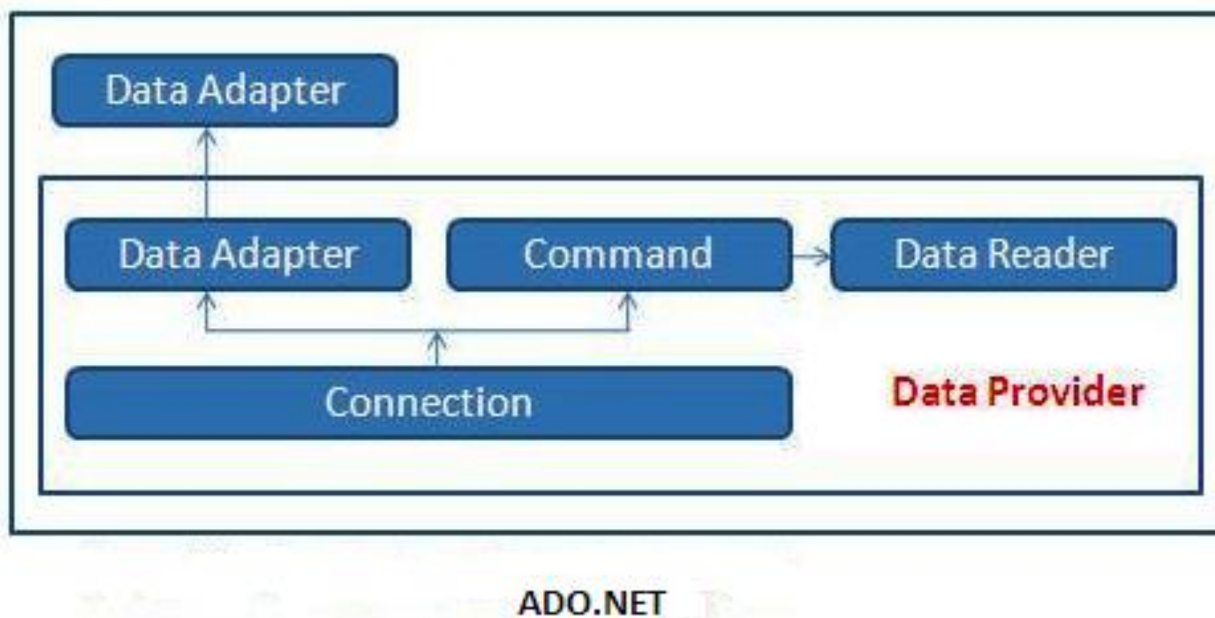
## ARCHITECTURE OF ADO.NET

**ADO.NET**



ADO.NET is a data access technology from Microsoft .Net Framework , which provides communication between relational and non-relational systems through a common set of components .ADO.NET consist of a set of Objects that expose data access services to the .NET environment. ADO.NET is designed to be easy to use, and Visual Studio provides several wizards and other features that you can use to generate ADO.NET data access code.

Data Providers and DataSet



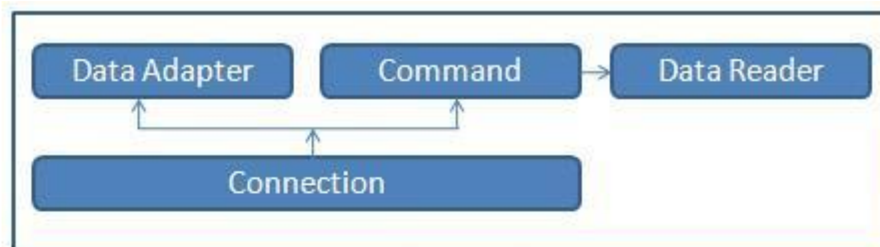

The two key components of ADO.NET are Data Providers andDataSet . The .Net Framework includes mainly three Data Providers for ADO.NET. They are the Microsoft SQL Server Data Provider , OLEDB Data Provider and ODBC Data Provider . SQL Server uses the SqlConnection object , OLEDB uses the

OleDbConnection Object and ODBC uses OdbcConnection Object respectively.

C# SQL Server Connection

C# OLEDB Connection

C# ODBC Connection




http://csharp.net-informations.com

Data Provider

The four Objects from the [.Net Framework] provides the functionality of Data Providers in the ADO.NET. They are**Connection** Object, **Command** Object , **DataReader** Object and**DataAdapter** Object. The Connection Object provides physical connection to the Data Source. The Command Object uses to perform SQL statement or stored procedure to be executed at the Data Source. The DataReader Object is a stream-based , forward-only, read-only retrieval of query results from the Data Source, which do not update the data. Finally the DataAdapter Object , which populate a Dataset Object with results from a Data Source .

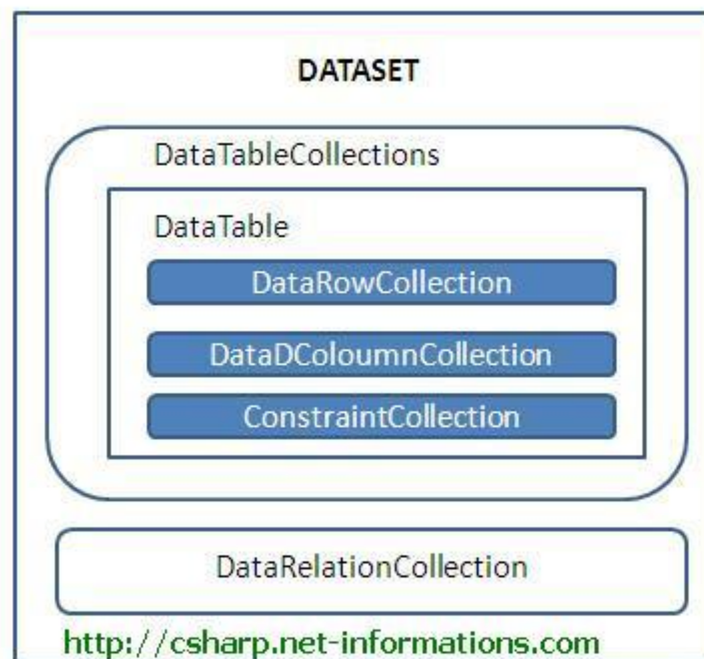
C# Connection

C# Command

C# DataReader

C# DataAdapter

DataSet




DataSet provides a disconnected representation of result sets from the Data Source, and it is completely independent from the Data Source. DataSet provides much greater flexibility when dealing with related Result Sets.

DataSet consists of a collection of DataTable objects that you can relate to each other with DataRelation objects. The DataTable

contains a collection of DataRow and DataCoulumn Object which contains Data. The DataAdapter Object provides a bridge between the DataSet and the Data Source. From the following section you can see each of the ADO.NET components in details with C# Source Code .

# Connected and Disconnected Data Access Architecture

The ADO.NET Framework supports two models of Data Access Architecture, Connection Oriented Data Access Architecture and Disconnected Data Access Architecture.

In Connection Oriented Data Access Architecture the application makes a connection to the Data Source and then interact with it through SQL requests using the same connection. In these cases the application stays connected to the database system even when it is not using any Database Operations.

ADO.Net solves this problem by introduces a new component called Dataset. The DataSet is the central component in the ADO.NET Disconnected Data Access Architecture. A DataSet is an in-memory data store that can hold multiple tables at the same time. DataSets only hold data and do not interact with a Data Source. One of the key

characteristics of the DataSet is that it has no knowledge of the underlying Data Source that might have been used to populate it.

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

In Connection Oriented Data Access, when you read data from a database by using a DataReader object, an open connection must be maintained between your application and the Data Source. Unlike the DataReader, the DataSet is not connected directly to a Data Source through a Connection object when you populate it. It is the DataAdapter that manages connections between Data Source and Dataset by fill the data from Data Source to the Dataset and giving a disconnected behavior to the Dataset. The DataAdapter acts as a bridge between the Connected and Disconnected Objects.

```
SqlDataAdapter adapter = new SqlDataAdapter("sql", "connection");
DataSet ds = new DataSet();

adapter.Fill(ds, "Src Table");
```

By keeping connections open for only a minimum period of time, ADO .NET conserves system resources and provides maximum security for databases and also has less impact on system performance.