UNIT - I

.NET FRAMEWORK & FUNDAMENTALS

WHAT IS .NET FRAMEWORK :-

- ★ .Net Framework is a Software Framework developed by Microsoft.
- ★ Collection of Programming Execution Environment
- ★ Allows developer to develop, run, deploy the application

eg ★ Windows application
★ Web appln
★ Mobile appln
★ ~~Desktop~~ gaming appln. & IO.T etc..,

→ .Net Cost is Free Product.
→ It is a Open Source Platform (Source Code)
→ It is a Cross Platform (Multiple used lang.)
→ Support C#, VB, F#
→ Version 1.0 - 4.8

Two Product
★ .NET Framework (windows) 1.0 - 4.8
★ .NET Core. (windows, MAC, Linux) 2.2

OVERVIEW OF THE .NET FRAMEWORK:-

The .NET framework is designed as an

Integrated environment.

Developing and running appln on the Internet

On Desktop/windows and Mobile devices.

Primary Objectives:

→ To Provide a Consistent object-oriented environment across the range of appln.

→ To Provide a portable environment that can be hosted by an OS, Already C# & Major Part of the .NET runtime, CLI Common Language Infrafstructure (CLI)

→ To Provide a Managed environment in which code is easily Verified for Safe Execution.

ARCHITECTURE OF .NET FRAMEWORK

.Net Framework designers settled on architecture that separates in two Parts:-

→ Common Language Runtime (CLR)

→ Framework class Library (FCL)

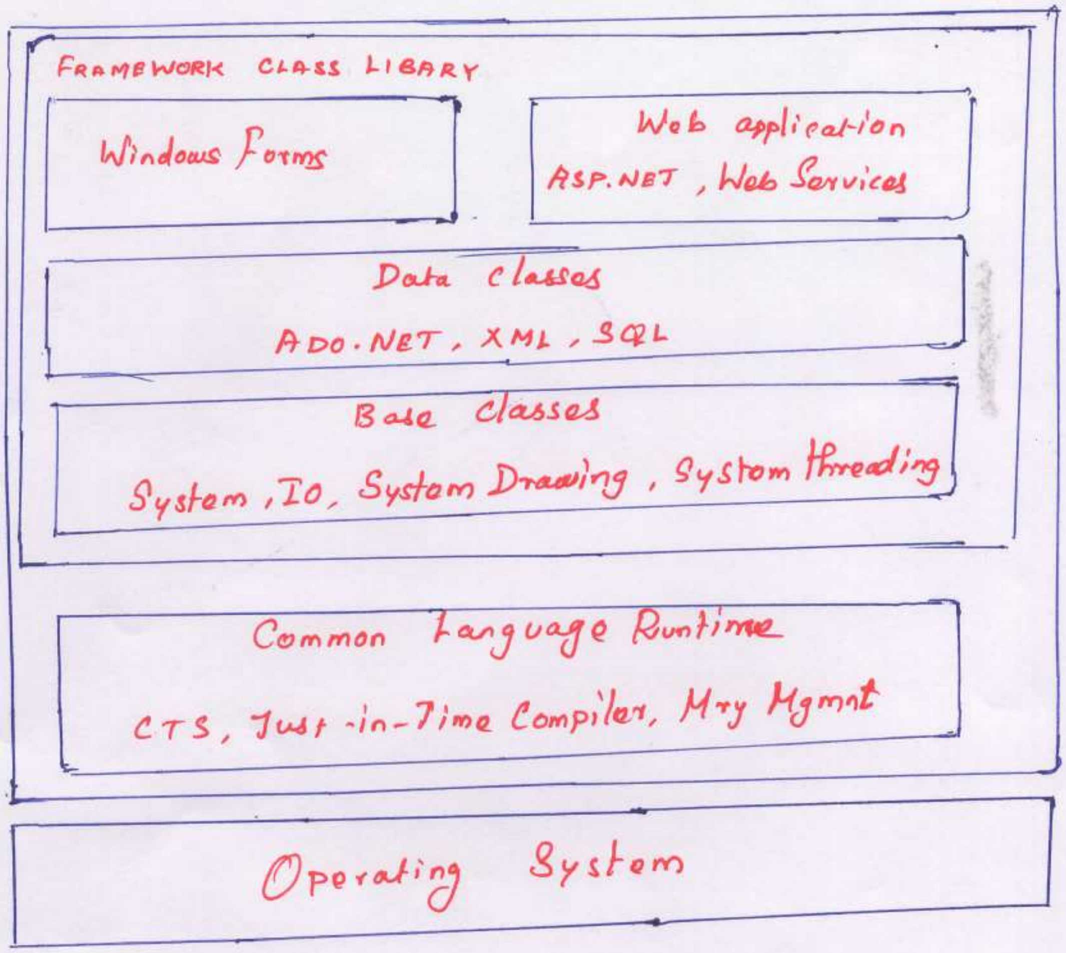


Fig:- .Net FrameWork.

The CLR - which is implementation of the CLI

- Handles to Code Execution and all of the tasks Compilation, Mry Mgmnt , Security & thread Mgmnt Safty to use.

Code run under the CLR is referred to as Managed Code.

Un managed code that does not implement The requirements to run CLR

Microsoft .NET and the CLI Standards.

.NET a MicroSoft Product ~~is~~ tethered Only to the Windows Operating System

It is a Portable runtime and development Platform that will be implemented on Multiple OS.

CLI define a Platform-independent Virtual Code. Execution Environment.

It Specifies no Operating System, so it Could Just as Easily be Linux as Windows.

The Centerpiece of the standard is the definition for a CIL (common ~~Language~~ Intermediate Language must be produced by any Compliant language

- It defines the data types Supported by any Compliant Language.

This intermediate code is Compiled into the native language of its host Operating System.

Architecture of CLI Specification:-

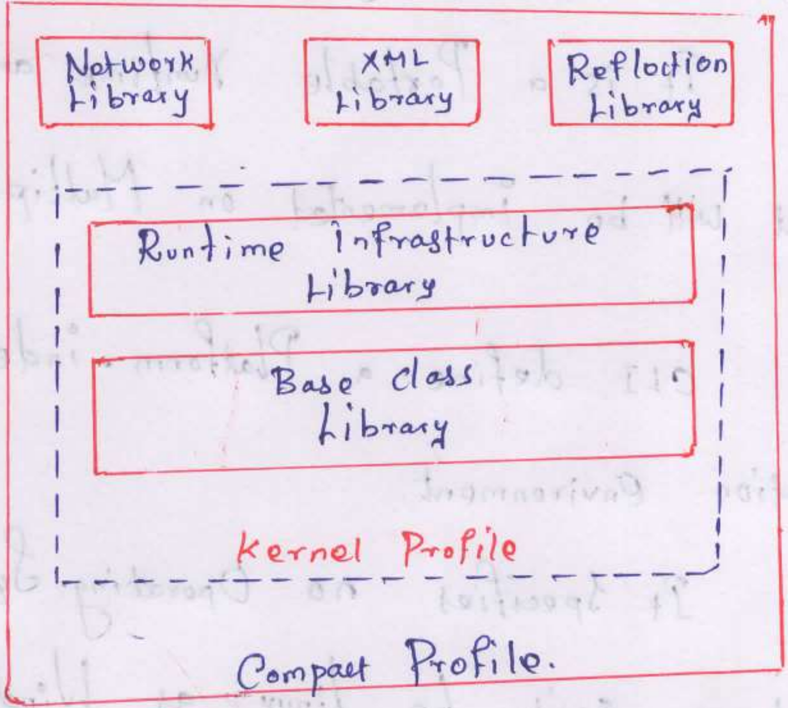


Fig: Architecture defined by CLI Specification.

CLI define two implementations:

- * Kernel Profile
- * Compact Profile.

Kernel Profile:- Minimal implementation is Known as Kernel Profile. It Contains the types and classes required by a Compiler that is CLI Compliant.

Compact Profile:-

It is more feature rich Compact Profile.

→ adds three class Libraries

- ★ XML Library (simple XML Parsing)
- ★ Network Library (HTTP Support and access ports)
- ★ Reflection Library (reflection a way for a Pgm to examine itself through metacode)

It would be considerably shorter, if it is described only the CLI recommendations.

ADO.NET (database classes) } or Windows Forms and
ASP.NET (Web classes) } the XML would be greatly reduced.

These libraries depend on the underlying Windows API for functionality

.NET Permits a Program to invoke the WIN 32 API using an Interop feature.

This means that a .NET developer has access not only to the WIN32 API but also Legacy applns and Components (COM).

1.2 Common Language Runtime

To Manage the entire life cycle of an application

It locates the code → Compiles it → Load associated classes, Manages its execution → and Ensures the Memory Mgmt.

It Supports cross-language integration to Permit Code generated by dlf Language interact

In this Section peers into the Inner Workings th CLR

Comportable with terminology

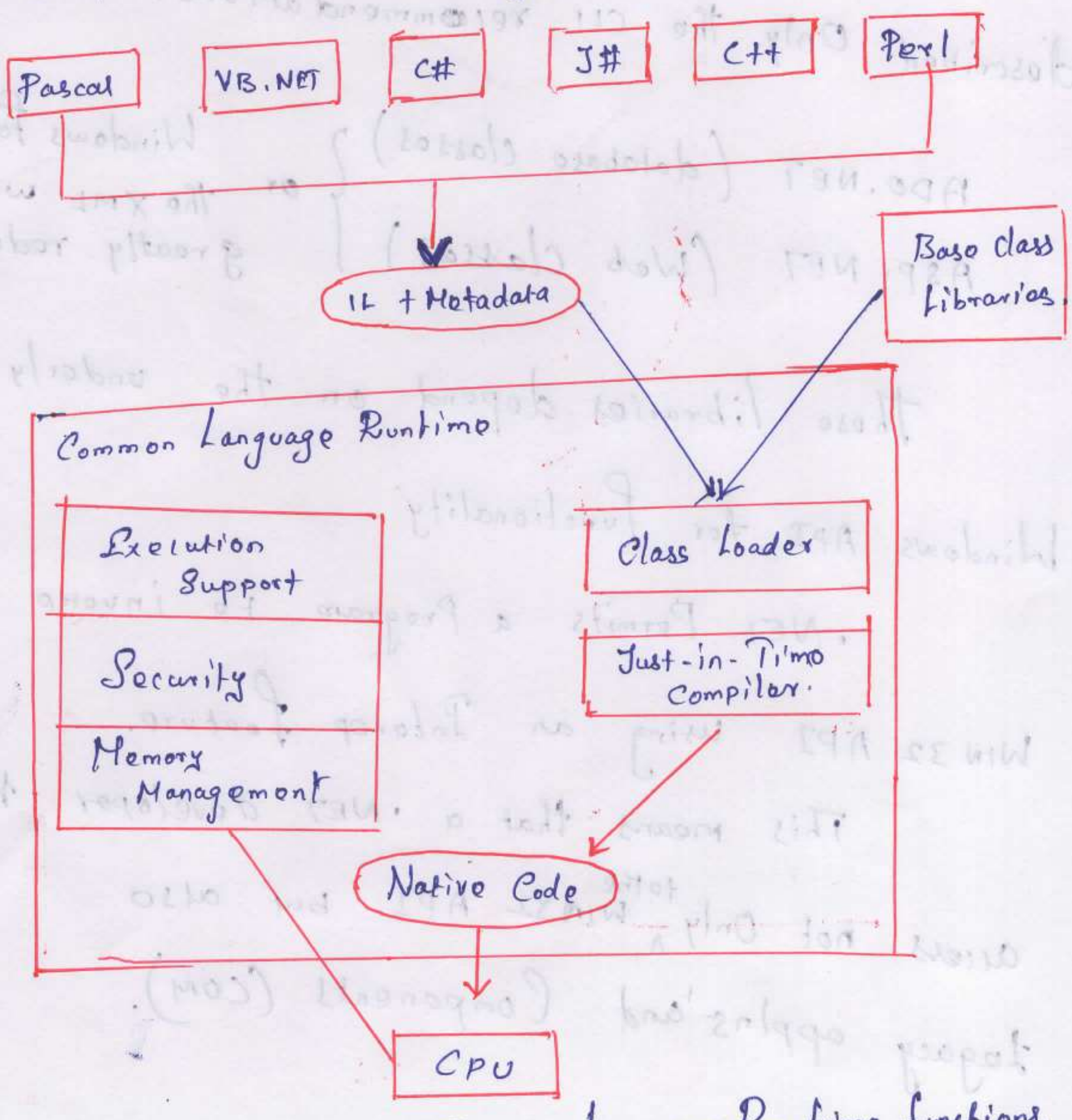


Fig: Common language Run-time functions.

Compiling .NET Code :-

Compilers that are Compliant with CLR - Pf generate the code targeted for the run time this Code known variously as Common Intermediate Language (CIL) Intermediate Language (IL) (or) MSIL Microsoft IL

Assembler type language that is Packaged EXE or DLL file.

Note that ~~not~~ These are not Standard executable files and require that the runtime's JIT (Just-in-Time)

That Compiler Convert into (IL) them to M/c Specific code the code known as Managed code.

.NET Framework formal obj of Lang Compatibility CL Lang Created it, It's interaction is with the Language independent, Bcz appln Communicate throug IL

Another .NET Goal platform portability is addressed localizing the Creation of M/c code in the JIT Compiler

IL produced on One Platform can be run on any other Platform that has it's own framework JIT Compiler that emits it's own m/c code.

Compilors that target the CLR must omit metadata into every code module.

The Metadata is a Set of tables that allows Each code module to be Self - descriptive. the tables Contain Information about the assembly Containing the Code. as well as full description of the Code itself.

Metadatas uses:-

-> Most important use is by the JIT Compiler -> which gathers all the type information it needs for Compiling directly from the meta code.

It is also uses this information for Code Verification to ensure Program performs Correct Operations

-> Metadata is used the Garbage Collection Process (Mry Mgmnt), Garbage Collector (GC) GC can determine what obj can and Can't have their memory reclaimed.

CLI defines a formal Specification called the Common Type System (CTS), which is an integrated part of the CLR.

Common Type System :-

The CTS provides a base Set of data types For Each languages that runs on the .NET Platform.

It Specifies how to declare and Create Custom types and how to Manage the lifetime of instances of these types.

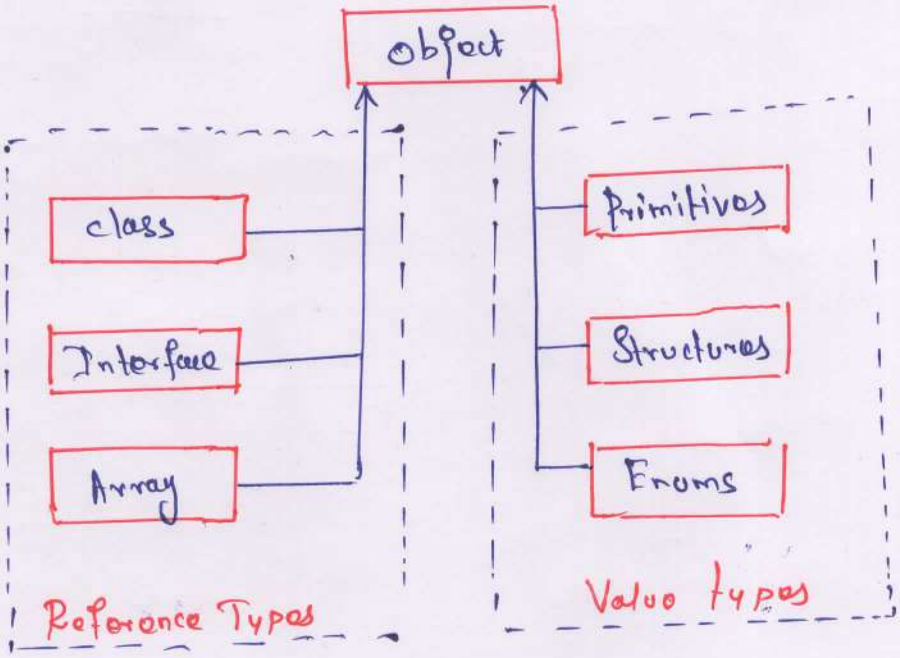


Fig:- Base Types defined by Common Type System.

Common Type System :-

The CTS provides a base set of data types for Each languages that runs on the .NET Platform.

It Specifies how to declare and Create Custom types and how to Manage the lifetime of instances of these types.

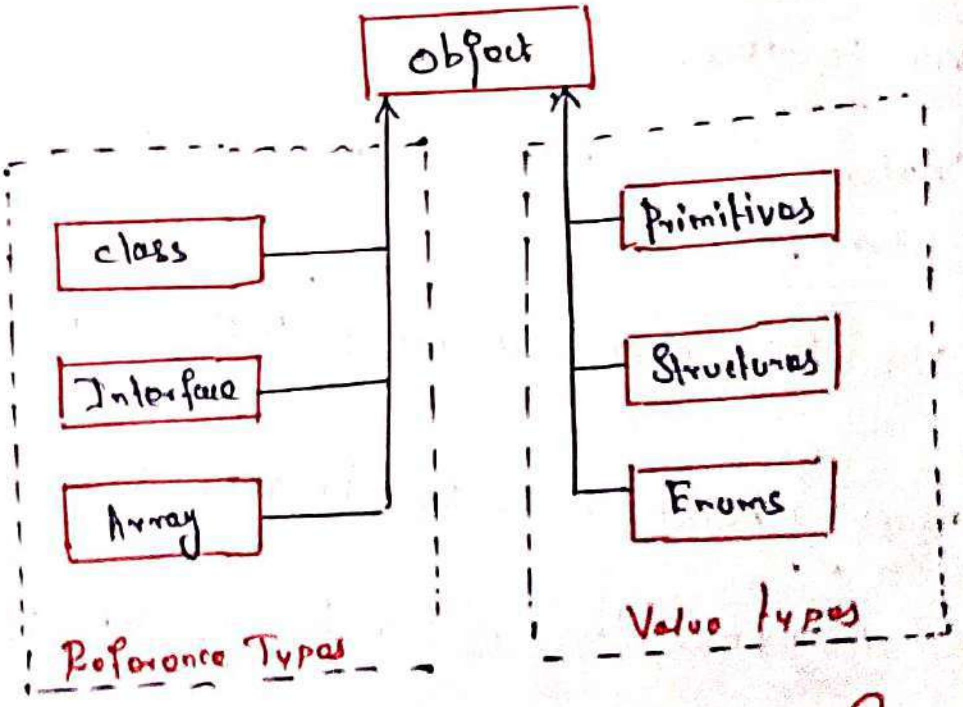


Fig:- Base Types defined by Common Type System.

Two things of types :-

→ reference types

→ Value types

This taxonomy is based on how the types are Stored and accessed in memory.

reference types are accessed in a Special Memory area via Pointers.

Value types are referenced directly in a Program Stack.

· Other thing Note · all types both custom and .NET defined must inherit from the Predefined System.object type.

that all types Support a basic Set of Inherited methods and Properties.

~~CTS~~ ~~Specfn~~ (i.e) types Can be hosted by CLR

This alone does not gurantee that the language Can communicate with Other language.

There is more restrictive Set of Specfn called CLS - Common Language Specification.

CLS Features and Rules:-

→ Visibility (Scope) - The Rules apply Only to those members of type that are available.

→ Character and Casting - For two Variables to be Considered

→ Primitive types - The following Primitive data types are CLS compliant:
Byte, Int16, Int32, Int64, Single, Double
Boolean, Char, Decimal, IntPtr and String

→ Constructor invocation - A Constructor must call the base class's Constructor b4 it can access any of it's instance of data.

→ Array bounds - All dimensions of arrays must have a lower bound of Zero (0)

→ Enumerations - The Underlying type of an enumeration (enum) - type Byte, Int16, Int32, (or) Int64

→ Method Signature - All return and Parameter types used in a type (or) Member Signature, must be CLS Compliant.

Assemblies:-

All of the Managed Code that runs in .NET must be Contained in an assembly.

Logically, the assembly is referenced as One EXE or DLL File.

physically, it May Consist of a Collection Of One or more files that Contain code (or) resources Such as images or XmL data.

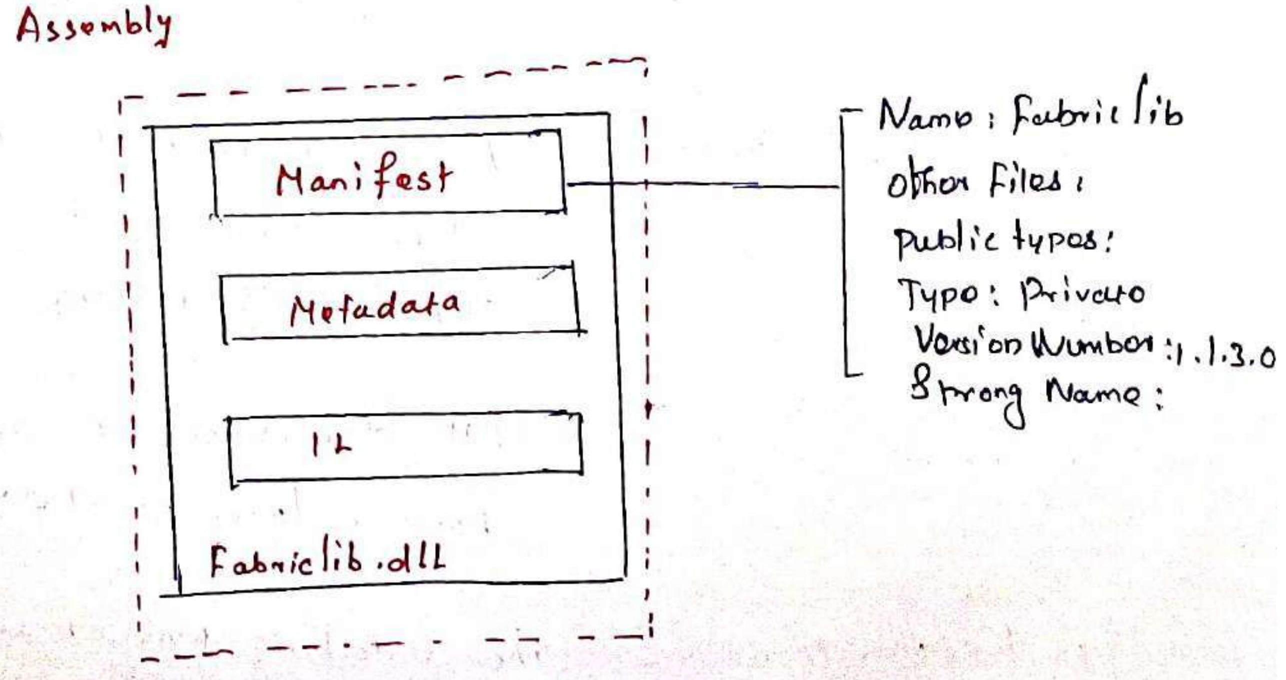


Fig 1.5 Single file assembly

An assembly is Created when a .NET Compatible Compiler converts a file Containing Source code int a DLL, or EXE File.

Manifest.

Each assembly must have one file that contains a manifest.

The manifest is a Set of tables Containing metadata that lists the name of all files in the assembly, references to external assemblies, and information such as name and version that identify the assembly.

Strongly named assemblies (discussed later)

↳ it also include a Unique digital Signature

↳ when an assembly loaded, the CLR's first order of business open the file Containing the manifest so it can identify the members of the assembly.

Metadata:-

In addition to the Manifest tables just described. C# Compiler produces definition and reference tables.

The definition tables provide Complete description of the types Contained in the IL

IL:

Intermediate language. by the CLR Can use IL - it must be packaged in an EXE (or) DLL assembly the two are not identical.

MULTI-FILE ASSEMBLY:-

- An assembly Contain Multiple Files.

- These Files are not restricted to Code modules. but May be resources ~~that~~ files such as graphic images & text Files.

- There is no limit to the number of files in the assembly

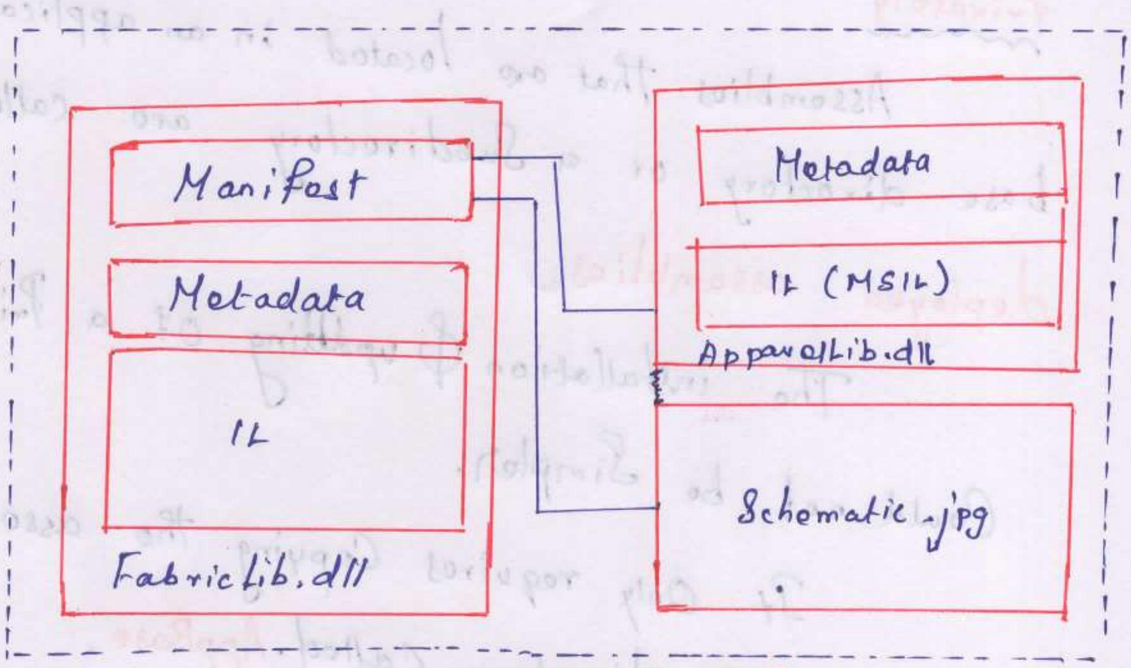


Fig: Multi-File assembly

In the assembly Manifest Contains the information that identifies all files in the assembly.

Advantages:-

→ They allow to Combine modules created in d/f pgm Lang

A Pgming may relay VB.NET for its RAD (Rapid Appln Dvlpmnt)

→ To Optimize how code is Loaded into CLR

Frequently used Code should placed in one module. infrequently used Code in another. CLR does not load the modules Untill they are needed. If Creating a class Library. go a step further.

Private and Shared Assemblies:

Assemblies May be deployed in two Ways:
→ Privately
→ globally

Privately

Assemblies that are located in an application's base directory or a Subdirectory are called Privately deployed assemblies.

The installation & updating of a Private assembly Could not be Simpler.

It Only requires Copying the assembly's ~~files~~ int to ~~be~~ ~~moved~~ directory Called. AppBase.

Globally :-

A Shared assembly is One installed in a global location. Called Global Assembly Cache (GAC)

It is accessible by Multiple appln. GAC permits and Multiple appln Execute Side-by-Side.

To Support this .NET Overcomes the name Conflict problem that plagues DLLs by using

Four attributes to identify an assembly:

→ The file name - Assembly - referred - friedly nam

→ Culture identity . assembly may be associated with a Particular ~~Culture~~ or Lang

English _ en
French - fr

→ a Version number. - Every assembly has a Version no. to all files

→ public key token

↳ shared assembly is Unique & authentic .NET require.

eg

Assembly Name	Version	Culture	public keyToken.
Accessibility	2.0.36000		b03f5f711d50a3a
ADODB	7.0.3300.0		b03f5f7f13d50a3a.
apphost	2.0.3620.0		692fbea5521e1304.

—xxx—

WORKING WITH THE .NET Framework and SDK:-

→ It Contains The tools, Compilers and documentation

→ It required to Create a S/W that will run on any M/c that has the .NET Framework Installed.

→ Free download (100 Mb) - Win XP, 2000, 7 8, 10 and Subsequent Windows OS.

→ client using s/w developed with the SDK do not require the SDK. on their Machine.

→ do require a Compatible Version of .NET framework.

→ This .NET Framework Redistributable is available as a free download³ (20+ MB)

→ .NET appln will run identically on all OS Platform

Updating the .NET framework :-

→ If Many development Environments,

→ Installing a new Version of the framework is almost effortless.

\winnt\Microsoft.NET\Framework\v1.0.3705

. v1.1.4322

\winnt\Microsoft.NET\Framework\V2.0.40607.

→ The installation of any new s/w raises the question of Compatibility with Version appln developed using an Older Version.

→ The Key to this is the application Configuration File.

.NET Framework Tools :-

The .NET Framework automates as Many tasks as Possible and Usually hides the details from the developer. However, there are times when manual intervention is required.

→ Add a file to an assembly

→ View the Contents of an assembly

→ View the details of a Specific class.

→ Generate a public/private key pair in Order to create a Strongly named assembly.

→ Edit Configuration Files.

few, Such as those for Exploring Classess and assemblies, Should be Mastered early in the .NET Curve.

Selected .NET Framework Tools:-

AL.exe [Assembly Linker] : It can be used for creating an assembly Composed of modules from d/f Compilers.
It is also used to build resource-Only assemblies.

FusLogvw.exe [Assembly Binding Log Viewer] : used to troubleshoot assembly loading process

ILdasm.exe [MSIL Disassembler] : A tool for exploring an assembly, ~~is~~ IL and Metadata.

MsCorCfg.msc [.NET Framework Configuration tool] : A Microsoft Mgmnt Console (MMC) Snap-in used to Configure an assembly while avoiding direct Manual Changes to an appln Configuration fools.
Available for individual pgmr's

Ngen.exe [Native Image Generator] : Compiles an assembly's IL into native m/c code.
This img is then placed in the native image cache.

Sn.exe [Strong Name tool : Generates the keys that are used to create a Strong - or Signed - assembly.

Framework Configuration Tool:-

It Provides an Easy way to manage and Configure assemblies as well as Set Security Policies for accessing code.

This tool is Packaged as a Microsoft Management Console. (MMC) Snap-in.

Administools – Ctrl Panel – M.NET Framework Configurar tool.

This Tool designed the following:-

→ Manage Assemblies

Assemblies Can be added (or) deleted to the GAC

→ Configure assemblies.

When an assembly is updated, the Publisher of the assembly is responsible for updating the binding policy of the assembler

→ View .NET Framework Security and Modify an assembly's:

To be assigned Certain Permissions or rights & access.

→ Manage how individual applications interact with an assembly or Set of assemblies.

View a list of all assemblies an appln uses and Set the Version that your appln uses.

——— xxx ———

Understanding tho C# Compiler:-

Many developers writing nontrivial .NET appln rely on Visual Studio (or) Some other Integrated Developement Environment (IDE) to enter Source code, Link external assemblies, Perform debugging, and Create the final Compiled Output.

If you fall into this category, it is not essential that you understand how to use the .NET SDK and raw C# Compiler

It will increase your Understanding of the .NET Compilation process and give you a better feel for working with assemblies.

It will also acquaint you with the Command Line as a way to work with SDK Pgm's. You will occasionally find it useful to perform Compilation in that environment rather than firing up your IDE

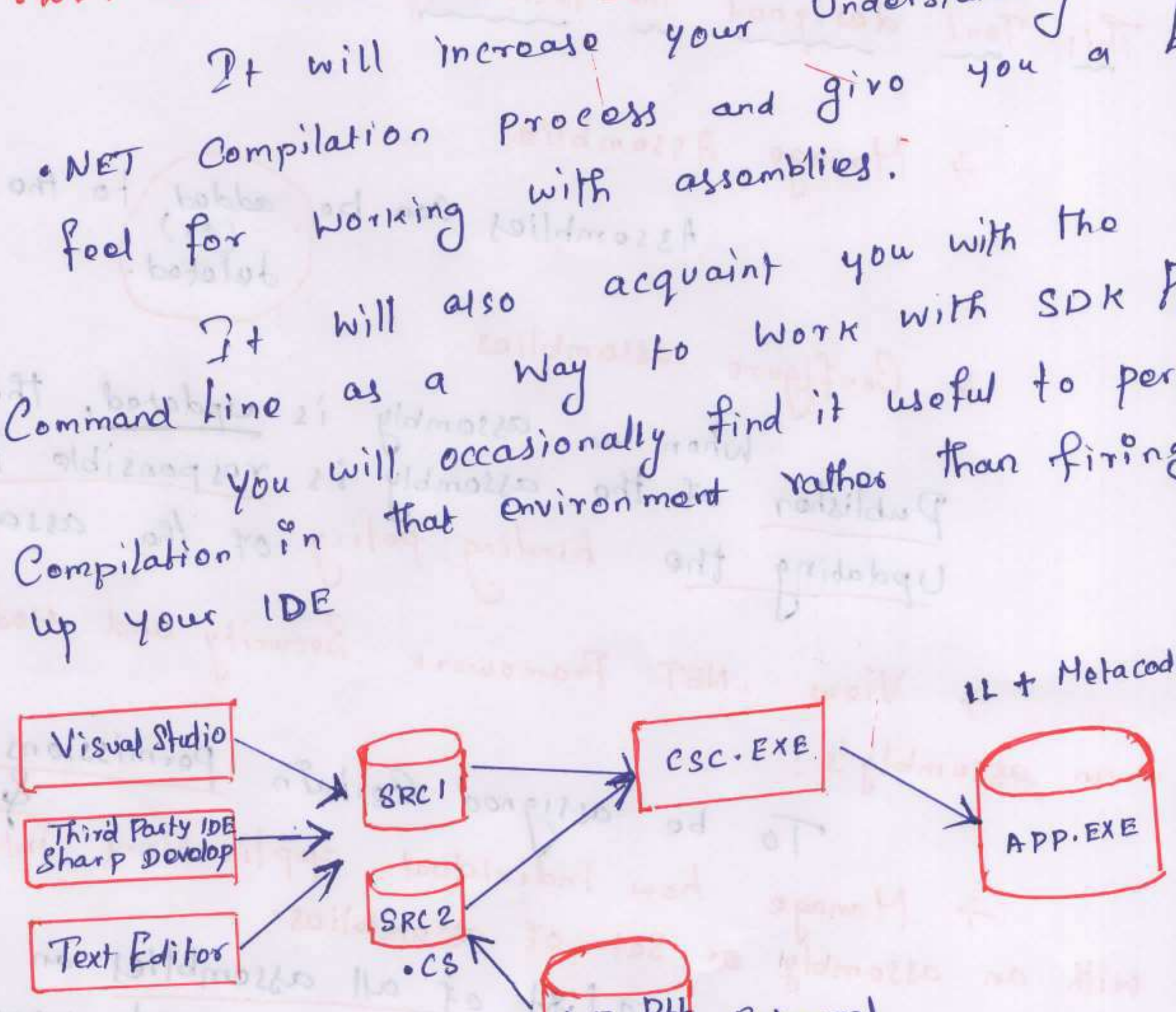


Fig: Compilation Process.

The basic Steps that occur in Converting Source Code to the final Compiled Output.

The purpose of this Section is to demonstrate how ~~to~~ a text editor and C# compiler can be used an application.

Locating the Compiler:-

The C# Compiler, csc.exe, is located in the Path where the .NET Framework is installed.

```
c:\winnt|Microsoft.NET|Framework\V2.0.40607
```

Of course, this may vary depending on your Operating System and the Version of Framework installed.

To make the Compiler available from the Command line in any Current directory

```
C:\>csc /help
```

Compiling from the Command Line:

The Compile C# Console application client.cs into the executable client.exe, enter either of the following Stmnt's at the Command prompt:

```
C:\> csc client.cs
C:\> csc /t:exe client.cs
```

Both stmnts Compile the Source into an executable (.exe) file - the default O/P from the Compiler.

/t:winexe — /t:exe - to create a WinForms appln. but the Console will be visible as background window.

C# Command-line Compiler

/addmodule : to be included in assembly Created
This is an Easy way to Create a
Multi-file assembly

/debug : Causes debug info to be produced.

/define : Preprocessor directive Can be passed
to compiler.
/define : DEBUG

/delaysign : Builds an assembly using delayed
Signing of the Strong name.

/doc : to Specify that an o/p file XML
documentation to be produced.

/out : Name of the file Containing
Compiled o/p, The default is the name
of the i/p file with .exe Suffix

client.cs

```
Using System;
public class MyApp
{ Static void main (String [] args)
{
   ShowName. Show me ("Core C#");
}}
```

Client Lib.cs

```
Using System;
public Class ShowName
{
 public Static Void Showme (String MyName)
{
  Console. WriteLine(MyName);
}
}
```

— Note —
not important
Code Just understand
use this Code

Example 1 : Compiling Multiple files:

The C# Compiler accepts any no. of input Source files.

It Combines their o/p into a Single file assembly:

```
csc /out: client.exe client.cs clientlib.cs
```

Example 2 : Creating and Using a Code library:-

The Code in clientlib Can be placed in a Separate Library that Can be accessed by any Client

```
csc /t: library clientlib.cs
```

The Output is an assembly named clientlib.dll Now, Compile the client Code and reference this external assembly:

```
csc /r: clientlib.dll client.cs
```

The o/p is an assembly named client.exe. If you examine this With Ildasm, you see that the manifest Contains a reference to the clientlib assembly.

Example 3 : Creating an Assembly with Multiple Files.

Rather than existing as a Separate Assembly, clientlib Can also be Packaged as a Separate file inside the client.exe assembly.

B'cz Only One file in an assembly may Contain Manifest

It is first necessary to compile clientlib.cs into a Portable Executable" (PE) Module. This done by selecting module as the target O/P.

csc /t: module clientlib.cs

The o/p file is client file.net module. Now, it can be Placed in the client.exe assembly by using the Compiler's addmodule Switch:

csc /addmodule : clientlib.net module client.cs

The Resultant assembly Consists of two files:

client.exe & clientlib.net module.

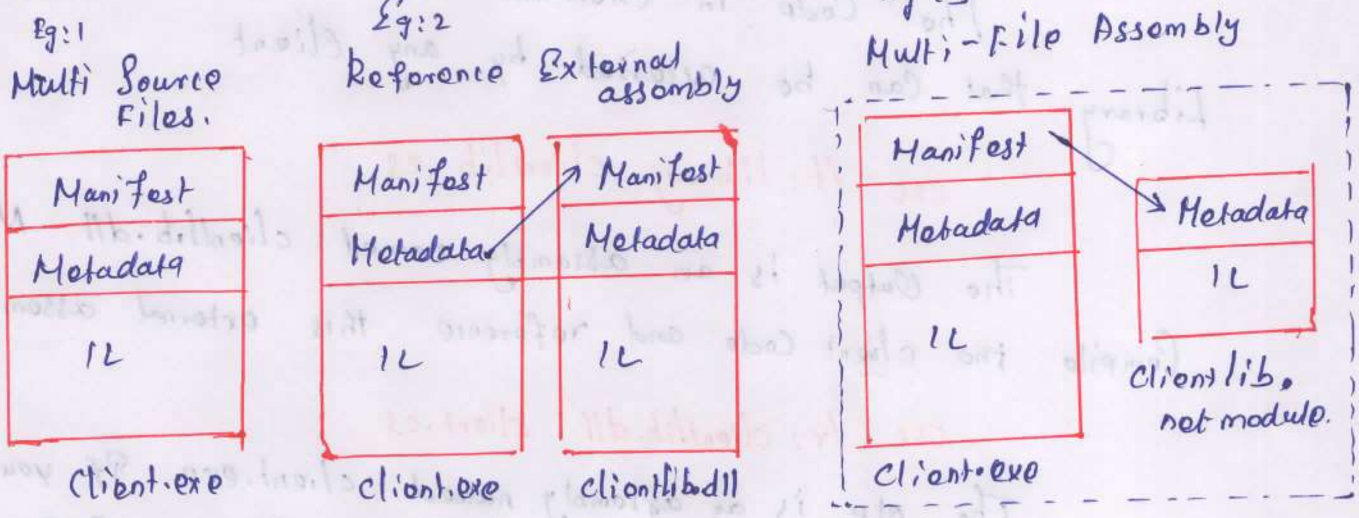


Fig: deploy an application.

—— xxx ——

CHAPTER : 2

OPERATORS :-

The C# Operators used for Arithmetic Operations, bit Manipulation, and Conditional Program Flow shoud be familiar to all Programmers.

Arithmatic Operators:-

The basic of numerical Operators.

The Precedence in which these Operators are applied during the Evaluation of an Expression.

Eg Numerical Operators:-

Operator	Description	Example

+
-
	Addition
	Subtraction
		int x = y + 10;
		x = y - 10;

* / %	Multiplication division Modulo	int x : 60. int y : 15 int z = x * y /2; //450 y = x % 29 // rem -2
++ --	Prefix / Post fix Preroment / decroment	x = 5; Consolo. Writoline(x++) x=5 Console. Writolino (++x) x=6
~	Bitwise Complement	int x = ~127 // return = ~128

>>
<<
	Shift right
	Shift Left
		byte x = 10; // binary 10 is 01010
		int result = x << 1; // 20 = 10100
		result = x >> 2; // 5 = 00101
		
		Works with byte, char, Short, Int and long

&
|
^
	Bitwise AND
	Bitwise OR
	Bitwise XOR
		byte x = 12; // 001100
		byte y = 11; // 001011
		int result = x & y; // 001000 = 8
		result = x ^ y; // 7 - 000111

Conditional and Relational Operators:

Relational Operators are used to Compare two Values and determine their relationship.

Statement	Description	Example.
== !=	Equality Inequality	if (x == y) {...}
< <= > >=	Numeric less than Less than or Equal to Greater That Greater than or equal to	if (x <= y) {...}
&&	Logical AND	if (x == y && y < 30) {...}
||	Logical OR	if first Expr is false. Second is not Evaluated.
&	Logical AND	if (x == y | y < 30) {...}
|	Logical OR	Always Evaluates Second Expression.
!	Logical negation	If !(x == y && y < 30) {...}

Note

The two forms of the logical AND / OR operations The && and || operators do not evaluate the second Expression if the first is false. a technique known as Short Circuit Evaluation.

The & and | operators always Evaluate both expressions. They are used primarily when the Expression. Values are returned from a method and you want to ensure that the methods are called.

C# Supports a ?: operator for Conditionally assigning a Value to a Variable.

It is basically Shorthand for using if-else Stmnt.

```
String pass;
int grade = 74;
if (grade >= 70) pass = "pass" else pass : "fail";
                // Expression ? op1 : op2
  pass = (grade >= 70) ? "pass" : "fail";
```

If the expression is true, the ?: Operator Returns the first Value.

If it's false the Second is Returned.

Control Flow Statements:-

The C# lang provides if and Switch Conditional Construct that Should be quite familiar to C++ and Java Pgmm'rs

Conditional Stmnts:

if - stmnt:-

```
if (boolean expression)
{
  // Stmnts
}
else { // stmnts
}
```

eg

```
if (bmi < 24.9)
{ weight = "normal";
  riskFactor = 2;
} else {
    weight = "over";
    risk factor = 6;
}
```

Switch Stmnt

```
Switch (expression)
{
 case Constant expression;
   // stmnts;
   // break/goto/return()
 Case Constant Expression;
   // stmnts;
   // break/goto/return():
default;
   // Stmnts
   // break/goto/return()
}
```

eg

```
Switch (ndx)
{ case 1:
     fabric = "Cotton";
     bliend = "100%.":
     break;
case 2:  // combine 2 & 3
case 3:
     fabric = "Cotton";
     blend = "60%";
     break;
default:   // optional
   fabric = "Cotton";
   blend = "50%":
   break:
```

if - else Statements:

Syntax

```
if ( boolean expression) Statement
if ( boolean expression) Statement else Stmnt 2
```

if stmnts behave as they do in other languages. The Only Issue you may encounter is how to format the stmnt's when nesting multiple if-else clauses.

```
// Nested if Stmnt's
if (age>16)
{
  if (gender == "M" )
  {
     type = "Man";
  }else{
     type = "Woman";
  }
}else
{
  type = "child";
}
```

Switch Statements:-

```
Switch (expression) {switch block}
```

The expression is One of the int types, a character or a String. The Switch block consists of Case labels and an Optiona defaults Labels.

associate with a Constant expression that must Implicitly Convert to the Same type as the expression.

Here is an example using a String expression.

```
// Switch with String expression

using System;
public Class MyApp
{
 static void Main (String[] args)
  {
   Switch (args[0])
   {
    Case "COTTON";
    Case "Cotton";
      Console.Writeline ("A good natural Fiber.");
      goto case "natural";
    Case "Polyestor";
      Console.Writeline ("A no-iron Synthetic Fiber.");
      break;
    Case "Natural";
      Console.Writeline ("A Natural Fiber.");
      break;
    default:
      Console.Writeline("Fiber is Unknown");
      break;
 } } }
```

=> C# does not permit execution to fall through One case block to the next. Each case block must end with a Stmnt that transfers Ctrl. This will be a break, goto, or return Stmnt.

Loops :-

C# Provides Four iteration Statements: while, do, for and foreach.

1st 3 are the Same Construts you find in C, C++ & Java.

The Foreach Stmnt is designed to loop through Collections of data Such as arrays.

While Loop :- (It is used to Execute of block of Stmnt's Untill the Specified the expression return as a true)

Syntax :-

```
while ( boolean expression)  { body }
```

The statement(s) in the loop body are executed until the boolean expression is false.

Example :-

```
int i = 1
while (i <= 4)
{
Console.writeline ("ivalue: {0}", i); i++;
}
```

```
byte [] r = { 0X00, 0X12, 0x34, 0x56, 0xAA, 0x55, 0xFF};
int ndx = 0;
int totVal = 0;
while ( ndx <= 6)
 { totVal += r[ndx];
   ndx += 1;
 }
```

do Loop:- [do-Whiloloop is Same as while loop, but Only the difference is while Loop will execute the stmnts only when the defined Contidition returns true but do-While loop will Execute the Stmnt' atleast Once.

Syntax:-

```
do {do-body} while (boolean expression);
```

Example

```
int i=1
do {
Console.Writeline ("i. Value: {0}", i); i++;
} while (i <= 4)
```

This is Similar to the While Stmnt except that the ~~Iteration~~ evaluation is Performed at the end of the iteration.

This loop executed at least Once.

```
byte [] r = {0x00, 0x12, 0x34, 0x56, 0xAB};
int ndx = 0;
int totVal = 0;
do
{ totval += r[ndx];
        ndx += 1;
}
while (ndx <= 6);
```

For loop:-

Syntax:-

```
for ([Initialization]; [termination Condition]; [iteration])
        {for-body}
```

The For Construct Contains initialization, a termination Condition, and the iteration statement to be used in the Loop.

All are Optional.

The Initialization is executed Once, and then the Condition is checked; as long as it is true, the iteration update occurs after the body is executed.

The iteration Statement is Usually a Simple increment to the Control Variable, but May be any Operation.

Example:-

```
int[] r = {80, 88, 90, 72, 68, 94, 83};
int totVal = 0;
for (int ndx = 0; ndx <= 6; ndx++)
{
    totVal += r[ndx];
}
```

If any of the clauses in the for statement are left out they must be accounted for elsewhere in the Code. This example illustrates how Omission of the for-iteration clause is handled.

```
for (ndx = 0; ndx < 6;)
{
    totVal += r[ndx];
    ndx++;                  // increment here
}
```

You can also leave out all of the For clauses:

```
for ( ; ; ) {body}    //equivalent to While (true) {body}
```

A return, goto or break Statement is required to exit this loop.

For each Loop:-

Syntax:-

```
Foreach (type Identifier in Collection)
    {body}
```

The type and identifier declare the iteration Variable. This Construct loops Once for each element in the Collection and sets the iteration Variable to the Value of the Current Collection element.

The iteration Variable is read-only, and a Compile error occurs if the Program attempts to set" it's Value.

Example:-

```
int totVal = 0;
Foreach ( int array Val in r)
{
    totVal += array Val;
}
```

1-D - index. 0
move - Asc ow

M-D - RMI - first

2-D - 1st colum move
across row
when it reaches end
it moves to the nxt row of the first

UNIT - II

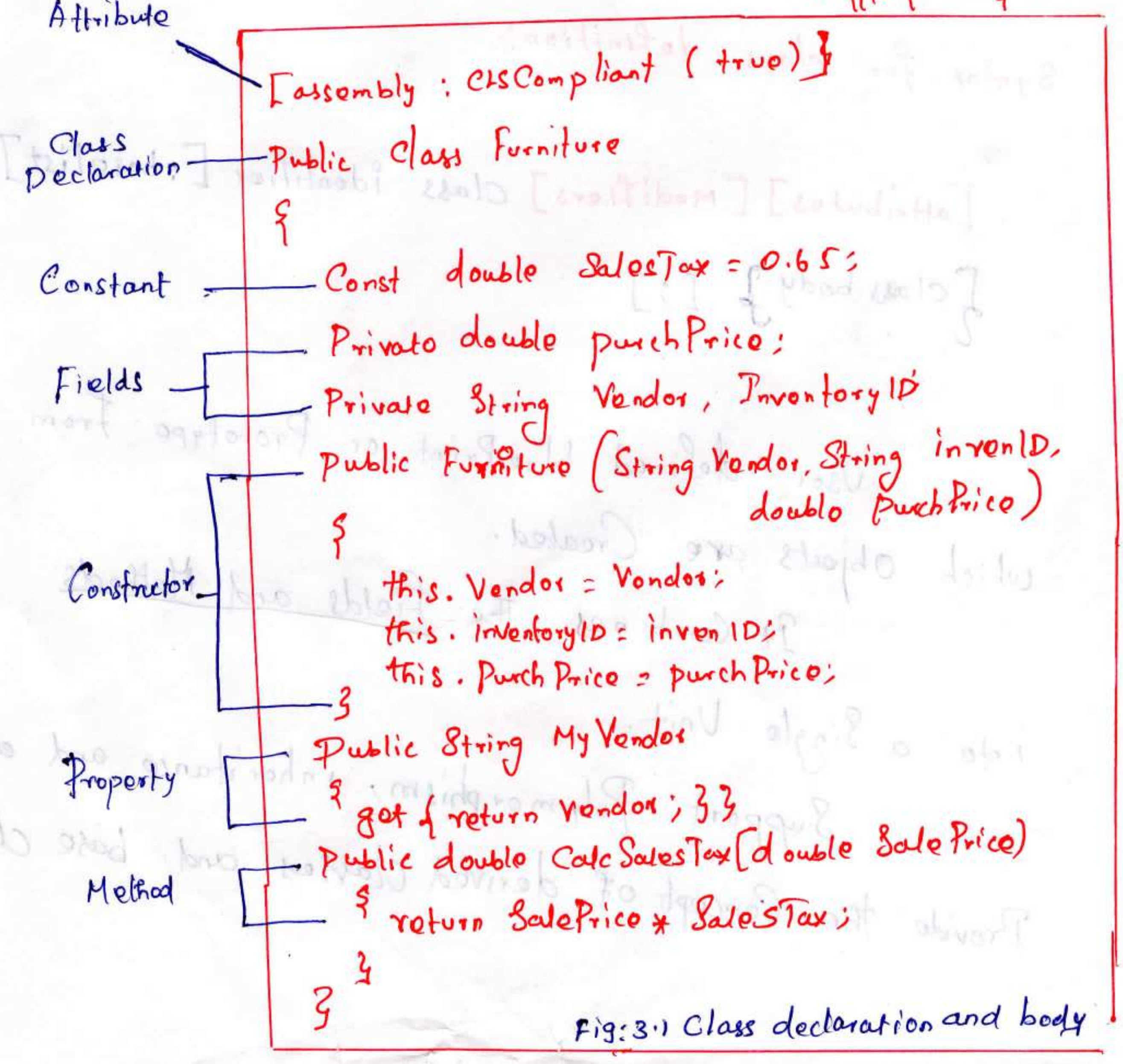
CLASS DESIGN AND FILE I/O

Introduction to a C# class:-

Fig 3.1 displays a class declaration followed by a body Containing typical Classe Members, a Constant Fields, a Constructor Containing initialization code, a Property and Method.

[It Combine various ~~dat~~ types of data members. Such as Fields, Properties, Member functions, Events

public class users
{ //proper method, events, etc
}



```
[assembly : ClsCompliant (true)]
Public Class Furniture
{
    Const double SalesTax = 0.65;
    Private double purchPrice;
    Private String Vendor, InventoryID
    Public Furniture (String Vendor, String invenID,
                      double PurchPrice)
    {
        this. Vendor = Vendor;
        this. inventoryID = inven ID;
        this . Purch Price = purch Price;
    }
    Public String My Vendor
    { get { return vendor ; } }
    Public double CalcSalesTax[double SalePrice)
    { return SalePrice * SalesTax;
    }
}
```

Fig:3.1 Class declaration and body

3.2:- Defining a Class:-

A class definition Consists of an Optional attributes List, Optional Modifiers, the Word class followed by the class identifier (name), and an Optional list Containing a base class (or) interfaces to be used for inheritance.

Consisting of the Code and class Members Such as Methods and Properties.

Syntax for class definition:-

[attributes] [Modifiers] Class identifier [: baselist]

{class body} [;]

User defined bluePrint or Prototype from which Objects are Created.

It Combines the fields and Methods into a Single Unit.

Support Polymorphism, inheritance and also Provide the Concept of derived Classes and base classes.

Attributes:- [attributes]

An attribute Consists of attribute name followed by an Optional list of Positional (or) named arguments

Examples:-

The attribute Section Contains an attribute name Only

[classDesc]

Single attribute with named arguments and positional argument (0);

[classDesc (Author = "Gopal", 0)]

Multiple attribute can be defined within brackets:-

[classDesc (Author = "Gopal"), classDesc[Author = "Selva")]

Conditional attribute:-

The conditional attribute is attached to methods Only.

Compiler should be generated Intermediate Language (IL) Code to call the method.

```
[conditional]. ("TRACE")]
    public static Void ListTrace()
    { Console.Writeline ("Trace is on"); }
[conditional] ("DEBUG")]
    public static void ListDebug()
    { Console.Writeline ("Debug is on"); }
```

Access Modifiers :-

The Primary role of Modifiers is to designate Scope (or) Visibility) of the accessibility / also Called as types and type Members.

Control to access a class / Members

A class access modifier indicates whether a Class is accessible from other assemblies.

The Same assembly. a Containing class, or classes derived from a Containing class.

Public : A class can be accessed from any assembly.

Protected : Applies Only to a nested Class (class defined within another Class)

Internal : Access is limited to classes in the Same assembly (This is default Access)

Private : Applies Only to a nested Class, Access is limited to the Container Class.

Protected : The Only Case where Multiple Modifiers May be used

Internal : Access is limited to the current assembly (or) Types derived from the Containing class.

class Identifier:-

Identifiers are used for identification purposes,

(or) identifiers are ~~[illegible]~~ user - defined name of the Program Components.

C# an identifier can be class name, Method name Variable name (or) Label.

Eg

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

3 identifiers

GFG - class name
Main - Method name
x - Var name

Rules:-

-> The only allowed char for identifiers are all alphanumeric characters [A-Z], [a-z] [0-9], (underscore) '_'

-> identifiers should not start with digits [0-9]
Eg:- 123Gopal in not Valid

-> Identifier are not allowed to use as keyword unless they include @ as a Prefix

eg @Gopal

Baso classes, Intorfacos, and Inheritance.

Baso class:-

" " Is a Class that is used to Create, or derive, Other Classes.

Classes derived from a base class are called child classes. Sub classes or derived classes.

A base class does not inherit from any other class

If Consider that Parent of a derived class.

eg:-
```
using System;
namespace InheritanceAppln
Class Shape
{
public Void Setwidth (int w)
  { width = w;
}
class Rectangle : Shape
{
  public int getarea()
   { return (width * height);
  }
}
```

Interface:-

-> Std.

-> Accessbility for Std Creator

-> Security for the Implementation

(eg: nokia) Moha

Class, method, Propert
↳ implem
i/f -> derived

Eg:-

```
Using System;
Interface IEmployee
{
  void Imethod();
}
class Employee : IEmployee
 {
  public Void Imethod()
  {
   Console.writeline("IMethod");
  }
Class Program
 {
 public Static void main (String[] args)
 {
  IEmployee e= new Employee();
  e.Imethod();
 }}
```

O/P

I Method

Inheritance

,, from a base class is referred to as implementation Inheritance.

derived Class inherits all of the members of the base class.

Base classes, I/f & Inheritance.

It contains a define a class or interface(s) from which a class may derive it's behaviour & capabilities.

The new class is referred to as the derived class, and the class or i/f from which it inherits i's the base class (or) i/f.

Eg:-

FCL I/f and User-defined a base class

```
public interface System.IComparable
{ Int32 CompareTo (Object object); }

Class Furniture
  { ....
    } //Derived class
Class sofa : Furniture
  { ....
    .. ..          } //Inheri'ts from One base class
  }
```

// following inherits from One base class & One interface.

```
class Recliner : Furniture , IComparable
  {
     ----
     - - -
  }
```

Overview of Class Members:-

. NET Class It classified broadly as members that hold data Constants, Fields, and Properties

Members that Provide functionality - the Constructor, method, and Event.

Member Type	Valid-in	Description
Constant	class, structure	A symbol that represent an Unchanging Value.
Field	class, structure	A Variable that holds a data Value It May be Read-Only/ R/W
Property	Class, Structure	The code to read or write to a Property is implemented implicitly by. NET as 2 Separate Methods. It uses an Accessor that speifig the code to be executed.
Constructor	Class, Structure	3 types of Constructors - Instance ↳ Initialize Fields when an instance a Class is Created. - Private - Commonly used to prevent instances of Class - Static Initialize Class before any instance is Created.

Method	Class Structure Interface	defines an action or Computation.
Events	Class Structure Interface.	A way for a Class/object to notify Other classes/objects (ie) State has Changed.
Types	Class Structure I/f	Classes, I/f, Structs, delegates.

Member Access Modifiers:-

The access modifiers used for a Class declaration Can also be applied to class members.

classes & assemblies that have access to the Class.

Class can be accessed by class in :	Access modifiers.			
	Public	Protected	Internal	Private
Another Assembly	YES	*	No	*
Same Assembly	Yes	*	YES	*
Containing Class	YES	YES	YES	YES
Class derived from Containing Class	YES	YES	YES	YES

* Not Applicable.

Constants , Fields , and Properties :-

Constants , Fields and properties are the members of a class that maintain the Content or State of the class

Use Constants for Values that will never Change.

use Fields to maintain Private data within a Class.

use Properties to Control access to data in a class.

Constants :-

C# uses the Const keyword to declare Variables that have a fixed. unalterable Value.

Rules :-

★ The Const Keyword Can be used to declare Multiple Constants in One Stmnt.

★ A Constant must be defined as a Primitive type Such as String or double

★ Constants Cannot be accessed ~~the~~ ~~Constants~~ ~~be~~ from an instances of the class.

The ShowConversion class access the Constants without instantiating the Class.

Eg:-

```
    Using System;
    class Conversions
    {
      public Const double cm = 2.54;
      public Const double Grams = 454.0 , Km = .62 ;
      public Const String ProjectName = "Metrices";
    }
      class ShowConversions
      {
        Static void main()
        {
          double pounds, gramweight;
          gramweight = 1362;
          pounds = gramweight
          Console.Writeline ("{0} Grams = {1} pounds", gramweight
                                         , Pounds);
          Console.Writeline ("cm per inch {0}", Conversion.cm);

          Conversions c = now Conversions(); //create Class
                                              // instance

       // This fails to compile. Cannot access Const from obj

          Console.Writeline ("cm per inch {0}", c.cm);
        }
  }
```

Fields :-

A Field is also used to Store data within a class. It differs from a Const in two Significant Ways. It's Value is determined at runtime, and It's Type is not restricted to Primitives.

Field Modifiers :-

Fields have two additional modifiers: Static and read Only

Modifier	Definition
Static	The Field is part of the Class State rather than any instances of the class. This Means that it can be referred directly (like a Constant) by Specifying Classname . fieldname without Creating an Instance of a Class.
readOnly	The Field can Only be assigned Value in the declaration Stmnt (or) Class Constructor not effect is turn the field into a Constant An Error results if Code later attempts to change the Value of the Field.

Note:- If a Field is not initialized, it is set to the default value for it's type = 0, for numbers, null for a reference type, Single Quotation Marks (' ') for a String & false for boolean.

There is One case where Setting a Field to Public makes Sense: When your program requires a global Constant Value.

By declaring a field to be public Static readonly You Can Create a runtime Constant.

Eg:-

Const double SalesTax = .065;

Can be replaced with a field

public Static readonly double SalesTax = .065;

Properties:-

It is used to Control read and Write access to Value within a Class.

Java, C++ Pgm'rs Create Property by Writing an Accessor method to retrive field data and a Mutator method. to set it.

Unlike these languages, the C# Compiler actually recognizes a Special property

To Construct & provides a Simplified Syntax for Creating and accessing data.

Syntax:-

```
[attributes] <modifiers> <data type> <property name>
{
[access modifier] get
{ ....
   return (propertyValue)
}
[access modifier] Set
{ .... Code to set a field to the keyword Value }
}
```

Note:-

-> 4 access modifiers -> static, abstract, new, Virtual (or) override. Abstract is used Only in an abstract class

virtual is used in a base class and Permits a Sub class to Override the Property.

-> Value is an implicit Parameter Containing the Value Passed when a Property is Called.

-> To get and set accessors May have d/f access modifiers.

Methods :-

Methods are to Classes as Verbs are to Sentences. They Perform the actions that define the behaviour of the Class.

A Method is identified by it's Signature. Which Consists of the Method name and the number and data type of Each Parameter.

A Signature is Considered Unique as long as no Other method has the Same name and matching parameter.

Method Modifiers :-

methods have Seven additional modifiers in table. Five of these - new, Virtual, Override, Sealed and abstract provide a means for Supporting polymorphism.

Modifier	Description
1) Static	It can be referenced directly by Specifying class name.method (Parameter) without Creating an instance of the class.
2) Virtual	It Can be Overridden in a Subclass this can't be used with Static/Private access modifiers.
3) Override	The method overrides a method of the Same name in a base class The overridden method in the base class must be Virtual

4) new — Permits method in an inherited class to "hide" a non-virtual method with a Same name in the base class.

It replaces the Original method rather than overriding it.

5) Sealed — Prevents a derived Class from overriding this method.

used with the "Override" modifier.

6) abstract — It contains no implementation details and must be implemented by any Sub class

Can Only be used as a member of an abstract class

7) extern — It is generally used with DLLImport attributes that Specifies ~~and methods helper Class~~ that a DLL to provide the implementation.

Static Method:-

As with Other class members, the Static modifier defines a member behaviour is global to the class and not Specific to an instance of a class. The modifier is most Commonly used with Constructors. & methods in helper classes that can be used w/o instantiation.

Eg

```
Using System;
Class Conversions
{
        Private Static double cmPerInch = 2.54;
        Private Static double gmPerPound = 455;
        Public Static double inchesToMetric (double inches)
        {
            return (inches * cmPerInch);
        }
        Public Static double poundsToGrams (double pounds)
        {
            return (pounds * gmPerPound);
        }
}
Class Test
{
    Static void main()
    {
        double cm, grams;
        cm = Conversions.inchesToMetric(28.5);
        grams = Conversions.PoundsToGrams(984.4);
    }
}
```

In this eg :- the Conversions class contains methods that convert Units from the metric System.

Syntax Classname. method (Parameter).

Method Inhoritance with Virtual and Override Modifiers

Inheritance enables ~~to~~ a Program to Create a new class that takes the form and functionality of an existing (base) class.

The Capability of the Subclass and base Class to respond differently to the Same message is Classical Polymorphism.

base and derived class(es) provide different Code for methods having Same Signature.

default, methods in the base class Can not be Changed. in the derived Class.

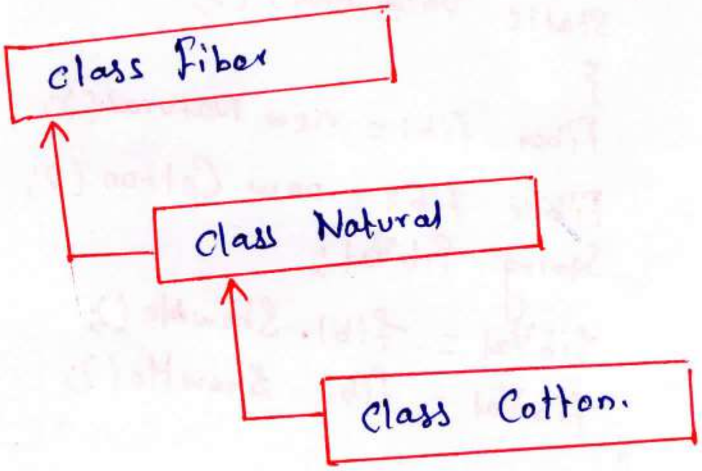


Fig:- Relationship b/w base class and Sub classes.

Eg:-

```
using System;
class Fiber
{
  public Virtual String ShowMe()
  {return ("Base");
  }
Class Natural : Fiber
  {public override String Showme()
   {return("Natural");
   }
Class Cotton : Natural
  {
   public Override String ShowMe()
   {return ("Cotton");
   }
}
```

Eg

```
Class Test
{
 Static Void main ();
 {
 Fiber fib1 = new Natural();      //instance of Natural
 Fiber fib2 = now Cotton ();      // Instance of Cotton
 String fibVal;
 fibVal = fib1. ShowMe ();        // Returns Naturel
 fibVal = fib2. ShowMe ();        // Returns cotton
 }
}
```

Each SubClass implements its Own Overriding Code for the Virtual Method ShowMe

New Modifier and Versioning:-

There are Situating where it's useful to hide inherited members of a base class.

Eg Your class may Contain a method with a Same Signature as one in it's base class

To notify the compiler that your Subclass is Creating it's Own Version of the method.

it should use the new modifier in the declaration

ShowMe() is no longer Virtual and Can not be Overridden. For Natural to Create its own version.

It must use the new modifier in the declaration.

An instance of the Natural Class that calls ShowMe() invokes the new method.

```
Natural myFiber = new Natural();
String fibType = myFiber.ShowMe(); //return's Natural.
```

Sealed and Abstract Modifiers:-

A Sealed modifier indicates that a Method Cannot be Overridden in an inheriting Class.

an abstract modifier requires that the inheriting Class implement it.

The base class provides the method declaration but no implementation.

This code Sample illustrates how an inheriting Class uses the Sealed Modifier to Prevent a Method from being Overridden the inheritance chain.

Sealed is always paired with the Override modifier.

```
Class A
    { public Virtual void PrintID'
      ...
    }
Class B: A
    { Sealed override public void PrintID
    }...
Class C : B    // This is illegal bcz it is marked Sealed in B.
    {
      Override public void PrintID
      ...
    }
```

An Abstract method represents a functions with a Signature - but no implementation Code - that must be defined by non-abstratt class inheriting it.

Rules:-

★ Abstrat Methods Can Only be declarod in abstract Class, however abstrat Classes May have non-abstract methods.

★. The Method body Consists of a Semicolon:

Public abstract void myMethod();

★ Although implicitly Virtual, abstract methods Can not have the Virtual Modifier.

★ A Virtual method Can be Overridden by an abstract Method.

Passing Parameters :-

C# Provides two modifiers that signify a Parameter is being passed by reference: Out and ref. Both of these keywords cause ~~of~~ the address of the Parameter to be Passed to the the target Method.

Use ref when the calling Method initializes the Parameter Value.

Out when the Called method assign's the Initial Value.

Using the ref and out Parameter Modifiers:-

Eg|

```
Class TestParms
{
  public static void FillArray (out double[], Prices)
  {
     Prices = new double [4], {50.00, 80.00, 120.00, 200.00};
  }
Public Static Void UpdateArray (ref double[] Prices)
  {
     Prices [0] = Prices [0] * 1.50;
     Prices[1] = Prices[1] * 2.0;
  }
Public Static double Taxval (double ourprice, Out doublo taxAmt)
{
   doublo totVal = 1.10 * Our Price ;
     taxAmt = totVal - Our Prico ;
     Our prico = 0.0 ;      // Doos not affoct Calling Parametor
     return totval ;
   }
}
Class MyApp
  {
  public Static Void Main ()
    {
      doublo[] PricoArray ;
      doublo taxAmt ;
    // (1) Call Mothod to initialize array
     TestParms. FillArray (Out priceArray);
       Console.Writolino ( PricoArray[1].ToString() );  //80
    // (2) call mothod to updato array
     TestParms. UpdatoArray ( rof priceArray);
       consolo.writolie (PricoArray[1].ToString()); // 160
    // (3) call method to calculato amount of tax.
       double newtax = TestParms. TaxVal( Ourprice, out taxAmt);
       Console.Writoline (taxAmt.ToString()); // 15
       Console.Writolino (Our Price);          // 150.00
    }
  }
```

Tho class MyApp is usod to invoke threo methods in tho Test Parms class:

1) Fill Array is invoKod to initialize the array.

This requires passing the array as a Parametor with the Out modifier.

2) Tho roturnod array is now passod to a second mothod that modifies two elements in the array. The ref modifier indicates tho array can be modified.

3) OurPrice is passod to a mothod that calculatos the amount of tax and assign it to the Parametor taxAmt.

Athough OurPrico is set to 0 within The method bcz it is passod by Value.

Constructors :-

The CLR requires that Every class have a Constructor - a Special - purpose method that initialize a class (or) class instance the First time it is referenced.

Three types of Constructors:-

→ Instance
→ Private
→ Static

Instance Constructor:-

Syntax:-

```
[attributes] [modifiers] <identifier> ([Parameter-List])
[: Initializer]
{
    Constructor-body
}
```

The Syntax is the Same as that of the method Except, it does not have a return data type and adds an initializer Option.

→ The Identifier (constructor name) must be the Same as the class name
→ The Initializer provides a way to invoke code Prior to entering the Constructor - body

2 forms
base (argument List.) - Calls a base class
this (argument list) - Calls a another Constructor

→ If no Explicit Constructor is provided for a class, the the Compiler Creates a default Parameterless Constructor

The instance Constructor is called as "part of Creating a class instance."

Bcz, a class may contain multiple Constructors the compiler calls the Constructor whose signature matches that of the call.

```
Fiber fib1 = new Cotton();
Fiber fib2 = new Cotton("Egyptian");
```

.NET Constructs an Object by allocating memory, Zeroing out the Memory and Calling the instance Constructor.

The Constructor sets the state of the object, Any Fields in the class that are not explicitly initialized are set as Zero (or) null.

Private Constructor :-

Private Constructor is a Special instance Constructor Present in C# language.

Private Constructor are used in class that Contains Only Static Members.

Private Constructor is always declared by using a "Private" Keyword.

Points

-> Only Static Members

To Prevent the Creation of the instances of that Class.

The class Contains Only Private Constructor "w/o Parameters".

2 Static Fields:-

-> Pi

-> e (Natural logarithmic base)

The Methods behave a builtin-functions and there is no reason for a Program to Create an instance of the Math class in Order to use them.

Eg:-

```
using Systom;
Class Conversions

Static cm PerInch = 2.54;
Privato Static double gmPerPound = 455;
Public Static doublo inchesToMetric (double inches)
 {
    return(inches * cmPerInch);
 }
Public Static double poundsToGrams (double Pounds)
  {
     return (pounds * gmPerPound);
 }
// Privato Constructor prevents Creating class instance
  Privato Conversions ()
     {
      ---;
     }
}
```

Static Constructor:-

Also known as a class or type Constructor,

The Static Constructor is executed aftor the type is Loaded and before any one of the type members is accessed.

It's Primary purpose is to Initialize Static Class Members

→ It can't have Parameter & can't be Overloaded Consequently a Class have Only One Static Constructor.

→ It must havo a Privato access, which C# assign automatically.

→ It cannot call other constructors.

→ It can only access static Members.

```
Class  Baso class
{
  Privato Static int callcounter;
  //static Constructor
  Static  BasoClass ()
  {
    Console.writoline ("Static Constructor : " + callCounter);

  }
  //Instance Constructor
  public    BasoClass ()
  {
    call counter += 1 ;
    Console.write line ("Instance Constructor : " + callcounter);

  }
  //...other class operations

}
```

This class contains a Static Initializer, a Static Constructor.

Instance Constructor let's look at the Sequence of events that occur when the class is instantiated.

```
Baseclass myclass1 = new BaseClass( );
BaseClass myclass2 = new BaseClass( );
BaseClass myclass2 = new BaseClass( );
```

O/P

```
Static Constructor = 0
Instance constructor = 1
   ''        ''      = 2
   ''        ''      = 3
```

The Compiler First emits Code to initialize the Static field to 0; It then executes the Static Constructor Code. that displays the initial Value of calcounter, Next the base Constructor is executed. It increments the counter and displays it's Current Value which is now 1. Each time a new instance of Base class is created. the Counter is incremented. Note that the static Constructor is Executed Only Once, No matter how many instances of the Class are created.

Delegates:-

→ A delegate is an object that can refer a Method.

→ Therefore, when we create a delegate, we are creating an object that can hold a reference to a Method

Creating and Using delegates involves Four steps:-

- → Delegate Declaration
- → Delegate Method definition
- → Delegate instantiation
- → Delegate invocation.

Delegate Declaration:-

→ A delegate declaration is a type declaration.

modifier delegate return-type delegate_name (Parameters)

→ That delegate is the keyword that signifies that the declaration represents a class type derived from System.Delegate.

→ Modifiers used with delegates are:-
new, internal, public, private, protected.

For Eg:-
```
delegate void SimpleDelegate();
delegate int MathOperation (int x, int y);
public delegate int Compareitems (object obj1, object obj2);
```

→ Delegate types are implicitly Sealed.

Delegate Methods:-

The methods whose references are encapsulated into a delegate instances are known as delegate methods (or) Callable entities.

The Signature and return type of delegate methods. The Signature and return type of the must exactly match the Signature and return type of the delegate.

for

```
delegate void Delegate1();
Can encapsulate to the following methods.
public void F1()
  {
   Console.Writeline("F1");
  }
Static void F2()
  {
  }
```

Delegate Instantiation:-

Syntax for Instantiating for their instances.

```
        new delegate-type(expression);
delegate int productdelegate(int x, int y); //Delegate Declartion
Static int Product (int a, int b) // delegate method
{
   return (a*b);
}
Productdelegates P= new productdelegate(expression);
```

Delegate Invocation:-

When a delegate is invoked, it in turn invokes the method whose reference has been encapsulated into the delegate.

delegate_Object (Paramaters list);

for eg:-

```
delegate1 (x, y);
double result = delegate (4.5, 5.6);
```

Events:-

An Event is a delegate type class member ie used by object (or) class To Provide a notification to other objects that an Event has Occured.

Events are declared using the Simple Event declaration format as follows

modifier event type event-name;

The modifier may be a new, Static, Override abstract and Sealed

public event EventHandler

The EventHandler is a delegate and click is an Event.

Delegates and Events :-

The .NET delegates eliminates this Problem. The C# Compiler Performs type checking to ensure that a delegate Only calls methods that have a Signature and ~~d~~ returntype Matching that Specified in the delegate declaration.

Eg:- of delegate declaration.

```
public delegate Void MyString (String msg);
                         └> <identifier>
```

When the delegate is declared, the C# Compiler Creates a Sealed class having the name of the delegate identifier (identifier>.

This class defines a Constructor that accepts the name of a Method - Static (or) instance - as one of it's Parameters.

It also Contains methods that enable the delegate to Maintain a list of target methods. This Means that - Unlike the Callback approach - a Single delegate can call Multiple Event handling methods.

```
Using System
Using System.Threading;
Class DelegateSample
{
 public delegate void MyString (string s);
 public static void Printlower(String s)
 {
   Console.writeline (s.ToLower());
 }
 public static void PrintUpper(String s)
  { Console.writeline (s.ToUpper());
  }
 public static Void Main()
  {
    MyString myDel;
     // register method to be called by delegate
    myDel = new MyString(Printlower);
    // register Second Method
    myDel += new MyString(Print Upper);
    // Call Delegate
    MyDel ("My Name is Gopal");
  }
}
```

O/P:-

```
my name is gopal
MY NAME IS GOPAL
```

```
myDel += new MyString (PrintUpper); // register for Callback
MyDel -= new MyString (PrintUpper); // remove method from list
```

Delegate - Based Event Handling:-

.NET event model is based on Observer Design Pattern.

This pattern is defined as "a One-to-Many dependancy b/w Objects so that when One Obj Changes State, all its dependents are notified and updated automatically

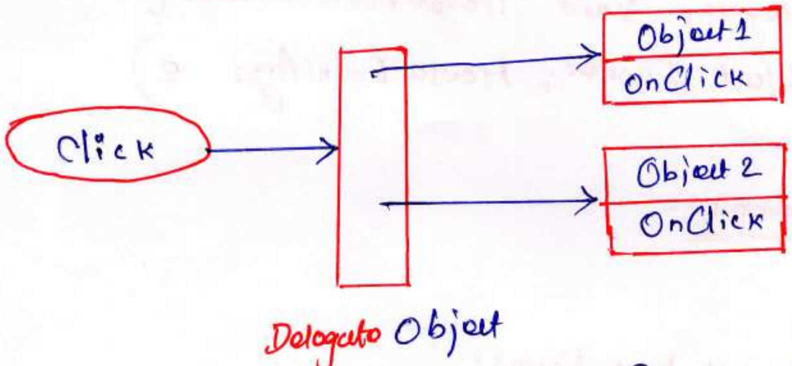


Fig:- Event Handling Relationship

Working with Built-In Events:-

The 3.12 displays a Form and permits a user to draw a line on the form by pushing a mouse key down, dragging the mouse and then raising the mouse key

To get the end points of the line. it is necessary to recognize the MouseDown and MouseUP Events.

When a MouseUP occurs, the Line is drawn.

The delegate, Mouse EventHandler, and the Event, MouseDown are predefined in the FCL (Frame work class Library)

The += Operator is used to register method associated with an Event.

```
this.MouseDown += new MouseEventHandler(onMouseDown);
```

The underlying construct of this Stmnt is

```
this.event += new delegate(Event handler method);
```

Mouse Event Handler delegate

```
public delegate void MouseEventHandler(
        Object Sender, MouseEventArgs e)
```

3.12 Event Handler Example:-

```
Using System;
Using System.Windows.Forms;
Using System.Drawing;
Class DrawingForm.Form
{
   Private int Last X;
   Private int Last Y;
   Private Pen MyPen = Pens.Black; //defn color of drawn line
   public DrawingForm()
   {
     this.Text = "Drawing pad";
   // Create delegates to call MouseUp and MouseDown
 this.MouseDown += new mouseEventHandler(on mouseDown);
 this.MouseUp += new MouseEventHandler(on MouseUp);
 }
Private void onMouseDown (Object Sender , MouseEventArgs e)
  { Last x = e.X;
    Last y = e.Y;
  }
Private void onMouseUp (Object Sender, MouseEventArgs e)
 {
    // The next two statements draw a line on the form
  Graphics g = this.CreateGraphics ();
    if (last x >0)
      {
         g.DrawLine (myPen, Last X , Last Y, e.X , e.Y);
      }
    Last X = e.X;
    Last Y = e.Y;
 }
  Static void main()
   { Appln.Run (new Drawing Form());
   }
}
```

Using Anonymous Methods with Delegates:-

.NET 2.0 introduced a language Construct known as anonymous Methods. that eliminates the need for a Separate event handler Method.

Instead, Event handling Code is encapsulated within delegates.

Eg:-

```
this.MouseDown += new MouseEventHandler(onMouseDown);
```

with this Code that Creates a delegate and includes the Code to be Executed when the delegate is invoked.

```
this.MouseDown += delegate(Object Sender, EventArgs e)
{
    LastX = e.X;
    LastY = e.Y;
}
```

The code block, which replaces OnMouseDown, requires No method name, and is thus referred to as an anonymous method.

Look it's formal Syntax:

```
delegates [(parameter-List)] {anonymous-method-block}
```

→ The delegates Keyword is placed in front of the Code that is Executed. When the delegate is invoked

→ An Optional Parameter list may be used to Pass data to the Code block required by the Predefined delegate MouseEvent Handler

→ It Creates a new Class and Constructs a method inside it to Contain the code block. This method is called when the delegate is invoked.

To further clarify the Use of anonymous methods let's use them to Simplify the examples. 3.11 (Multicasting) Delegates.

a Custom delegate is declared & two callback methods are implemented and registered with the delegate, two callback methods are replaced with anonymous code blocks.

```
// delegate declaration
  public delegate void MyString (String s);

// Register two anonymous methods with the delegate.

   MyString myDel ;
   MyDel = delegated(String s)
      {
        Console.Writoline( S.Tolower());
      };
  MyDel += delegato( String s)
      {
        Consdo.Writeline (s.ToUpper());
      };
 // invoke delegate
      MyDel ("My name is Gopal");
```

When the delegate is called, it executes the Code provided in the two anonymous methods. which results is the input String being printed ~~to~~ in all lower and Upper Case letters, respectively.

Defining a Delegate to Work with Events :-

The Event delegate defines a Method Signature that takes a Single String Parameter.

The Signature Should Conform to that used by all built-in .NET delegates.

The EventHandler delegate provides an eg of the Signature that Should be used:

```
public delegate void EventHandler (object Sender
                  EventArgs eventArgs);
```

The delegate Signature should define a Void return type.

If have an Object & EventArgs type Parameter

The Sender parameter identifies the publisher of the Event.

That enables a client to use a Single Method to handle & identify an Event from Multiple Sources.

The Second Parameter Contains the data associated with the Event .NET Provide the "EventArgs" Class generic Container to hold a list of arguments.

Creating an EventArgs type to be used as Parameter requires defining a new Class that inherits from EventArgs.

```
public class IOEventArgs : EventArgs
{
  public IOEventArgs (String msg)
    {
      this.eventMsg = msg;
    }
  public String Msg
    {
      get
        {
          return eventMsg;
        }
    }
  Private String eventMsg;
}
```

→ It must inherit from the EventArgs class

→ It's name Should end with EventArgs

→ Define the arguments as readOnly Fields (or) Properties

→ Use a Constructor to initialize the Values.

If an Event does not generate data, there is no need to create a Class to Serve as the EventArgs Parameter. Instead, Simply pass EventArgs. Empty

Operator Overloading :-

Built in Oporators Such as + and - usod So One rarely thinks of thom as a Prodefined implomontation. for Manipulating intrinsic types.

Tho + Operator usod for addition or Concatenation doponding on the data typos involved.

It makos Sense thon that these Operators Can't be appliod to Custom classes or Structuros of which the Compilor has no Knowlodge.

C# Providos a Mochanism referred to as Operator Overloading that enablos a Class to implement Code that dotormines how the class rosponds to the Oporator.

Syntax for Oporator Overloading

```
Public static <return typo> Operator <op> (Paramotor list)
   {
     implemontation code
   }
```

Rulos:-

→ Tho public & Static modifiors aro roquirod.

→ Tho roturn type is the Class type when Working with classes. It can novor be Void.

→ Op is a binary, Unary, or rolational Oporator Both equals (==) and not equals (!=) must be implomontod in a rolational Pair.

→ Binary Operators require two argumonts

→ Unary Oporators require Ono argumont

Operator Ovorloading with classes does not have to bo limitod to geometric (or) Spatial Objects.

Two Oporator Ovorloads (+ and -) add & remove Stocks from tho Portfolio.

It contains two classes: Ono that ropresonts a Stock and one that ropresents a Stock portpolio of Stocks.

```
Using Systom;
Class portfolio.
{
 public docimal risk;
 public docimal totValuo;
//Ovorloadod Oporator to add stock to portfolio
 public static portfolio Operator + (Portfolio p, Stock s)
 {
   docimal  currVal = p.totValue;
   docimal  currRisk = p. risk;
   p.totValue = p.totValue + s.StockVal;
   p.risk = (currVal/p.totValue) * p.risk +
                 (s.StockVal/p.totValue) * s.BetaVal;

   return p;
 }
//Ovorloadod Oporator to romovo Stock from portfolio
 public Static portfolio Operator - (Portfolio P, Stock s)
 {
   p.totValuo = p.totValuo - S.StockVal;
   P.risk = P.risk - ((s.BetaVal - p.risk) *
                         (s.StockVal/p.totValuo));
   return p;
 }
}

Class Stock
 {
   Privato docimal Valuo;
   Privato decimal beta; // risk increaso with Valuo.
   public Stock (decimal myBeta, docimal myValue,
                                   int Shares)
   {
```

```
Valuo = [docimal ) my Valuo * Sharos;
 beta  = my Beta ;
}
 public docimal StockVal
    {
     get
      {
       return Valuo ;
      }
    }
  public docimal BetaVal
    {get
      { return Bota ;
      }
    }
  Class MyAPP
  {
  public Static Void Main()
    { portfolio p = now Portfolio ();
      Stock hpq = now Stock(1.1m, 25M, 200);
      Stock ibm = now stock(1.05m, 95.0m, 100);
      p += hpq
      p += ibm
    Console.Write ("Valuo : {0}", p.tot Valuo.ToString());
    Consolo.Writolino ("risk: {0}" p.risk.ToString ('#.00"));

     p -= ibm.
     Consolo.Write ("valuo : {0}", p.tot valuo.ToString());
     Consolo.Write ("risk: {0}", p.risk.ToString ("#.00"));
    }
  }
}.
```

Intorfaces:-

An I/f generally dofines Set of Methods that will be implomontod by the class.

But I/f doesn't implement any method itself

Syntax

[attributos] [modifiers] interface identifier [: basolist]

{ interface body

}

Tho Syntax of the interface doclaration is identical to that of a class except that the keyword Intorfaces replacos class.

An I/f is basically a class that declares, but does not implement, it's Mombers

An instance of it can't be Created. and Classes that inherit from it must implement all of it's Methods

This sounds similar to an abstract class, which also can't be instantiatod and requires derived classes to implement its abstract Methods.

Tho difference is that an abstract class has many more capabilities.

It May be inhoritod by Sub classos, and it may Contain State data and Concrete Methods.

An interface defines a behavior for a class - Something it "can do".

The built-in .NET ICloneable I/f, Which Permits an object to create a copy of itself,

It should be used when the inheriting class is a "type of" the base class.

For Example:-

you could a create a shape as a base class, a circle as a subclass of shape. and the capability to change the size of the shape as an i/f Method.

-> An I/f can't inherit from a class

-> An I/f can inherit from Multiple i/fs.

-> A class can inherit from Multiple i/f, but Only one class

-> I/f Members must be Methods, Properties, events or indexers.

-> All I/f Members must have public access (The default)

-> By Convention, an I/f name should begin with the Uppercase I

Creating and using a Custom Interface:-

```
public interface IShapeFunction  //Inherit I/f
{
  double GetArea();  // Public abstract method

}
Class Circle : IshapeFunction
 {
   Private double radious
   Public circle(double rad).
   {
      radius = rad;
   }
   public double GetArea()
   {
    return (MathPI* radius * radius);
   }
   public String Show me ()
   {
     return ("circle");
  } }
Class Rectangle : IShapeFunction  // Inherit I/f
{
   Private double width, height;
   Public rectangle (double mywidth, double myHeight)
   {
     Width = MyWidth;
     Height = MyHeight;
   }
   public double GetArea()
    {
      return (width * height);
    }
```

The declaration of the I/f and Implementation of the class Straight forward.

A Circle & Rectangle class Inherit ~~from~~ the IShapeFunction Interface and implement it's GetArea Method.

=> The First Stmnt creates an instance of the Circle class

=> The Second Stmnt Creates a Variable that refers to this Circle Object

=> It's Specified as an ~~attemp~~ 'IShapeFunction type

=> It can only access members of that I/f.

=> This is why an attempt to reference the Showme Method fails. By Using the Interface type

=> You effectively Creates a filter that restricts access to members of the I/f only.

=> One of the most Valuable aspects of working with I/f is that a Pgmr Can treat disparate classes in a similar manner. as long as they implement the same I/f.

```
IShapeFunction [] myshapes = { mycircle . myRectangle };
MessageBox . Show ( myshape[0] . GetArea () . ToString() );
```

Working with Intefaces

Determining which Interface Members ~~Are~~ Available.

You May ~~d~~ not always know at Compile time whether a class implements a Specific I/f.

This is often the Case when Working with a Collection that contains a number of types.

To Perform this check at runtime, use the as or is keyword in your code.

```
// (1) as keyword to determine if i/f is implemented

Circle mycircle = new Circle(5,0);
IShapeFunction ~~my~~ myIcircle;
 myIcircle = mycircle as IShapeFunction;
 If (myICircle != null)   // I/f is implemented.

// (2) is Keyword to determine if i/f is implemented

     Circle mycircle = new Circle (5,0);
  If (mycircle is IShapeFunction) // True if I/f implemented.
```

Accessing Interface Methods:-

Bcz, a Class may inherit methods from a base class and/or Multiple i/fs there is the possibility that Inherited methods will have the same name.

To avoid this ambiguity, Specify an interface method declaration in the derived class with the I/f & Method name

```
double IShapeFunction.GetArea()          // <interface>.<method>
{
}
```

This not only permits a class to implement Multiple methods, but has the added effect of limiting access to this method to Interface references only

eg the following would result an error.

```
Circle myCircle = new Circle (5.0);
                // Can't reference explicit method
double myArea = myCircle. GetArea ();
```

Generics:-

To understand and appreciate the Concepts of generics.

Consider the need to Create a class that will Manage a Collection of objects.

The objects may be of any type and are specified at Compile time by a Parameter passed to the class.

In the 1.x Versions of .NET there is no way to Create ~~a~~ such a Class.

Your best Option is to create a class that ~~Cointa~~ Contains an array. that treats everything as object.

```
Object [] myStack = new Object [50];
myStack [0] = new Circlo [5, 0);  // Place Circle Object in array
myStack [1]  = "circle";          // Place String in array
Circlo C1 = myStack [0] as Circlo;
if ( c1 != null)                  // Circlo obj obtained
  {
  Circlo c2 = Circlo mystack [1];  // Invalid case Exception
```

The Primary clallange of working with genorics is getting used to the Syntax. Which can be used with a class Interface or Structure

The best way to approach the Syntax is to think of it as a way to pass the data type you'll be working with a Parameter to the gonoric Class.

Here is the declaration for a genoric class:

```
public Class MyCollection <T>
{
    T[] myStack = new T[50];
}
```

//T is Type Parameter
(replace the actual type rqsted)

Creating an instance of a generic class is straightforward:

```
myCollection <string> = new myCollection <string>;
```

↳ Type argumen

The following declaration requires that type parameter implement the ISerializable & IComparable interfaces:

```
public class myCollection <T> where
     T : ISerializable
     T : IComparable.
```

A Parameter may have multiple interface constraints and a single class restraint.

Three Class restraint:

Class – Parameter must be reference type
Struct – Parameter must be Value type
new() – Type Parameter must have a Parameterless constructor.

Eg

```
Using System.Collections.Generic
public class GenStack <T>
   Where T : class    // Constraints restricts access to ref types
{
  Private T[] StackCollection;
  Private Int Count = 0;
// Constructor
  public GenStack (int size)
  {
    StackCollection = new T[size];
  }
  Public void Add [T item)
  {
    StackCollection[count] = item;
    Count += 1;
  }
//
public T this (int ndx ]
 {
   get
    { if (! (ndx <0 || ndx > count -1]))
         return StackCollection [ndx];
    //return empty objeut .
  olso  return (defoult (T));
 }
}
 public  int  ItemCount
  {
    get {return count ;
    }
  public  int  Comparo<C> (T Value1 ,T Valuo 2)
   {
      //Case-Sensistivo Compurison : -1, 0(match), 1
    return Compare<T.>. Default. Compare <valuol, Valuo 2);
   }
 }
```

Structures :-

Struct - is often described as lightWeight Class.

It is similar to a Class in that its members include Fields, Properties, methods, events and Constructors

It Can inherit from interfaces.

→ A Struct is a Value types.
It implicitly inherits from the System.ValueType.

→ A Struct Cannot inherit from Classes
nor Can it be inherited

→ An explicitly declared Constructor must have at least One Parameter.

→ Struct Member Can't have initializers.

The field Member must be initialized by the Constructor or the client Code that Creates the Constructor.

Bcz, Struct is a Value type.
It is stored on the Stack where a Pgm works directly with it's Contents.

Quicker to access than indirectly accessing data through → a Pointer to the heap

The CLR Copies a Struct and Sends the Copy of the receiving method.
Also a Struct faces the boxing and Unboxing issue of Value types.

When deciding whether to use a Struct to represent your data, Consider the fact that types that naturally have Value Semantics

↳ Objects that directly Contain their Value as opposed to a reference to a Value.

Defining Structures :-

Syntax

```
[attribute][modifier] Struct identifier [: interfaces]
{struct-body
 - - -
}
```

Eg:-

```
public Struct DressShirt
{
 public float collarSz;
 public int SleeveLn;
 // Constructor
 public DressShirt ( Float Collar, int Sleeve)
 {
    This. CollarSz = Collar;
    this. SleeveLn = Sleeve;
 }
}
```

The Syntax clearly resembles a class, In fact, replace Struct with class and the Code Compiles. It has a public modifier that permits access from any assembly.

No interface are Specified, although a Struct Can inherit from a Interface - So the Struct is not required to implement any Specific methods.

Structure Versus Class:-

Many developers Select a class To prepresent data and the Operations performed on.it. However, there are cases where a Struct is a better Choice, as evidenced by its use in the Framework Class Library to represent Simple data types.

Comparison of Structure and Class:-

	Structure	class.
Default Access level of the Type	Internal	Internal
Default Access level for data Members	public	Private
Do-Properties & Methods	Private	Private.
Value or reference Type	Value	Reference type
base for a new types	No	Yes
Implement Interfaces	Yes	Yes
Raise and handle Events	Yes	Yes.
Scope of Members	Structure	class.
Instance of Initialization	Constructor - with or w/o Parameter A struct a cannot contain a custom Parameterlesss Constructor	Constructor - with/w. Parameter
· nested Destructor	· No	Y YES (Final)

K

Exception Handling:-

One of the most important aspects of Managing an object

To Ensure that it's behavior and interaction with the System. does not Result in a Pgm terminating in Error.

This Means That an application must deal gracefully with any runtime errors. that occur.

Whether they Originate from faulty application Code. The framework Class Library (or) H/w faults

.NET Provides developers with a technique Called Structured Exception Handling (SEH) to deal with Error Conditions.

An Exception Object is Created and Passed along With Program Control to Specially designated Section of Code.

In .NET terms, the exception Object is thrown from One Section of Code to another Section that Catches It.

SEH - Advantages:-

-> The Exception is Passed to the application as an object.

-> An Exception thrown and an application does not Catch it.

The CLR terminates the appln.

-> The Exception handling and detection Code does not have to be located where the error occurs.

-> Exception are used ~~&~~. exclusively and Consistently at both the appln and System level. All Methods in the .NET Framework throw Exeception when an error occurs.

SYSTEM. Exception Class

System. Exception is the base class for two generic Sub classes - SystemException, and ApplicationException - from which all Exception objects directly inherit.

.NET Framework Exception (such as IOException & Arithmetic Exception) derive directly from IOException. whereas Custom application Exception should inherit from Appln Exception.

The Sole Purpose of these classes to Categorise Exceptions. bcz they do not add any Properties or methods to the base System.Exception Class.

SEH - Advantages:-

-> The Exception is passed to the application as an object.

-> An Exception thrown and an application does not catch it. The CLR terminates the appln.

-> The Exception handling and detection code does not have to be located where the error occurs.

-> Exception are used exclusively and Consistently at both the appln and System level. All Methods in the .NET Framework throw Exoception when an error occurs.

SYSTEM. Exception Class

System. Exception is the base class for two generic sub classes - SystemException, and ApplicationException - from which all Exception objects directly inherit.

.NET Framework exception (such as IOException & Arithmetic Exception) derive directly from IOException. whereas Custom application Exception should inherit from Appln Exception.

The sole purpose of these classes to Categorise Exceptions. bcz they do not add any Properties or methods to the base System. Exception Class.

Writing Code to Handle Exceptions:-

C# uses a try/catch/finally Construct to implement exception handling.

When an exception occurs. The System Searches for a Catch block that can handle the Current type of Exception.

Three blocks:-

→ Try Block

→ Catch Block

→ finally Block

The try Block:-

The coode inside the try block is referred to as a guarded region bcz it has associated catch or finally blocks to handle possible Exception (or) cleanup duties.

Each try block must have at least One accompanying Catch (or) Finally block.

The Catch Block

The catch Block Consists of the keyword Catch followed by an expression in Paranthesses Called the Exception Filter.

↳ that indicates the type of Exception too which it responds.

The finally Block:-

The finally block is exeruted whether or not an Exception occurs and is a Convonient place to perform any Cleanup Operations such as Closing Files (or) db Connections

~~Exempte~~

Syntax

```
try {- - -
     - - -
   }
  catch (exception name)
    {
      - - -
      - - -
    }
 catch (exception name)
    {- - -
     - - -
    }
 Finally {- - -
          - -
    }
```

Example

```
   using System
   public class TestExcep
    {
     public Static int calc (int j)
      {
       return( 100 / j );
      }
    Class MyAPP
     { public Static void Main ()
       {
        TestExcep exTest = new TestExcep();
        try
         { // Create divide by zero in called Method
          int dzero = TestExcep . Calc(0);
         } Console . writeline ("Result; {0}", dzero);
```

```
Catch (Divido By Zoro Exception ex)
  { Console. writoline ("{0} \n {1} \n", ex.Message, ex. Sourre);
    Consolo. writolino (ex. Target Site. ToString());
    consolo.writolino (ex. Stack Trace);
  }
Catch (Exoption ex)
  {
    Consolo. writeline ("General" +ex. Message);
  }
finally
  { consol. writeline ("Cleanup occurs here.");
  }
 }
}
```

Unhandled Exceptions:-

Unhandled exceptions occur when the CLR is Unable to find a Catch filter to handle the Exception.

The default result is that CLR will handle it with it's Own methods.

The code can be implemented to recognise whether it is dealing with a debug or release version. and respond accordingly.

You should log the error and provide a Meaningful Screen that allows the user to end the Pgm.

Unhandled Exception in a Windows Forms Appln

We can now use those techniques to register our own callback method that processes any unhandled exceptions thrown in the Windows appln.

When an Exception occurs in Windows. the appln's OnThread Exception method is ultimately called.

You can override this by creating and registering your own method ~~for the~~ that matches the Signature of the System.Threading.ThreadExceptionEventHandler delegate.

MyUnhandledMethod is defined to handle the Exception and must be registered to receive the callback.

Ex:-
```
Using System.Diaganostics;
Using System.Windows.Forms;
Using System.Threading;
Public class UnForgiven.
{   // class signature matches ThreadExceptionEventHandler
    Public Static void MyUnHandled Method
        (Object Sender, ThreadExceptionEventArgs. e)
    {
    #if DEBUG
        // Stmnts for debug mode
        // Display trace and start the Debugger.
    MessageBox.Show ("Debug", "+e.ToString());
# else
    // Stmnts for release mode
    // Provide O/P to user and log errors.
    messageBox.show ("Release:"+e.ToString());

#endif
 }
}
```

System.IO : classes to Read and Write Streams of Data :-

The System.IO namespace contains the Primary classes used to move and process Streams of data.

The data Source may be in the form of text Strings. or raw bytes of data Coming from a n/w or device on an I/o Port.

Classes derived from the Stream class Work with raw bytes. those derived from the TextReader and TextWriter classes operate with char & text Strings

The Stream class and look at how its derived classes are used to Manipulate byte Streams of datdombU

How data is more Structured text Format is handled using the TextReader and TextWriter classes.

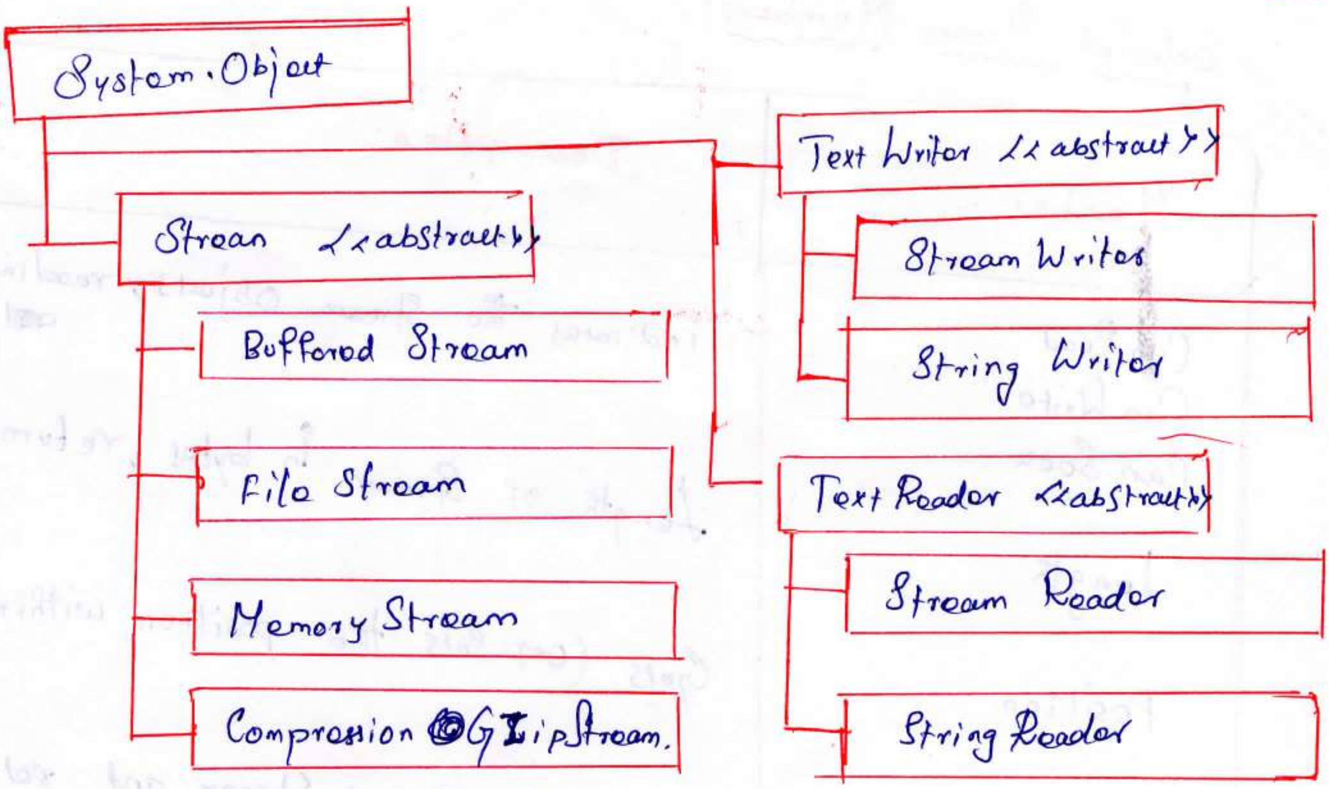


Fig:- Selected System.IO classes.

The Stream Class :-

This class define the generic Members Working with raw byte Streams

It's Purpose is to abstract data into a Stream of bytes independent of any underlying data devices

The class members support three fundamental areas of Operations reading, Writing and Seeking (identify the Current byte position within a stream)

Selected Stream Members:-

Member	Description
Can Read Can Write Can Seek	indicates the Stream objects, reading, writing and Seeking
Length	Length of Stream in bytes, returns Long type
Position	Gets (or) Sets the position within Current Stream
close ()	Closes the Current Stream and releases resources associated with it.
Flush()	Flushes data in buffers in the underlying device - for eg:- a file (sudden)
Read (byte array, Offset, count) Read Byte ()	Reads a Sequence of bytes from the Stream Read Byte reads one byte.
Set Length()	Sets the length of the Current Stream. It can be used to Extend / truncate Stream
Seek	Sets the position within the Current Stream
Write (byte Array, Offset, Count) Write Byte()	Writes a Sequence of bytes or One byte.

File Stream:-

A File Stream Object is Created to Process a Stream of bytes associated with a bauuing Store

a term used to refer to any Storage Medium such as disk or Memory.

code Sgmnt used for reading and writing bytes:-

```
try
{
  //Create FileStream Object
  FileStream fs = new FileStream (@"c:\artists\log.txt",
                  Filemode.Open or Create, FileAccess.ReadWrite);
  byte[] alpha = new byte[6] {65, 66, 67, 68, 69, 70};
                                                        //ABCDEF
  // Write Array Of bytes to a file
  // Equivalent to: fs.write (alpha.0, alpha.length);
  ~~fo~~ foreach ( byte b in alpha)
  {
    fs.WriteByte (b);
  }
  // Read bytes from file
  fs.position = 0;            //move to beginning of file
  ~~Cor~~ for ( int i=0; i < fs.length; i++)
      Console.write ((char). fs.Readbyte ()); //ABCDEF
  fs.close();
Catch ( Exception ex)
  { Console.Write(ex.message);
}
```

Creating a File Stream:-

The FileStream class has several Constructors. The most useful ones accept the path of the file being associated with the object and Optional Parameters that define file mode, access rights & sharing rights.

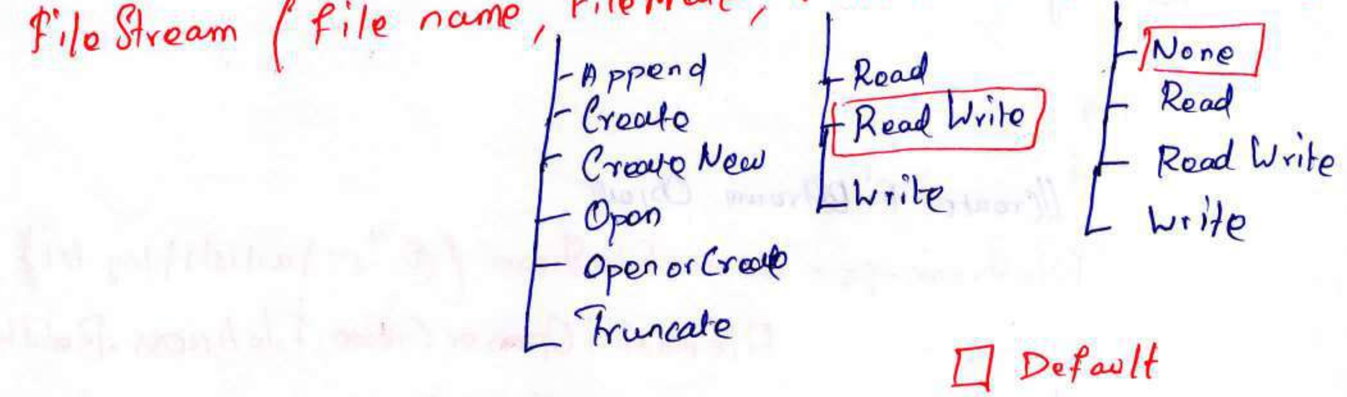


Fig:- Options for FileStream Constructors.

File Mode Enumeration Values:-

Value	Description
Append	Opens an existing file or Creates a new One.
Create	Create a new file. An existing file is overwritten.
Createnew	Create a new file. An Exception thrown if the file already exists.
Open	Opens an Existing file.
Open Or Create	Opens a file if it exists, Otherwise Creates a new one
Truncate	~~Opens a file, if it exists~~. Opens an existing file, removes it contents, and Position the file pointer to the beginning of the file.

MEMORY STREAMS:-

This class is used to Stream bytes to and from Memory as a substitute for a temporary external physical Store.

It reads the Original file into a memory Stream and then writes this to a FileStream using the WriteTo method.

```
FileStream fsIn = new FileStream(@"c:\manet.bmp",
                  FileMode.Open, FileAccess.Read);
FileStream fsOut = new FileStream(@"c:\manetcopy.bmp")
                  FileMode.OpenOrCreate, FileAccess.Write);
MemoryStream ms = new MemoryStream();
// input imgByte byte-by-byte and Store in Memory Stream
int imgByte;
while ((imgByte = fsIn.ReadByte()) != -1)
{
   ms.WriteByte((byte)imgByte);
}
ms.WriteTo(fsOut);  // copy image from memory to disk
byte[] imgArray = ms.ToArray(); //convert to array of bytes
fsIn.Close();
fsOut.Close();
ms.Close();
```

BUFFERED STREAMS:-

One way to improve I/O performance is to limit the number of reads and writes to an external device. Particularly when small amounts of data are Involved.

Buffers have long offered a Solution for collecting Small amounts of data into larger amounts that could then be Sent more efficiently to a device.

The Buffered Stream Object Contain a buffer that Performs this role for an underlying Stream. (FileStream)

Buffers are commonly used to improve Performance when reading bytes from an I/o Port or n/w.

The code Consists of a loop in which Fill Bytes (Simulating an I/o device) is called to return an array of bytes.

These bytes are written to a buffer rather than directly to the file. When FileBuffer is closed any remaining bytes are flushed to the FileStream fsOut.

```
Private void SavedStream()
{
    Stream fsOut1 = new FillStream(@"c:\captured.txt",
        FileMode.Open or Create, FileAccess.Write);
BufferedStream fileBuffer = new BufferedStream (fsOut);
byte[] buff;
bool readmore = true;
While (readmore)
{                               // Get array of objects.
  buff = FillBytes();
for (int j=0; j< buff[16]; j++)
{
   fileBuffer.WriteByte ( buff [j]);      //stores bytes in buffer
}                                         //indicates no more data
if (buff[16] < 16) readmore = false;
}
fileBuffer.close();
fsOut11.Close();
}
Private static byte[] FillBytes()
{
 Random rand = new Random();
 byte [] r = new Byte[17];
// store random numbers to return in array
 for (int j=0; j<16; j++)
 {
     r[j] = (byte) rand.Next();
                          // Arbitrary end of Stream Value
 if ( r[j] == 171)
 {
   r[16] = (byte)(j);
 return r;
 }
}
 System.threading.Thread.Sleep(500);
 return r;
}
```

Using StreamReader and StreamWriter to Read and Write

lines of text:- StreamWriter and StreamReader are designed to work with the text rather than raw bytes.

The abstract TextWriter and TextReader classes from which they derive define methods for reading and writing text as lines of characters.

Writing to text file:-

→ Writeline works only with Strings and automatically appends a new line (carriage return\line feed)

→ Write does not append a newline charactor can write Strings as well as the textual representation of any basic data type (int32, Single, and So on) on the text Streams.

The StreamWriter objet is Created using one of Several Constructors

Syntax (partial list)

```
public StreamWriter(string path)
  ,,       ,,       (String  s)
  ,,       ,,       (String path, bool append)
  ,,       ,,       (S  ,,   ,,   ,,   ,,  Encoding encoding)
```

Parameter:

path - Path and name of file to be Opened

s - Previously Created Stream objet - a FileStream

append - Set to true to append data to file
False overwrites.

encoding - Specifies how Charators are encoded as they are written to a file. The default is UTF-8
Stores char in minimum no. of bytes required.

Roading from a Toxt File:-

A StroamRoader object is usod to toxt from a file. much liko StroamWritor an instanco of it can be Croatod from undorlying Stream Object.

Slactod StroamRoader Methods :-

Mombor	Description
peek ()	Roturns the noxt available Charactor w/o Moving the position of the Roader.
Road () Road (char buff, int ndx, int count)	Roads noxt charator (Road ()) from a Stroam or roads next Count Char into a Charactor array boginning at ndx.
Readline	Return a String Comprising one lino of toxt
Road To End ()	Roads all charactors from the Current Position to the ond of the ToxtRoader.

String Writer and String Roader:-

The StroamWritor and StroamRoador. The Main dif is that these Stroams are Storod in momory. rather than in a file.

Those two classos do not require a lot of

discussion.

System.IO : Directories and Files:-

The System.Io namespace includes a Set of System-related classes that are used to Manage files and directories.

Directory and Directory Info Contain members to create, delete, and query directories.

File and FileInfo provide Static and instance Methods for working with files.

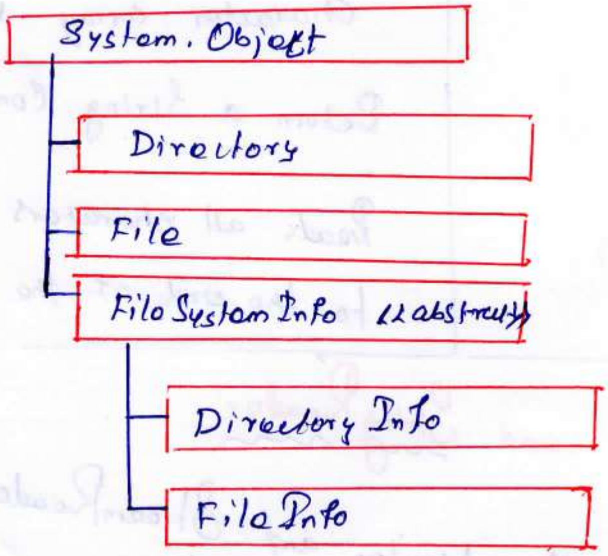


Fig: Director and File Classes

File SystemInfo :-

The FileSystemInfo class is a base class for DirectoryInfo and FileInfo.

It defines a range of members that are used primarly to provide information about a file (or) directory.

Two methods :-
→ Delete - delete a file or directory
→ Refresh - checking the directory and file attributes.

```
//Directory Info
    String dir = @"c:\artists";
    DirectoryInfo di = now DirectoryInfo(dir);
    di.Refresh();
    Date Time IoDate = di.CreationTime;
    Console.Writeline ("{0:d}", IoDate).
// File Info
    String file = @"C:\artists\manet.jpg";
    FileInfo fi = now FileInfo (file);
    if (fi.Exists)
    {
        fi.Refresh();
        IoDate = fi.CreationTime;
        Console.Writeline ("{0:d}", IoDate);      // 07/04/2022
        Console.Writeline (fi.Name);              // manet.txt
        Console.writeline (fi.extension);
        FileAttributes attrib = fi.Attributes
        Console.writeline ((int) attrib);         //32
        Console.writeline (attrib)                // Archive
    }
```

Working with Directories Using the DirectoryInfo, Directory and path classes:-

When Working with directories, you usually have a b/w using the instance methods of DirectoryIn[fo] or the Corresponding static methods of Directory.

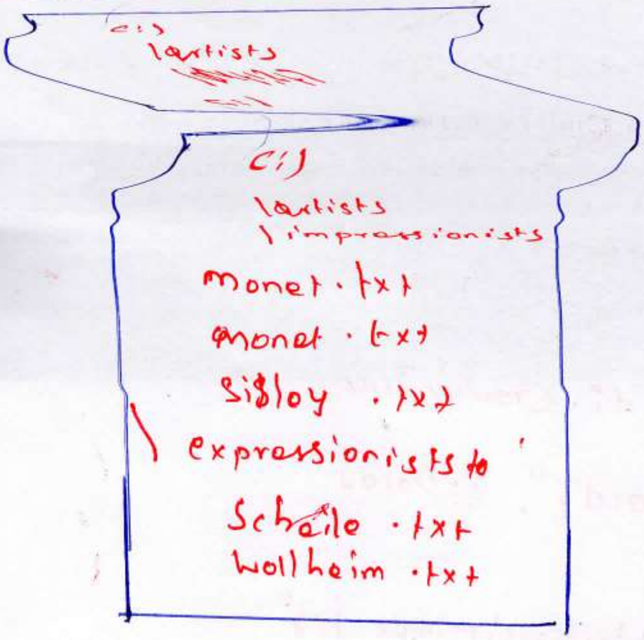
~~Ex~~ ~~using~~ Let's look at Some examples of usin[g] both Static and instance methods to Manipulate and List

Directory Members.

Fig:- Screenshot

Directory Members.

Fig:- Screen



Create a Subdirectory:-

This code adds a Subdirectory named cubist below expressionists:-

```
    // Directory Static Method to Create directory
String newPath = @"C:\artists\expressionists\cubists";
if (!Directory.Exists(newPath))
    Directory.CreateDirectory(newPath);
// DirectoryInfo
  String curpath = @"c:\artists\expressionists";
        di = new DirectoryInfo(curPath);
   if (di.Exists) di.CreatedSubdirectory(newpath);
```

Delete a Subdirectory:-

This code deletes the Cubists Subdirectory just Created'.

```
String newPath =@"c:\artists\expressionists\cubists";
// Directory
  If (Directory.Exists(newPath)) Directory.Delete(newPath);
// The following fails bcz the directory still contains.
  Directory.Delete(@"c:\artists\expressionists");
// The following succeeds bcz true is passed to the Method.
  Directory.Delete(@"c:\artists\expressionists", true);

// DirectoryInfo
  DirectoryInfo di = new DirectoryInfo(newPath);
  If (di.Exists) di.Delete();
```

List Directories and Files:-

This code defines a method that recursively loops through and lists the Subdirectories and Selected files on the c:\artists Path.

It uses the Static Directory methods GetDirectories and GetFiles.

Both of these return String Values.

GetFiles returns a full Path name. The Static Path.GetFileName method used to extract the file name and extension from the Path.

```
foreach (string filename in
           Directory.GetFiles(sourceDir, "s*.*"))
  {
   Console.writeline(" "+Path.GetFileName(filename));
  };
```

```
foreach (string SubDir in
        Directory.GetDirectories(SourceDir))
    {
  ShowDir(SubDir, recursionLvl+1);     //Recursive call
 }
```

The Method is called with two parameters; a DirectoryInfo object that encapsulates the path and an Initial depth of 0.

Working with files using the FileInfo and File classes:-

The FileInfo and File classes are used for two purposes. To provide descriptive information about a file and to Perform basic file Operation.

The classes include methods to copy, move and delete files. as well as open files for reading and Writing

This short Segment uses a FileInfo object to display a file's properties, and the Static File.copy method to copy a file.

```
String fnamo = @ "c:\artists\ impressionists\dogas.txt";
// Using the FileInfo classes to print filo information.
FileInfo fi = new FiloInfo (fname);   // creato fileInfo Object.
if (fi.Exists)
{
Console.Write ("Length : {0} \n Namo: {1} \n Directory: {2}",
        fi.Length , fi.Name, fi.DirectoryName);
}
// uso File class to copy a filo to another directory
if (File .Exists (fname))
{
  try
  {
      //Exception is thrown if filo exists in target directory
      // Source, destination , Overwrites = false)
   File. Copy (fnamo , @"C:\artists\ 19th century\dogas.txt", false);
  }
  catch (Exception ex)
  {
   console. Write (ex. message);
  }
}
```

Using FileInfo and File to Open Files:-

The File and FileInfo classes offer an alternative to Creating FileStream, StreamWriter and StreamReader objects directly.

FiloInfo methods used to Open a file, the Static File Methods are identical except that their first Parameter is always a String containing the name or path of the file to Open.

Selected FileInfo Methods for Opening a File

Member	Returns	Description

Open(mode)
Open(mode, access)
Open(mode, access, Share)
	FileStream	Opens a file with access and Sharing privileges.
		3 overloads
		mode, FileAccess & File Share

Create()	FileStream	Create a file and returns a FileStream obj
OpenRead()	FileStream	Opens the file in read mode.
OpenWrite()	FileStream	Opens the file in Write mode.
AppendText()	StreamWriter	Opens the file in append mode. if file is doesn't exist ~~if~~ it created StreamWriter(String, true)
CreateText()	StreamWriter	Opens a file for writing. If the file exists it's contain Over Written. StreamWriter(String, false)
OpenText()	Stream ~~Writer~~ Reader	Opens the file in read mode, Equivalent to StreamReader(String) StreamReader(String)

The FileInfo.Open method is the generic and most flexible way to open a file:

Public FileStream Open(FileMode mode, FileAccess access, FileShare Share)

——— x x x ———

UNIT - III

BUILDING WINDOWS FORMS and CONTROLS

Programming a Windows Form

All Windows Forms programs begin execution ~~of~~ at a designated Main Window.

This Window is actually a Form object that inherits from the System.Windows.Forms.Form class.

The Initial Window is displayed by passing an instance of it to the Static Application.Run method.

Building a Windows Forms Application by Hand

Let's Create a Simple Windows application using a text Editor and the C# Compiler from the Command line.

Eg:- Pgm → Consists of a Single Window with a button that pops up a Message the button is clicked.

Simple Exercise demonstrates how to Create a form, add a Control to it. and Set up an Event handler to respond to an Event fired by the Control.

Eg:- Pgm

```
Using System
Using System.Windows.Forms;
Using System.Drawing;
Class My WinApp.
{
  Static Void Main()
  {
   // Create Form and invoke it
   Form mainForm = new SimpleForm();
   Application.Run(mainForm);
  }}
   // User Form derived from base Class Form
  Class SimpleForm.Form
  {
   private Button button1;
   public SimpleForm()
   {
    this.Text = "Hand Made Form";
  // Create a button Control and Set Some attributes
    button1 = new Button();
    button1.location = new point(96,112);
    button1.Size = new Size(72,24);
    button1.Text = "Status";
    this.Controls.Add(button1);
  // Create delegates to call routine when click occurs.
    button1.click += new EventHandler(button1_Click);
  }
Void button1_click(Object Sender, EventArgs e)
    {
     MessageBox.Show("Up and Running");
    }
  }
```

The Command line Compilation requires providing a target Output File and a reference to any required assemblies. We include the System.Windows.Forms assembly that Contains the necessary Winforms classes.

The Source Code as Winform.cs and enter the following stmnt at the Command Prompt.

```
csc /t:Winform.exe/r:System.Windows.Forms.dll Winform.cs.
```

After it Compiles. run the pgm by typing Winform;

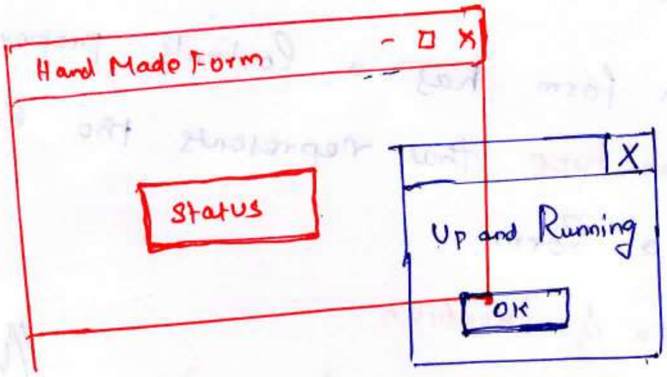


Fig: Windows application.

The code breaks logically into three Sections:-

-> Form Creation

-> Create Button Control

-> Handle Button click Event.

The Command line Compilation requires providing a target Output File and a reference to any required assemblies.

We include the System. Windows. Forms assembly that Contains the necessary Winforms classes.

The Source Code as Winform. CS and enter the following Stmnt at the Command Prompt.

```
csc /t : Winform . exe / r: System. Windows. Forms . dll Winform. cs.
```

After it Compiles. run the pgm by typing Winform;

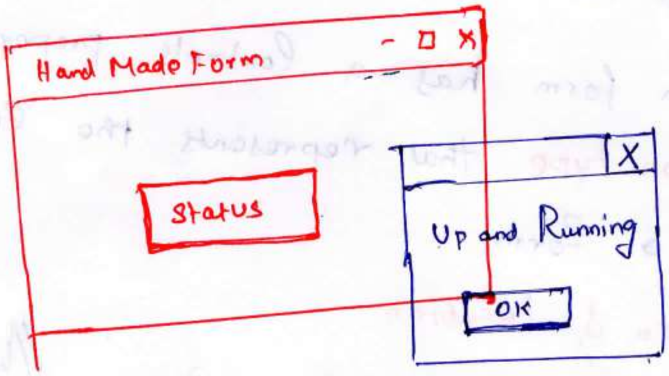


Fig: Windows application.

The code breaks logically into three Sections:-

→ Form Creation

→ Create Button Control

→ Handle Button click Event.

1) Form Creation:-

The parent form is an instance of the Class SimpleForm, Which inherits from Form and defines the Form's Custom features.

The pgm is invoked by passing the instance to the Application.Run method.

2) Create Button Control:-

A Control is placed on a form by Creating an instance of the Control and adding it to the Form.

Each form has a Controls property that returns Control.Collection type that represents the Collection of Controls Contained on a Form.

Size & location

```
button1.Size = new Size (72,24);          //width, height
button1.location = new point (96,112);    //x,y
```

3) Handle Button Click Event:-

Event Handling requires providing a method to respond to the Event and Creating a delegate that invokes the method when the Event Occurs.

The delegate associated with the Click event is Created with the following Statement.

```
button1.click += new EventHandler (button1_Click);
```

This stmnt creates an instance of the Built-in delegate EventHandler and registers the method button1_click with it

Windows.Forms Control Classes:-

The previous examples have demonstrated how a Custom form is derived from the Windows.Forms.Form class.

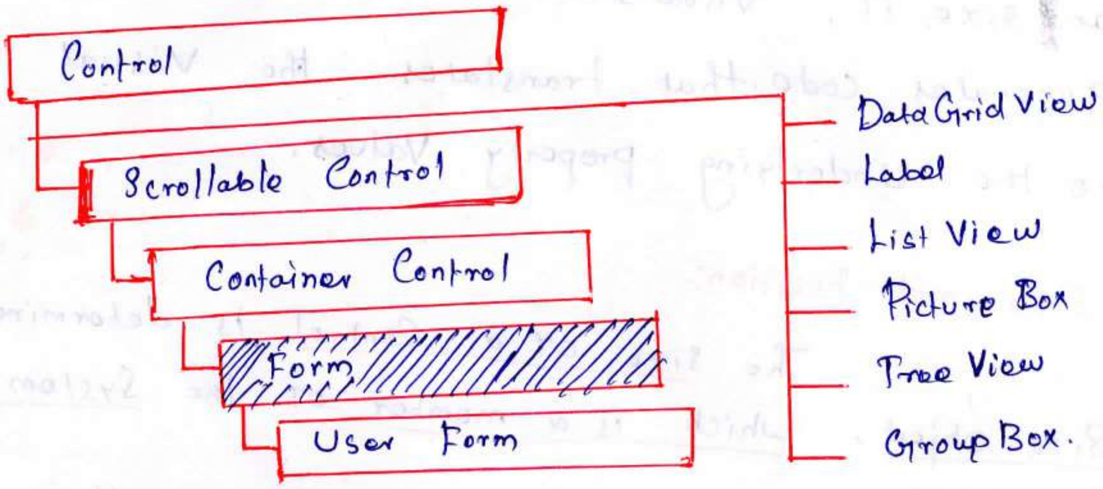


Fig:- Windows Forms class hierarchy

Control Class:-

A Form is a Container Control, and the generic Members of the Control Class can be applied to it just as they are to a simple Label (or) TextBox.

System.Windows.Forms.dll Contains more than fifty Controls that are available for use on a Windows Form.

All Controls share a core set of inherited Properties, Methods & Events. These members allow you to size and position the Control and adorn with text, colors, and style features and responds to Keyboard & Mouse Events.

* (7 ctrl properties - image copy)

Working with Controls:-

When you drag a control onto a form, positionit, and size it, Visual Studio.NET (VS.NET) automatically generates code.that translates the Visual design to the Underlying property Values.

Size and Position:-

The size of a Control is determined by the System.Drawing Size Object, which is a member of the namespace.

```
button1.Size = new Size(80, 40)   //(width & height
button2.Size = button1.Size   // Assign Size to Second button
```

A control can be resized during runtime by assigning a new Size Object to it. This Code Snippet demonstrates how the Click event handler method can be used to Change the Size The button when it is clicked:

```
Private Void button1_Click(object Sender, System.EventArgs e)
{
  MessageBox.Show("Up and Running");
  Button button1 = (Button) Sender   // cast Object to Button
  button2.Size = new Size(90,20)  // Dynamically resize.
```

The System.Drawing.Point Object can be used to assign a Control's location. It's arguments set the X & Y Conordinates. button1.location = new point(20,40); //(x,y) Conordinates

How to Anchor and Dock a Control:-

The Dock Property is used to attach a Control to One of the edges of it's Container, By default ~~Status~~ Most Controls have docking set to none

Some Exception are the StatusStrip/StatusBar that is Set to Bottom and the ToolStrip/ToolBar that is Set to Top

DockStyle Enumeration are Top, Bottom, Left, Right and Fill.

The Fill options are attaches the Control to all Four Corners and resizes it as the Container is resized.

To attach a TextBox to the top of a form use.

TextBox.Dock = DockStyle.Top;

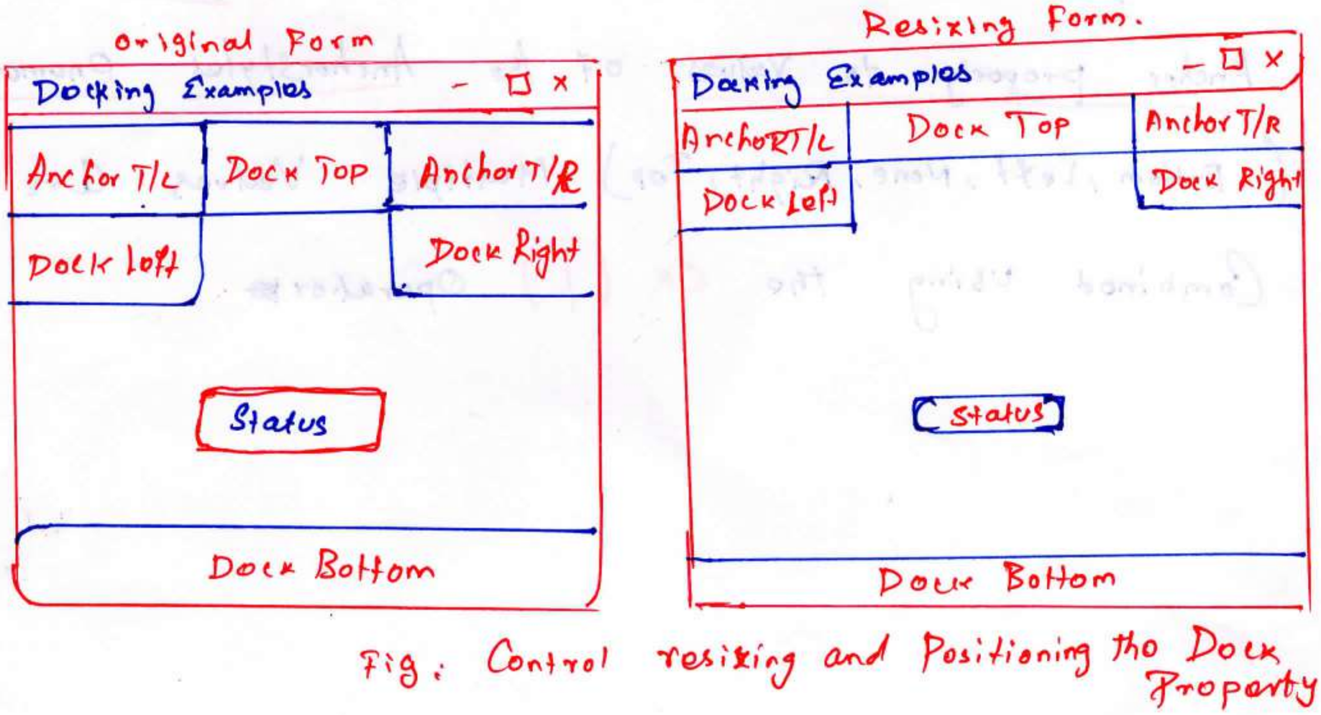
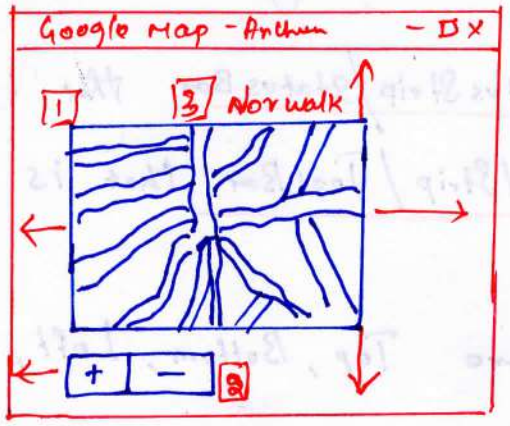
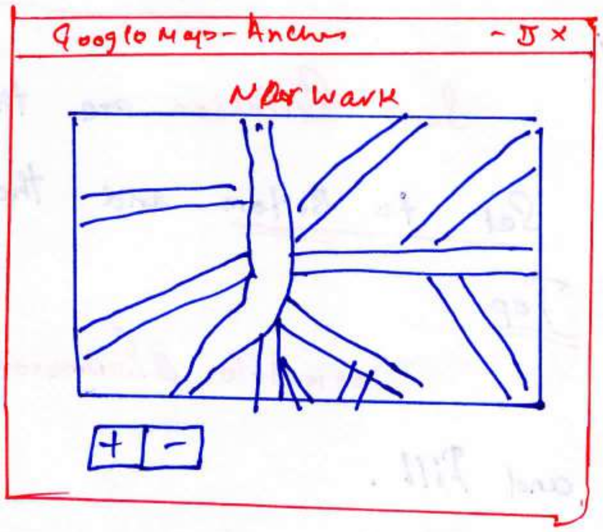


Fig: Control resizing and Positioning tho Dock Property

The Anchor Property allows a Control to be placed in a fixed Position relative to a Combination of the top, Left, Right (or) bottom edge of its Container.





1 - anchor Left, Right, Bottom

2 - anchor Left, Bottom

3 - anchor Top

```
btnPanel.Anchor = (AnchorStyles.Bottom|AnchorStyle.left);
```

The code defines a Controls ~~ab~~ anchor position Sets the Anchor property to Values of the AnchorStyles Enumeration (Bottom, Left, None, Right, Top) Multiple Values are Combined using the OR (|) Operator.

Control Events:-

When you push a key on the keyboard or Click the Mouse, the Control that is the target of this action fires an Event to indicate the Specific action that has Occured.

A registered event handling routine then responds to the event and formulates what action to take.

The First Step in handling an Event is identify the delegate associated with the Event. You must then register the Event handling method with it. and make Sure the method's Signature Matches the Parameters Specified the delegate.

Summarizes the information required to Work with mouse and keyboard triggered Events.

Control Events

Event	Built in Delegate/ Parameters	Description.
Click DoubleClick; MouseEnter,	EventHandler(Object Sender, EventArgs e)	Events triggered by Clicking, double clicking or Moving the mouse.
MouseEnter MouseLeave MouseHover MouseWheel MouseDown MouseUp MouseMove	MouseEventHandler(Object Sender, MouseEventArgs e)	Events triggered by mouse & mouse button [illegible]
KeyUP KeyDown	Key Event Handler(Object Sender, KeyEventArgs, e)	Events triggered by key being raised or Lowered.
KeyPress	KeyPressEventHandler(object Sender, KeyPressEventArgs e)	Event triggered by Pressing any key.

Handling Mouse Events:-

The familiar Click and DoubleClick events, all Windows Forms Controls inherit the MouseHover, MouseEnter. and MouseLeave events. The latter two are fired when the Mouse enters & Mouse leaves the Confines of a Control.

eg:- The changes the background color on a textbox When a mouse passes Over it.

The following Code sets up delegates to call

- OnMouseEntor and OnMouseLeave to Perform the background Coloring:

```
TextBox  UserID = new TextBox();
UserID. MouseEntor += new EventHandler (onMouseEntor);
UserID. MouseLeave += new Event Handler (onMouseLeave);
```

The Event Handler methods match the Signature of the EventHandler delegate and cast the Sender Parameter to Control type to access its Properties.

```
Private Void onMouseEntor (Object Sender, System. EventArgs e)
  {
   Control ctrl = (Control) Sender;
    Ctrl . BackColor = Color. Bisque;
  }
Private void onMouseLeave (object Sender, System. EventArgs e)
  {
   Control ctrl = (control) Sendor;
    Ctrl . Backcolor = Color. White;
  }
```

Properties of MouseEventArgs

Property	Description
Button	Indicates which mouse button was pressed.
Clicks	Number of clicks since last Event
Delta	No. of detents the Mouse wheel rotates. +ve no - Fwd, -ve no Bwd. motion
X Y	Mouse Co-ordinates relative to the Container upper loft Corner. This is equivalent to the Ctrl Mouseposition

Eg: pg 47

Handling Keyboard Evonts:-

Keyboard Events are also handled by defining a delegate to call a Custom Event handling Method Two argumont Passed to the Event handler. The Sender argumont identifies the Objact. Second argumet Contains fields describing the Event. For the KeyPress Event.

This type contains a Handled field that is Set to true by the PressEventArgs type. to indicates the processed the Event. It's Other Property is KeyChar, which identifies the key that is pressed.

Key char is useful for restricting the input that a field accepts.

```
Private void onKeyPress (objaet Sendor, KeypressEventArgs e)
{
if (! char . IsDigit(e. keychar) ) e. Handled = true ;
}
```

The KeyPress Event is Only fired for printable Charalter keys. it ~~gg~~ ignores non-Charaeter keys Such as Alt or Shift.

KeyEventArgs Properties:-

Member	Description
Alt, Control, Shift	Boolean Value that indicate whether Alt, Control (or) Shift Key was pressed.
Handled	Boolean Value that indicates whether an Event Was handled
Keycode	Returns the key code for the Event
keyData	Return the key data for the Event.
Modifiers	Indicates which Combination of Modifier keys (Alt, Ctrl & Shift) was Pressed.

The Form Class:-

The Form Object inherits all the mombors Of the Control class as wall as Scrollable Control class. it adds a large number of properties that enable it to Control it's appearance, work with child Forms, Create modal Forms, display Memus and Interact with the desktop via tool and Status bar.

Fig:- Screenshot

Setting a Form's Appearance:-

The Four Properties. Fig:- Control which buttons and icon are present on the top border of a form

Icon Property Specifies the .ico file used as the icon in the Left Corner

Control box value determines whether icon and close button are displayed (true), or not displayed (false).

MaximizeBox & MinimizesBox determine the. Whether their associated button appear.

↳ Maximize the Modal form in order to prevent a User form maximizing.

↳ hiding the Underlying Parent Form

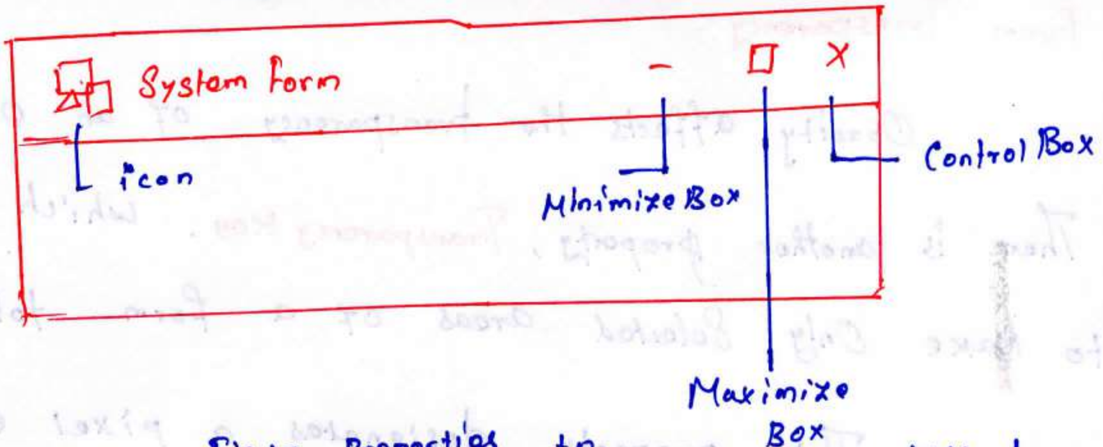


Fig:- properties to Control what appears on the title bar.

Form Opacity:-

A Form's Opacity property determines it's Level of transparency. Value ranges from 0 to 1.0

Anything Less than 1.0 results in Partial transparency that allows elements beneath the form to be Viewed.

Most Forms work best with a Value of 1.0 but adjusting Oapcity can be an effective way to display child (or) TOP MOST forms that hide an Underlying form.

Common approach is to Set the opacity of Such a form to 1.0 (when) it has focus. reduce the Opacity when it losses focus.

```
Void Form_Deactivate (Object Sender, EventArgs e)
  { this.Opacity = .8;  }
Void Form_ Activate ( Object Sender ; EventArgs e)
  { this.Opacity = 1;
            }
```

Form Transparency :-

Opacity affects the transparency of an Entire Form. There is another property, Transparency key. which can be used to make Only Selected areas of a form totally transparent.

This property designates a pixel color that is rendored as transparent When the form is drawn.

The effect is to Create a hole in the Form, that makes any area below the form visible. Infact, If you click a transparent area, the event its recognized by the form below

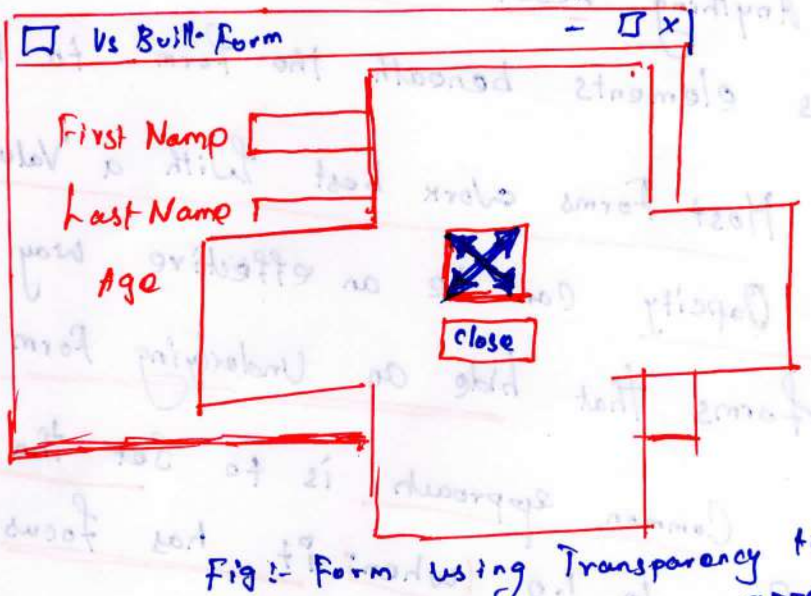


Fig :- Form. using Transparency to Create Irregular apperance.

To Create the form in figure 6-7, Place Panel Controls in each Corner of Standard form and Set their BackColor property Red. The form is Created and displayed using the Code.

```
Custom Form . myform = new CustomForm();
myForm . TransparencyKey = Color.Red;
myForm . FormBorderStyle = FormBorderStyle. None;
myForm . show();
```

Setting Form location and Sixe:-

The Initial location of a form is determined directly (or) Indirectly by it's StartPosition property.

Initial location is normally Set in the ~~form~~.

Form.Load Event Handllor

Eg:- loads the form 200 pixels to the rights of the Upper-left Corner of the Screen.

```
Private Void Opaque_Load (object Sondor, System EventArgs e)
{
this.Desktoplocation = new Point(200, 0);

}
```

The form's initial location can also be set by the form ~~of~~ that Creates and displays the form object.

```
Opaque OpForm = new Opaque();
OpForm.Opacity = 1;
OpForm.TopMost = true;    // Always display form on top
OpForm.StartPosition = FormStartPosition.Manual;
OpForm.DesktopLocation = new Point(10,10);
OpForm.Show();
```

This code Creates an instance of the form opaque and Set's it's TopMost property So that the form is always displayed on top of Other forms in the Same applicati

Displaying Forms:-

After a Main form is up and running. It can Create instances of new forms and display them in two ways: Using the Form.showDialog method (or) the Form.Show method inherited from the Control classes.

Form.showDialog displays a form as a Modal dialog box.

↳ are discussed at the end of this Section.

The life Cycle of a Modeless form:-

Action	Event triggered	Description
Form obj Created		Form's Constructor is called the initialize Component method is called to initialize the Form
Form displayed Form.show()	Form.load Form.Activated	The load Event is called First followed by Activated Event
Form activated	Form.Activated	This occurs when the User Select the form become a "active" form
Form deactivated	Form.Deactivated	Form is deactivated when it losses Focus.
Form Closed	Form.Deactivate Form.Closing Form.Closed	Form is closed by Executing Forms. Close (or) clicking On the forms close button

Creating and Displaying a Form:-

When one form Creates another Form. there are Coding requirements on both Sides

The Created form Must set up code in its Constructor to Perform Initialization and Create Controls.

Eg:- // Create a new form (or) give focus to existing One.

```
private void button1_Click (Object Sender, System.EventArgs e)
{
  if (Closed)
  {
    Closed = false;
    opForm = new opaque();
  // call OnOpClose when new form closes
    opForm.Closed += new EventHandler (OnOpClose);
    opForm.Show();          // Display a new form object
  }
  else {
    opForm.Activate();      // Give focus to form
  }
}
```

```
// Event handler called when Child form is closed

Private Void OnOpClose (Object Sender, System.EventArgs e)
{
  Closed = true;      // Flag indicating Form is closed
}
```

Form Activation & Deactivation :-

A form becomes active when it's is first Shown (or) later.

When the User clicks on it or move to it using an Alt + Tab key to iterate through the task bar.

```
[illegible] Deactivate ( objet Sender, EventArgs e)
[illegible]
        button1 . Text = "Resume";
Void  Form - Activate ( objet Sender, EventArgs  e)
  {  button1 . Enabled = true ;
  }
```

Closing Form :-

The closing Event occurs as a form is being closed and provides the last Opportunity to Perform Some Cleanup duties (or) provent the form from closing.

This Event Uses the CancelEventHandler ~~to~~ delegate to invoke event handling methods.

```
this. Closing += new CancelEventHandler ( Form_Closing);
Void Form_Closing (objet Sender, CancelEventArgs e)
{  if (MessageBox.Show ("Are you Sure you want to Exit ?", "",
          MessageBoxButtons . Yes & No ) == DialogResult . NO)
   {
      // Cancel the closing of the form
      e. Cancel = true;
   }
}
```

Message and Dialog Boxes:-

.NET Provides a Set of classes and enumeration that make it easy to Create a Message (or) dialog window to interact with a User.

MessageBox class and it's Versatile Show method. Other approach Create a custom form and invoke it with the form's ShowDialog method. Both of these methods Create modal forms.

Message Box

The Messagebox class uses it's Show method to display a message box that may Contain text, buttons and Events an icon. The show method includes these Overloads.

Syntax:-

```
Static DialogResult · Show (String msg)
Static DialogResult · Show (String msg, String caption)
Static DialogResult · Show (String msg, String Caption,
                     MessageBox Buttons buttons)

Static DialogResult · Show (String msg, String caption.
          MessageBoxButtons buttons, MessageBox Icon Icon,
          ,,    ,, DefaultButton defBtn)
```

DialogResult. The method returns one of the enum members Abort, Cancel, ignore, NO, None, OK, Retry. & Yes.

Show Dialog

The showDialog method permits you to Create a Custom form that is displayed in modal mode.

If it is useful when you need a dialog form to displays a few Certain field of information.

```
Private void buttonOK_Click(object Sender, System.EventArgs e)
{ this.DialogResult = DialogResult.OK;
}
Private void buttonCancel_Click(obj Sender, Sys.EventArgs e)
{ This.~~to~~Dialog Result = DialogResult.Cancel;
}
```

6.4 Working with Menus

Menuitem Properties:-

The .NET Menu System is designed with the Utilitorian philosophy that the Value of a thing depends on Utility.

It's Menu item is not a thing of beauty, but it works. Hore Some more useful properties:-

- → Enabled
- → Checked
- → Radiocheck
- → BreakBar
- → Shortcut

Enablod

↳ setting this to false, grays out tho button and Maxos it Unavailable.

Checkod

↳ Places a Chockmark besides tho Menu item text

Radiocheck

↳ Places a radio button beside tho Menu item text. Checkod must also be true.

BreakBar (or) Break

↳ Setting this to true places tho Menu item in a new column.

Shortcut

↳ Defines a shortcut key from one of the Shortcut enum members.

Theso members represent a key or key Combination that causes tho Menu item to be Selected when the keys are Pressed. (Such as Shortcut. Alt F10)

Context Menus

In addition to the Mainmenu and MenuItem Classes that have been discussed, there is a ContextMenu Class that also inherits from the Menu Class

The Context Menu Class is associated with individual Controls and is used most often to provide a Context Sensitive pop-up menu when the User right-Click on a Control.

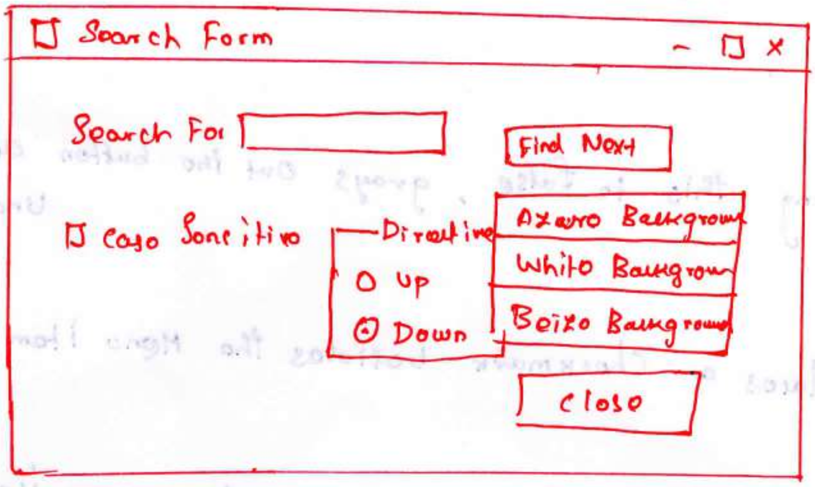


Fig: Context Menu

Constructing a Context Menu

Creating a Context Monu is similar to Creating a MainMonu. If using VS. NET you drag a Context Menu Control to the Form and visually add Monu items. If Coding by hand. you Create an instance of the Context Monu Class and add Menu items using the MenuItems. Add method.

Eg:-

```
Private  ContextMenu  ContoxtMonu1;   // Context Menu
Private  TextBox   txtSearch;         // Text Box that will use Menu.

// Following is in Constructor
   ContoxtMonu1 = now ContoxtMenu();
// Add Mono items and Event handler using Add method
   Context Menu1 . MenuItems. Add ("Azuro background",
        new System. Event Handler(this .MonuItem_Click));
   ContextMonu1 . MonuItems . Add ("White Bauground"
        new System . EventHandler (this. MonuItem _click));
   Context Menu1 . MenuItems. Add ("Beige  Bauground",
        new System . EventHandler ("this. MonuItem_Click));
```

The Completed Menu is attached to a control by Setting the Control's ContextMenu property to the ContextMenu:

```
// Associate text box with a Context Menu
this.txtSearch.ContextMenu = this.ContextMenu1;
```

A right click on txtSearch causes the Menu to pop-up. Click One of the Menu items and this event handling routine is called.

```
Private Void MenuItem_Click(object Sender, System EventArgs e)
{
  // Sender identifies Menu item Selected.
   MenuItem conMi = (MenuItem) Sender;
    String txt = conMi.Text;
// Source control is control associated with this Event
   if (txt == "Azure Backgroung")
      this.ContextMenu1.SourceControl.BackColor = Color.Azure;
   if (txt == " White Background")
      this.ContextMenu1.SourceControl.BackColor = Color.White;
   if (txt == "Beige Background")
      this.ContextMenu1.SourceControl.BackColor = Color.Beige;

}
```

The two most important things to note in this example are that the argument Sender identifies the Selected Menu item and that the ContextMenu property SourceControl identifies the Control associated with the Event.

Forms Inheritance :-

Just as a class inhorits from a Class.

a GUI form - which is also a class - Can inherit the settings, Properties and Control layout of a preexisting form. This Means that you Can Create a form, Beforo looking at the details of inhoritance.

Building and Using a Form's library

Each form Consists of a physical .cs file.

A library of Multiple forms is Created By Compiling each .cs filo into a Common .dll file. After this is done.

Tho forms Can be accessed by any Compliant Language

eg Let's Use the Compilor from the Command line to Compile two forms into a Single .dll file

```
CSC /t : Library product.cs Customor.cs /out. ADCFormLib.dll
```

A base Form must provide a namespace for the derived form to reforonce it. The following code defines is a product a namespace for our example.

```
namespace products
{
public class productForm : System.Windows.Forms.Form
{
```

To inherit this form, a class uses the Standard inheritance Syntax & designates the base Class by it's namespace and Class name

// User Form derived from base Class Form

class NewProductForm : products.productsForm

{

As a final Step, the Compiler must be given a reference to the external assembly ADCFormlib so that the base class can be located. If using VS.NET you use the Project . AddReference Menu Option to Specify the assembly from the Command line, the reference flag is used.

csc /t:winexe / r:ADCFormlib.dll MyApp.cs

Using the Inherited Form:-

If the derived form provides no additional code, it generates a form identical to it's base form when executed. Of course, the derived form is free to add Controls and Supporting Code.

The only restriction is that menu items cannot be added to an existing menu, however, an entire menu can be added to the form and even replace an existing One on the base form.

The properties of inherited Controls Can be Changed. but their default access Modifier of Private must first be Changed to protected.

The base Form then recompiled. The derived form is then free to make modifications. It May reposition the Control or even Set its Visible Property to false to keep it form being displayed.

Overriding Events:-

Suppose the base form Contain a button that responds to a click by Calling event handler Code to Close the Form.

However, in your derived form, you want to add Some data verification Checks before the form Closes.

One's Instinct to add a delegates and Event Handler Method to respond to the button click event in the derived form.

However, this does not override the Original EventHandler in the base form. and both event handling routines get Called. This solutions is to restructure the Original eventhandler to call a Virtual Method that can be Overridden in the derived Form.

eg:- Sample Code for the base Form

```
Privato void btn2_Clickod (objout Sender, System.EventArgs e)
{
ButtonClicks ();     // Have Virtual Method do actual Work.
}
Protected Virtual Void ButtonClicks ()
 {
  this.close();
 }
```

This derived form simply Overrides the Virtual Method and includes it's Own Code to handle the event:

```
 Protected Override Void ButtonClicks()
  {
   // Code to Perform any data Validation
   this.close();
  }
```

Button classes, Group Box, Panel, and Label

The Button Class:-

A button is The most popular Way to enable a User to initiate Some Program action

Typically, The button responds to a Mouse click or Keystroke by firing a Click event that is handled by an Event Handler.

Constructor : public Button()

The Constructor creates a button instance with no Label.

The button's Text property Set's its Caption and can be used to define an access key

Button's Appearance:-

Button's Styles in .NET are limited to placing text and an Image on a button. The following Properties are used to define the apperance of buttons, Checkboxes & radic buttons.

Flat Style

Foor Values

-> FlatStyle.Flat

-> Flat Style. Popup

-> Flat Style. Standard

-> FlatStyle. System. Standard

Flat Creates a Flat button, Popup Creates a Flat button that become a three-dimensional on a mouse Over. System

Image

↳ Specifies the image to be Placed on the button.

```
button1.Image = Image.FromFile("C:\\book.gif");
```

Image Align

↳ Specifies the Position of the Image on the buttoo

ContentAlignment enum;

```
button1.ImageAlign = ContentAlignment.MiddleRight;
```

TextAlign

↳ Specifies the position of text on the Image using the Content Alignment Value.

Handling Button Events:-

A button's click event can be triggered in several ways.

Mouse click of the button, by pressing the Enter key (or) Space bar or by pressing the Alt key in combination with an access ~~b.~~ key.

An access key is created by placing an & in front of one of the characters in the control's text Property Value

The following code segment declares a button set's it's access key to C. and register an Event handler to be called when the Click event is triggered.

```
Button btnclose = new Button();
 btnclose.Text = "& close";
 btnclose.click += new EventHandler (btnclose_Clicked);
// Handle Mouse Click, ENTER key or Space bar
 Private Void btnclose_Clicked (object sender,
                     System.EventArgs e)
 {
     this.close();
 }
```

Tho CheckBox Classs

The checkBox Control allows a User to Select a Combination of Options on a Form - in Contrast to the Radiobutton. Which allows Only One Selection from group.

Constructor : public CheckBox()

The Constructor Creates an Unchecked check box With no label. The Text and Image properties allow the Placement of an Optional text description Or Image beside the box.

Checkbox Apperance:

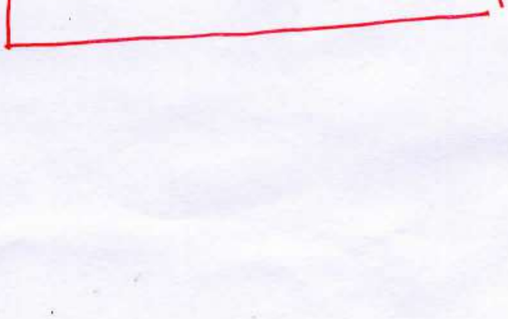
checkbox can be displayed in two Styles. as a traditional checkbox followed by text /or/ as a toggle button that is raised When Unchecked and Flat When Checked.

Appearance.Normal (or) Apperance.Button

The following Code Creates the two Check boxes

Eg

```
// Create traditional Checkbox
   this. CheckBox1 = new Checkbox();
   This. CheckBox1. Location = new System. Drawing. Point
                                              (10, 120);
   this. CheckBox1. Text = " "Selva ";
   this. checkBox1. Checked = true;
// Create Button Style checkBox();
   this. checkBox2 = new CheckBox();
   this. checkBox2. location
                = new System. Drawing. Point (10, 150)
   this. CheckBox2. Text = "Click"
   this. CheckBox2. Apperance = Apporance. Button;
   this. CheckBox2. checked = true;
   This. CheckBox2. TextAlign = Content AlignMent. MiddleCenter;
```



The RadioButton Class

The Radiobutton is a Selection Control the function the Same as a checkbox Except that Only One radiobutton within a group. Can be Selected

A group Consists of Multiple Controls located Within the Same immediate Container.

Constructor : public RadioButton()

The Constructor Creates an Unchecked Radiobutton With no associated text. The Text and Image properties allow the Placement of an Optional text description or image beside the box.

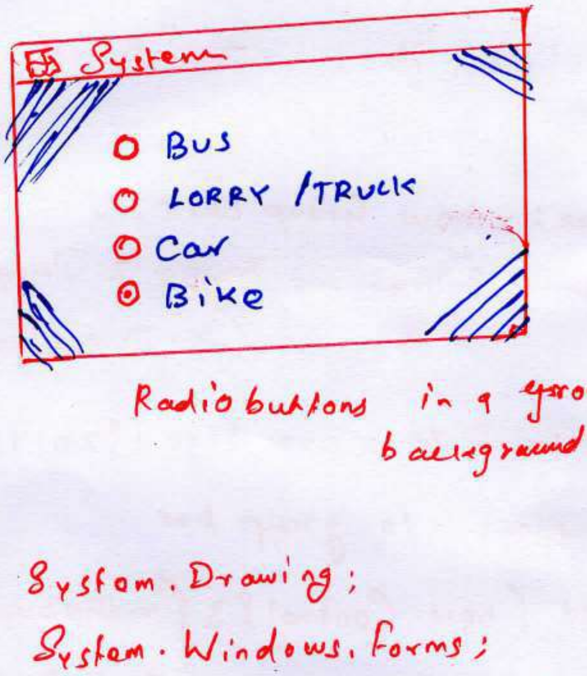
A radiobutton's apperance is defined by the Same properties used with the checkbox and button.

Apporance and Flat Style.

Placing Radio Buttons in a group

Radio buttons are placed in groups that allow Only One item in the group to be Selected.

The frequently Used GroupBox and Panel Container Controls Support background images and Styles that can enhance form's apperance.



Radio buttons in a groupbox that has a background Image.

eg:-

```
Using System.Drawing;
Using System.Windows.forms;
public class SystemForm : Form
{
 private RadioButton radiobutton1
   ,,         ,,          ,,    2
   ,,         ,,          ,,    3
   ,,         ,,          ,,    4
 Private  GroupBox  groupBox1;
public SystemForm()
{ this.groupBox1 = new GroupBox();
  this.radiobutton4 = new RadioButton();
  this.radiobutton3 = new RadioButton();
  this.radiobutton2 = new RadioButton();
    ,,        ,,  1  =  ,,      ,, ,, ;

  this radioButton4.BackColor = color.Transparent;
  this radioButton4.Font = new Font("Microsoft Sans Serif",
                                   8.25F, FontStyle.Bold);

  this.radioButton4.Forecolor =
                   SystemColors.ActiveCaptionText;

  this.radiobutton4.location = new Point(16,80);
   ,,     ,,    4.Name = "Radiobutton4";
  this   .,.    4.Text = "Bike

//Group Box
  this.groupBox1 = new Group Box();
  this.groupBox1.Background Image = Image.FromFile
                        ("C:\\Sys.jpg");

  this groupbox1.Size = new Size(120,112);

  //Add radiobuttons to group box

  groupBox1.Add(new Control[]{radiobutton1, radiobutton2
                          radiobutton3, radiobutton4})

  }
}
```

The Group Box Class

A Group Box is a Containes Control that Places of a border. around it's Collection of Controls.

Constructor : public GroupBox()

The Constructor Creates an untitled Group Box having a default width of 200 Pixels and a default height of 100 pixels.

Panel Class :-

The Panel Control is a Container Used to group of Collection of Controls. It's closely related to the GroupBox. Control. but as a descendent of the Scrollable Control class. It adds a Scrolling Capability.

Constructor: public panel()

It's Single Constructor Creates a borderless Container area that has Scrolling disabled. By default, a Panel takes the background color of it's Container, which makes it invisible On a form.

→ A GroupBox may have a Visible Caption, Wheras Panel does not

→ A GroupBox always displays a border ; a Panel's border determined.

Border Style Property

↳ BorderStyle . None

↳ BorderStyle . Single

↳ BorderStyle . Fixed3D

→ A GroupBox does not Support Scrolling, a Panel enables automatic Scrolling When it's AutoScroll property is Set to true.

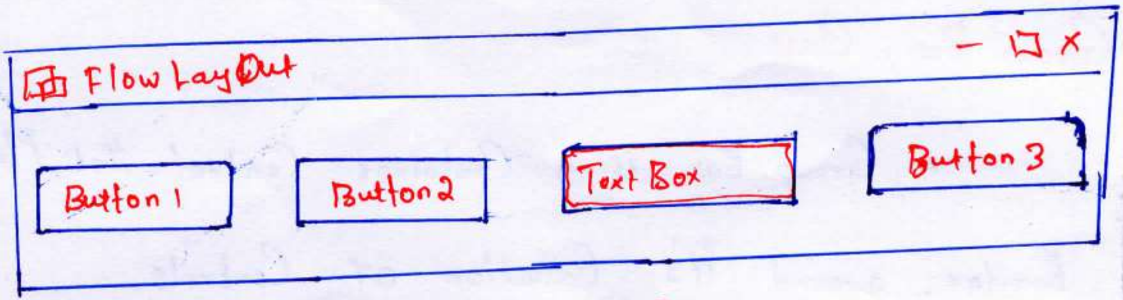


Fig: Flow layout Panel

Eg

```
Flow layout Panel flp = new FlowlayoutPanel();
flp. FlowDirection = FlowDirection. Left to Right ;
// Ctrls are automatically positioned left to right
flp. Controls. Add (Button 1);
  ,,  ,,  ,, (Button2);
  ,,  ,,  ,, (Textbox):
  ,,  ,,  ,, (Button3);
```

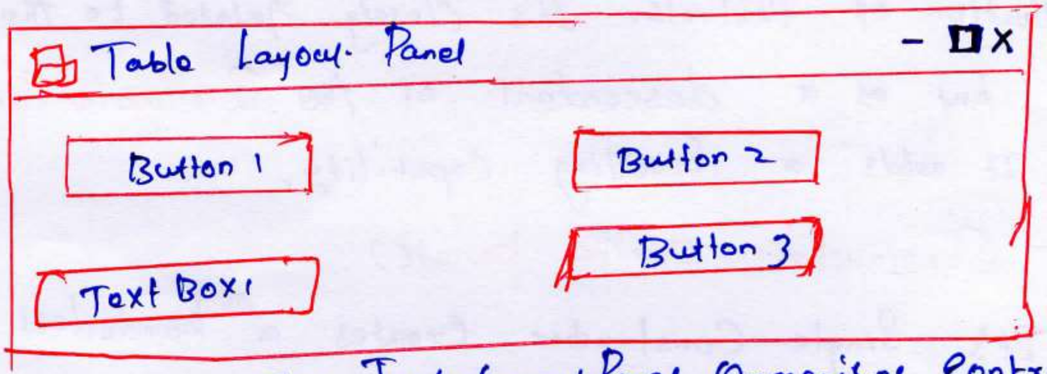


Fig: TableLayoutPanel Organises Controls in a Grid

eg

```
Table layoutPanel tlp = new Table Layout Panel ();
// Causes the Insert around Each cell:
tlp. CellBorder Style = TablelayoutPanel Cell BorderStyle . Inset;
  tlp. ColumnCount = 2
  tlp. Row Count = 2
// if Grid is full add extra Cells by adding colum
  tlp. Growstyle = Table layout Grow Style . Add Column.
  tlp. padding = new Padding (1, 1. 4, 5)    → row & column are filled.
  tlp. Controls. Add (Button 1);
  tlp. Controls. Add (Button 2);
```

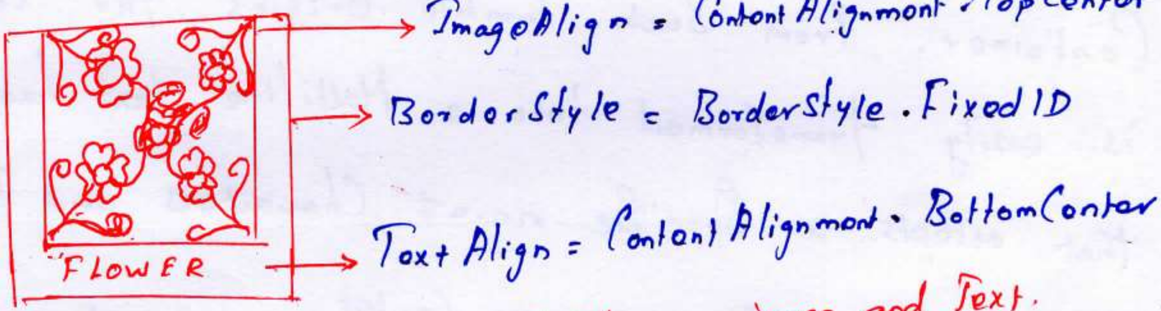
The Label Class

Tho label class is usod to add descriptive information to a form.

Constructor : public label()

This Constructor Croates an instanco of a label having no caption. Uso tho Toxt Proporty to assign a Value to the label.

Tho image, ~~and~~ TextAlign and Bordor Style proportios can be Used to dofine and ostablish the label's appoaranco.



ImageAlign = Contont Alignmont .Top contor
BorderStyle = BorderStyle . Fixed 1D
Toxt Align = Contont Alignmont . BottomContor
FLOWER

Fig :- Label Containing an image and Text.

eg :-

```
label imglabel = new label();
imglabel . Backcolor = color. white ;
image img = Image . FromFile ( "c:\\ flower.jpg" );
imglabel . Image = img
imglabel . ImageAlign = Content Alignmert. BottomContor ;
~~imglabel . TextAlign = Content Alig~~
imglabel . Borderstyle = BorderStyle . Fixed3D
imglabel . ImgeAlign = Content Aligrment . Top center ;
imglabel . Text = " FlowER " ;
imglabel . Size = new Size (img. Width + 10, img. height + 25);
```

Text Box Class:-

The familiar TextBox is an easy-to-use Control that has several properties that affect apperance but few that Control its Content.

Constructor: public TextBox()

The Constructor creates a TextBox that accepts one line of text and uses the color and font assigned to the Container. From such humble Origins, the Controls is easily transformed into a Multiline text handling box that accepts a Specific no. of Characters and formats them to the Left, right, or Center.

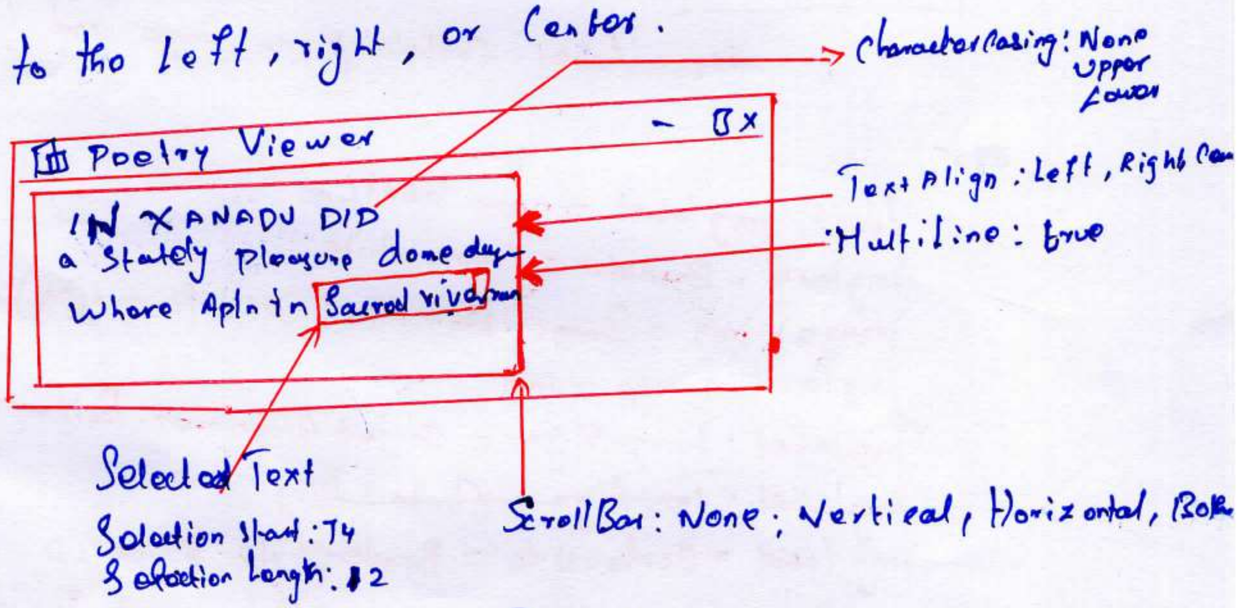

CharacterCasing: None
Upper
Lower
Poetry Viewer
IN XANADU DID
a Stately pleasure dome decree
Where Alph the Sacred river ran
TextAlign: Left, Right, Center
Multiline: true
Selected Text
Selection Start: 74
Selection Length: 12
ScrollBar: None, Vertical, Horizontal, Both

Fig:- TextBox Properties.

```
txt poetry.Text = "IN XANADU DID a Stately pleasure dome decree,";
txt poetry.AppendText("\r\nwhere Alph the Sacred river ran");
```

The listBox class

A listBox Controls is used to provide a list of items from which the User may Select One or more items. The list is typically, text but can also include Images and Objects.

Constructor : public listBox ()

The Constructor creates an empty listBox. The code to populate a listBox is typically placed in the Containing form's Constructor or Form.load eventhandler

If the listBox.Sorted . property is set to true. listBox items are sorted alphabetically in ascending Order.

Adding items to a listBox

A listBox has an items Collection that Contains all elements

Elements Can be added by binding the listBox to a data Source or Manually by using the Add Method.

```
lstArtists . Items . Add ("Monet");
lstArtists . Items . Add ("Rembrandt");
    "         "     "  ("Monet
    "         "   . Insert (0, "Botticelli;"); //Place at top
```

List Boxes may also contain Objects, Because an object may have many members raises the question of what is displayed in the TextBox. Bcz by default a listBox display the Results of an item's ToString method. It is necessary to Override the System.Object method to return the String you want displayed.

```
//Instance of this class will be placed in a listBox
public class Artists
{
 public String BDate, DDate, Country;
 Private String firstname;
 Private String lastname;
 public Artists (String birth, String death, String fname, String lname,
                                                String city);
 {
  BDate : birth;
  DDate : Death;
  Country : city
  firstname = fname;
  Lastname : lname;
 }
public Override String ToString()
 {
  return (lastname + ", " + Firstname);
 }
Public String GetName
 {
  get {return lastname;}
 }
public String GetName
 { get {return firstname; }
}}
```

To string has been Overridden to return the artists last and first name, which are displayed in the ListBox.

```
lst Artists .Itoms. Add
    ( now Artists ("1832","1883", "Edouard", "Manet", "Fr"));
lst Artists . Itoms. Add
    ( now Artists("1840","1926", "Claude", "Monet", "Fr"));
lst Artists . Itoms. Add
    (now Artists ("1606", "1669", "Von Rijn", "Rombrandt", "Ne"));
Lst Artists . Itoms. Add
   (now Artists ("1445","1510", "Sandre", "Botticelli", "It"));
```

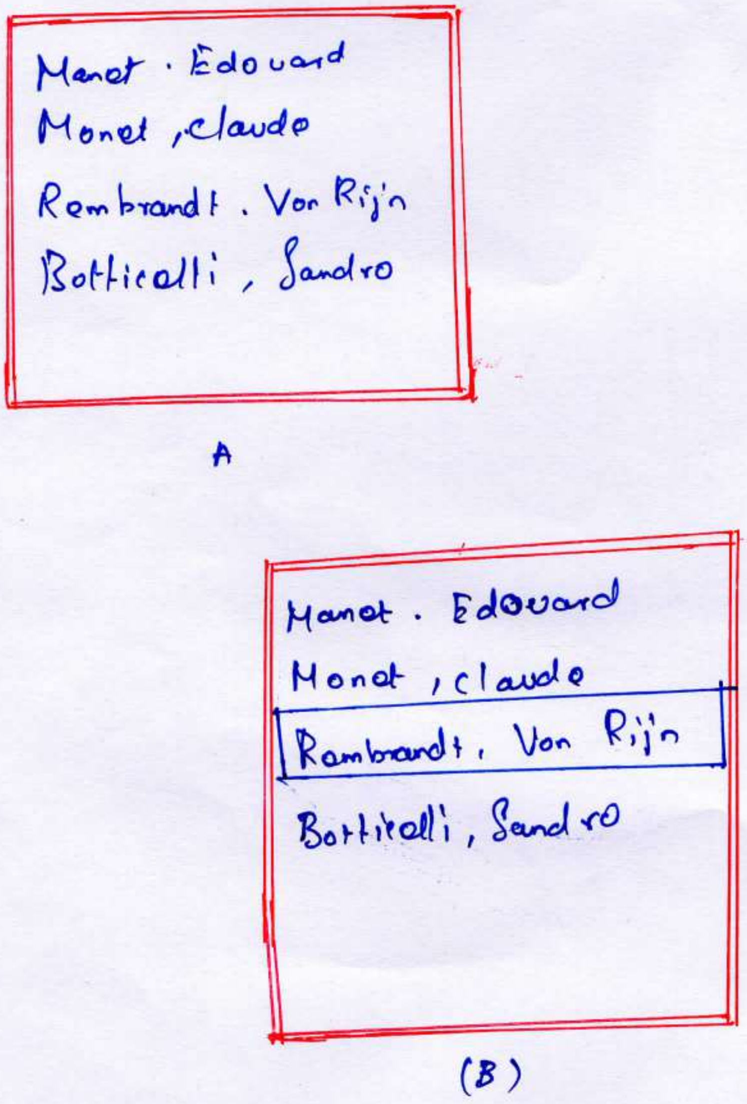


Fig:- (A) Default (B) Custom drawn.

UNIT - IV

XML DATA AND CONTROLS:-

XML:-

XML extends for Extensible Markup Language.

Objects' are stored (or) Streamed across the Internet by Serializing them into an XML

Web Services interCommunication is based on XML.

Working With XML:-

XML is defined as having the basic ability to read, and Write that language.

In XML, functional literacy embraces More than Reading and Writing XML data.

XML schema document (.xsd) that is used Validate the Content

One or more XML Style sheet (.XSL) it is used to transformation.

XML literacy as the ability to do five things:-

⇒ Create an XML file

⇒ Read and Query an XML File

⇒ Create an XML Schema document

⇒ Use an XML Schema document to Validate XML data

⇒ Create and Use an XML Style Sheet to transform XML data.

Using XML Serialization to Create XML Data.

Serialization is a Convenient Way to Store objects So they Can later be deserialized into the Original objects.

XML Serialization often offers a good Choice for Converting it into an XML format.

However, there are Some restrictions to keep in mind when applying XML Serialization to a class:

→ The class must Contain a public default (Parameterless) Constructor

→ Only a public property or Field Can be Serialized.

→ A read-Only property Cannot be Serialized.

→ To Serialize the objects in a Custom Collection Class

The class must derive from the System System.Collections.CollectionBase class and include an indexer.

The Easiest Way to Serialize Multiple Objects is usually to place them in Strongly typed array.

Example: Xml Serializer Class

```
<?xml Version="1.0" Standalone="yes"?>
<Films>
  <Movies>
      <Movie_ID> 5</Movie_ID>
      <Movie_title> Citizen</Movie_title>
  </Movies>
  <Movies>
      <Movie-ID> 6</Movie_ID>
      <Movie-title> K.G.F </Movie_title>
  </Movies>
</Films>
```

Serialization Attributes:-

By default, the elements Created from a class take the name of the Property they represent

ex The Movie_title Property is Serialized as a <Movie_title> element.

However, there is a Set of Serialization attributes that can be Used to Override the default Serialization results.

There are More than a dozen Serialization attributes.

Here are Some other Commonly used Ones.

XMlAttribute : Is Attached to a property (or) Field and Causes it to be rendered as an attribute with in element.

Example : XML Attribute { "Movie ID"

Result <Movies Movie.ID = "5">

XMl Ignore : Causes the Field or property to be executed from the XML.

XML Text : Causes the Value of the field or property to be rendered as text.

No elements are Created for the Member Name

Example : { [XML Text] public String Movie_title {

Result : <Movies MovieID = "5"> Citizen

XML Schema Definition: (XSD)

The XML Schema Definition document is an XML File. that is used to Validate the Contents of another XML document.

The Schema is essentially a template that defines in detail what is permitted in an associated XML document.

.NET Provides Several Ways to Create a Schema from an XML data document. One of the easiest Ways is used the XML Schema definition tool (xsd.exe), Simply, run it from a Command line.

```
C:/ xsd.exe    oscar winners .XML
```

Using an XML Style Sheet :-

A Style sheet is a document that describes how to transform raw XML data into a o/p format.

The Mechanism that Performs that transformation is referred to as an XSLT (extensible Style lang Transformation) Processor.

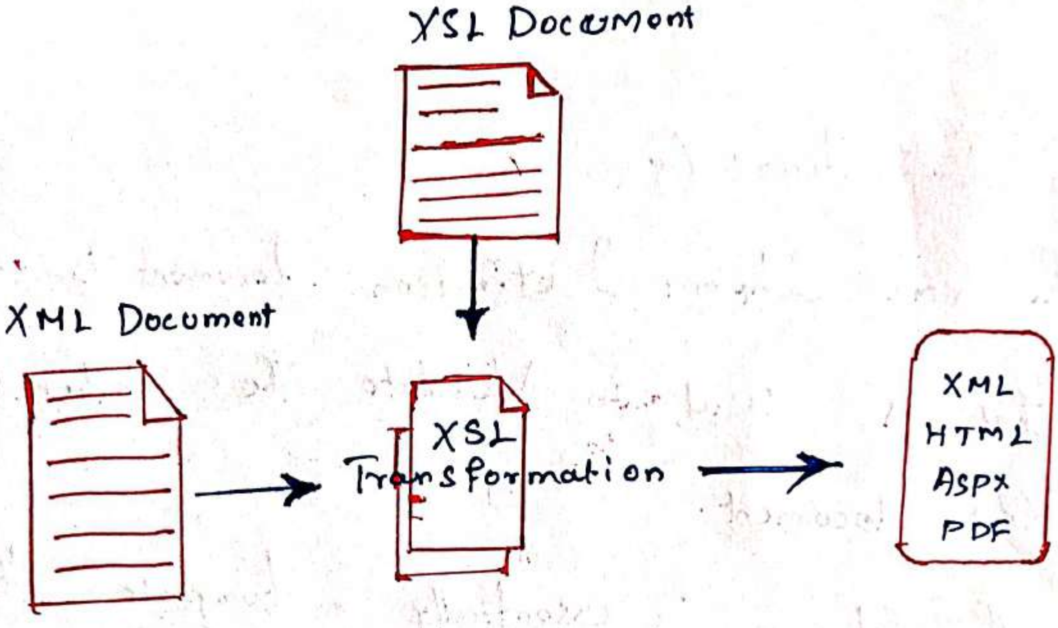


Fig:- publishing documents with XSLT

The XSLTransform Class:-

The .NET Version of the XSLT Processor is the XSLTransform class found in the System.xml.XSL namespace.

Movie title	Movie Year	AFI Rank	Director
Citizon	1996	2	EdwortL.
K.G.F	2019	1	prashanth Neel.

2: Techniques for Reading XML Data :-

XML Can be Represented in two basic ways:- external document Containing embedded data (or) as an in-Memory tree Structure know as a Document Object Model (DOM).

XML Can be read in forward-Only Manner as a Stream of token's representing the Files Content. The XML Reader and XMLText Reader Operate in this Manner.

More Options are available for processing the DOM bcz. it is Stored In Memory Can be traversed randomly.

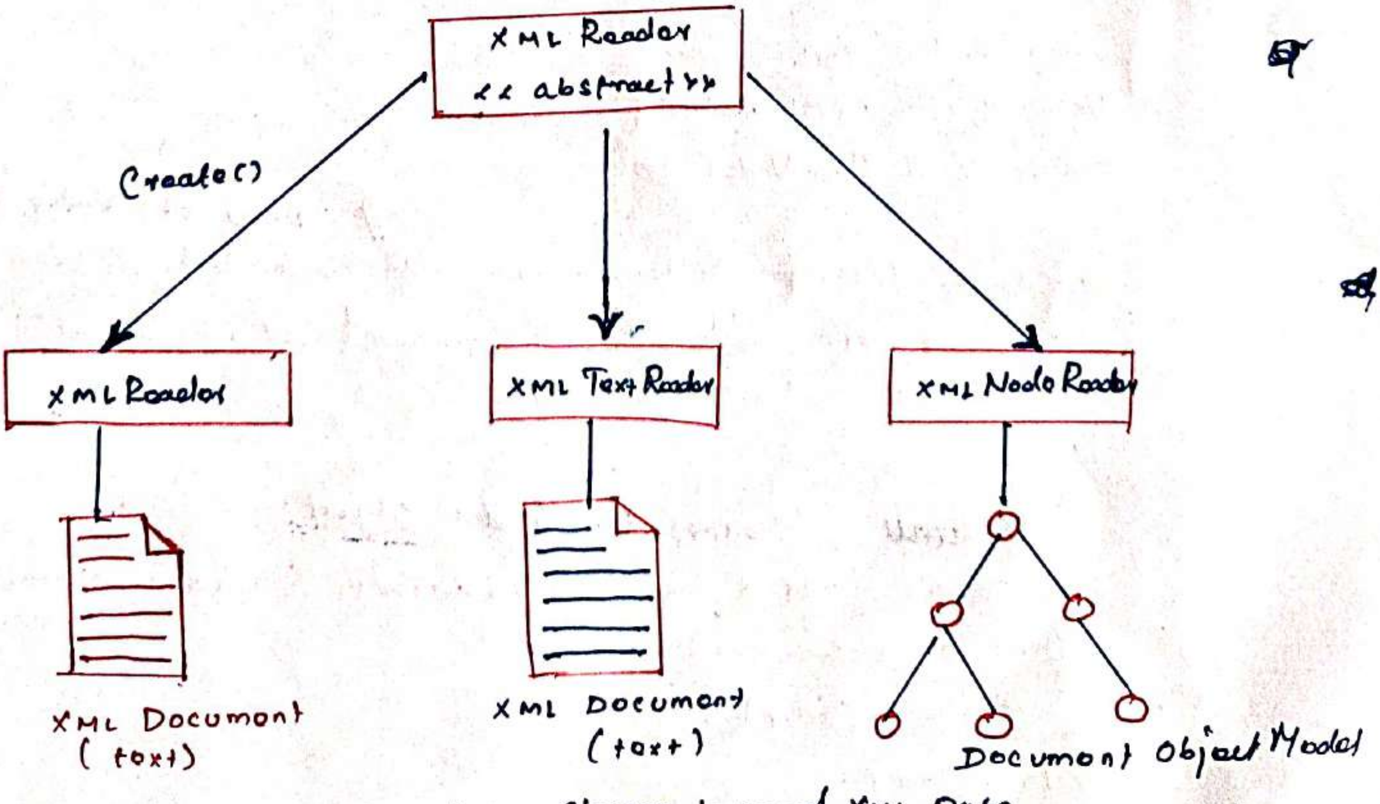


Fig:- Classes to read XML Data.

XML Reader Class:-

XML Reader is an abstract class processing Methods and properties that enable an application to pull data from an XML file One node at a time in a forward-Only, read-Only Manner.

Nodes are inspected Using the Name, NodeType and Value properties.

XML Reader Serves as a base Class for the Concrete classes XML Text Reader and XML Node Reader.

XMLReader Cannot be directly instantiated however, it has a Static Create Method that Can return an instance of the XMLReader Class.

Eg:- Using XMLReader to Read an XML Document:-

```
Using System.XML
Using System.XML.XPath;
public Void ShowNodes()
{                                        // settings obj enable/disables on XML Rea-
                                                                                -der
XML Reader Settings Settings = new XML Reader Settings()
Settings.Conformance level = Conformance level.Fragment';
Settings.Ignore WhiteSpace = true;
try
 {
   //Create XML Reader Object.
XML Reader  rdr = XML Reader.Create ("C:\oscar Winners.XML".
                                            , settings);
while (rdr.Read())
 { format (rdr);
 }
rdr.Close();
}
catch (Exception  e)
 {
   Console.Writeline ("Exception : {0}", e.ToString());
 }
}
private Static void Format (XMLTextReader  reader)
 {
   Console.Write (reader.NodeType + "<" + reader.Name + ">" +
                          reader.Value);
   Console.Writeline();
}
```

The code first creates an XML ReaderSettings object. This object sets features that define how the XMLReader object processes the input stream.

Conformance level property specifies how the input is checked. The stmnt is

Settings.ConformanceLevel = ConformanceLevel.Fragment;

Specifies that the input must conform to the , standards that define an XML 1.0 document fragment - an XML document that does not necessarily have a root node.

XML document file are then passed to the Create method that returns an XMLReader instance:

```
XmlReader rdr = XmlReader.Create ("c:\\oscar winner.xml,
                                   settings);
```

The files content is read in a node at a time by the XMLReader.Read Method.

XML Node Reader class:-

The xmLNode Reader is another forward-only reader that processes XML as a Stream of nodes.

It differs from the XML Reader Class in two Significant Ways:-

→ It processes nodes from an in-memory DOM tree Structure rather than a text file.

→ It can begin reading at any Subtree node in the Structure - not just at the root node (beginning of the document)

3: Techniques For Writing XML Data:-

The Easiest way to present data in an XML format is to use. If the data is in a Collection class.

it Can be Serialized Using XMl Serializer class.

if it is in a DataSet, The Data Set. Write XML method Can be applied.

Writing XML with the XML Writer Class:-

The XMl Writer class offers precise Control Over Each Character written to an XML Stream or file.

→ XML Writer Settings . Check Characters property Configures the XML Writer to check for illegal Characters in text nodes and XML names.

As well as check the Validity of XML names.

An Exception is thrown if an inValid Character is detected.

→ XML Writer Settings . Conformance Level property Configures The XML Writer to gurantee That the Stream complies with the Conformance level that is Specified.

→ XML Writer . Write Value metho is used to Write data to the XML Stream as a CLR Type (int, double)

The XML Writer class Not Surprisingly there are a lot of Similarities to the closely Related to XML Reader class. Both use the Create method to Create an Object instance and both have Constructor Overloads that accepts a Setting Object - XML Writer Settings.

Eg:- Write XML Using Xml Writer Class:

```
Private void WriteMovie()
{
String [,] Movielist = { { "Annie Hall", "Woody Allen" },
                         { "David", "Lawrence" } };

// Define Settings to govern Writer actions.
  xml Writer Settings Settings = new xml Writer Settings();
  Settings.Indent = true;
  Settings.Indent Chars = ("  ");
  Settings.Conformance level = Conformance level.Document;
  Settings.CloseOutput = false;
  Settings.OmitXml Declaration = falce;
// Create xml Writer Object
  xml Writer Writer = xml Writer.create("c:\\mymovies.xml, settings")
  writer.WriteStart Document();
  Writer.Write Element Comment ("Output from Xml Writer Class");
  Writer.WriteStart Element ("Films");
  for(int i=0; i<=Movielist.GetUpperBound(0); i++)
  {
    try
    {
      writer.WriteStart Element ("Movie");
      writer.Write Element String ("Title", Movielist[i,0]);
      writer.Write Element String ("Director", Movielist[i,1]);
        " "       "    Start Element ("Movie ID");
      writer.writeValue(i); //No need to convert to String
      writer.write End Element();
      Writer.Write End Element();
    }
    catch (Exception ex)
    {
      MessageBox.Show (ex.message);
    }
  }
Writer . WriteEndElement ();
Writer . Flush ();
Writer . Close ();

/*  O/P
   <?xml Version ="1.0" Encodings = "utf-8"?>
   <!-- O/P from XmlWriter class -->
   <films>
     <movie>
       <Title> Annie Hall </title>
       <Director> Woody Allen </Director>
       <Movie ID> 0 </Movie_ID>
  </Movie>
     <Movie>
         <Title> David </title>
         <Director> Lawrance </Director>
         <Movie_ID> 1 </Movie_ID>
     </Movie>
   </films>
*/
}
```

4: Using XPath to Search XML

Of representing XML in a tree model As opposed to a data Stream - is the Capability to query and locate the tree's Content using XML Path Language (xPath)

This technique is similar to using a SQL Command on relational data.

An xPath expression (query) is created and Passed to an engine that evaluates it.

xPath is a formal query Language defined by the XML Path language 2.0 Specification (www.w3.org/TR/xpath).

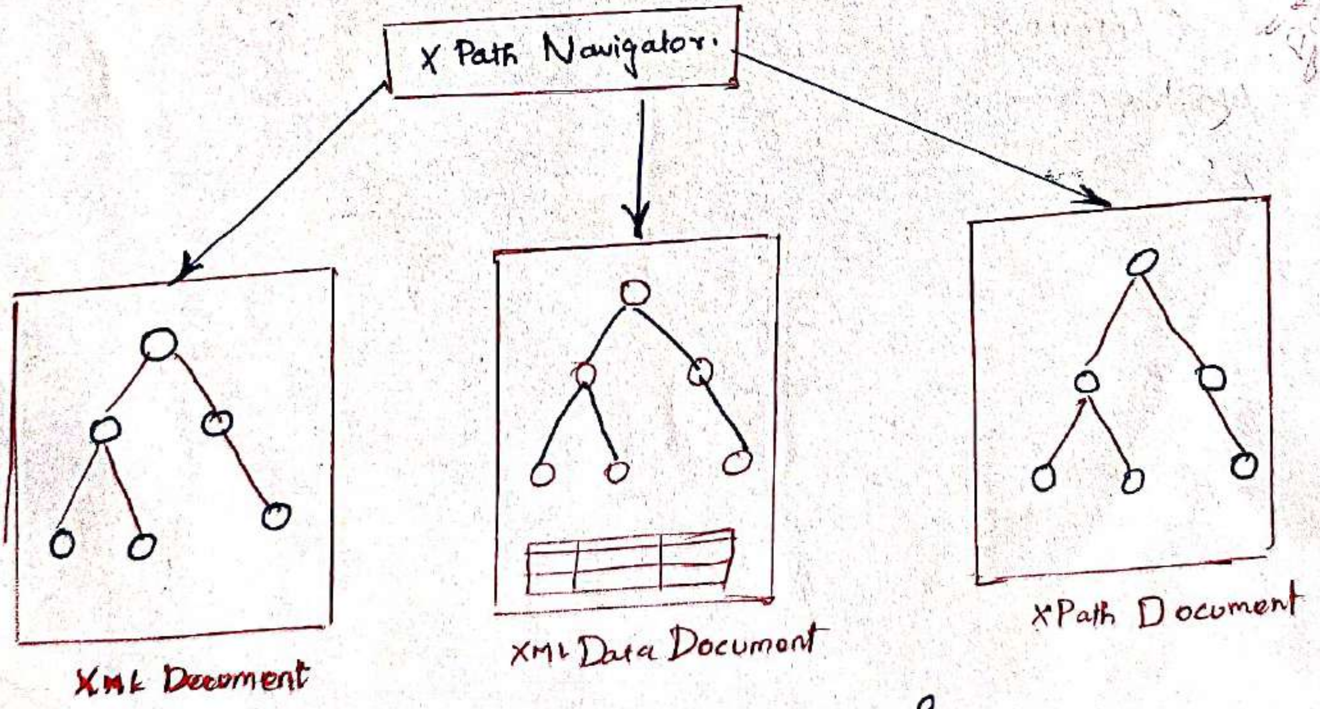


Fig: XML Classes that Support XPath Navigation.

XPath evaluation is exposed through the XPathNavigator abstract class.

The navigator is an XPath Processor that works on top of any XML data Source that exposes the IXPathNavigable Interface.

The most important member of this interface is the CreateNavigator Method, Which returns an XPathNavigator Object.

Three classes that implement this interface.

- → XML Document (implement the W3C DOM & Support XPath Queries, navigation & editing
- → XML Data Document (member of this ~~is~~ System.XML namespace;
- → XPath Document (as well as the XML Navigator Class.

Inherits from XML Document. It provides the capability to map XML tree and Vice Versa.

This class is Optimized to perform XPath queries and represents XML in a tree of read-Only nodes that is more Stroun of.

Constructing XPath Queries:

Queries can be executed against each of those Classes Using either an XPathNavigator Object or the SelectNodes method implemented by Each class.

```
// XPATH EXPRESSION is the xpath query applied to the data
(1) Return a list of nodes
    XmlDocument doc = new XmlDocument();
    doc.Load ("Movies.xml");
    XmlNodeList Selection = doc.SelectNodes (XPATHEXPRESSION);
```

// (2) Create a navigator and. Execute the query

```
XPath Navigator nav = doc.CreateNavigator();
XPath Node Iterator iterator = nav.Select (XPATHEXPRESSION);
```

The XPathNodeIterator class encapsulates a list of nodes and provides a way to iterate Over the list.

Fig:- Screenshot

XML Document and X Path

The expression in this example extracts the Set of last_name nodes. It then prints the associated text. SelectNodes uses a navigator to evaluate the expression

```
String exp = "//film/directors//last_name";
XmlDocument doc = new XmlDocument();  // Build DOM tree
doc.load ("directormovies.xml");
XmlNodelist directors = doc.SelectNodes (exp);
foreach (XmlNode n in directors)
    Console.Writeline(n.InnerText); // last name or director.
```

The XmlNode.InnerText Property Concatenates the Values of Child nodes and displays them as a text String.

This is a Convenient way to display tree Contents during application testing.

```
</films>
```

Table 10-3 summarizes commonly used XPath operators and provides an example of using each.

Table 10-3 XPath Operators

Operator	Description
Child operator (`/`)	References the root of the XML document, where the expression begins searching. The following expression returns the `last_name` node for each director in the table: `/films/directors/last_name`

Recursive descendant operator (`//`)	This operator indicates that the search should include descendants along the specified path. The following all return the same set of `last_name` nodes. The difference is that the first begins searching at the root, and second at each `directors` node:
	`//last_name`
	`//directors//last_name`

Wildcard operator (`*`)	Returns all nodes below the specified path location. The following returns all nodes that are descendants of the `movies` node: `//movies/*`
Current operator (`.`)	Refers to the currently selected node in the tree, when navigating through a tree node-by-node. It effectively becomes the root node when the operator is applied. In this example, if the current node is a `directors` node, this will find any `last_name` child nodes: `.//last_name`

Table 10-3 XPath Operators *(continued)*

Operator	Description
Parent operator (`..`)	Used to represent the node that is the parent of the current node. If the current node were a `movies` node, this would use the `directors` node as the start of the path: `../last_name`
Attribute operator (`@`)	Returns any attributes specified. The following example would return the movie's runtime assuming there were attributes such as `<movie_ID time="98">` included in the XML. `//movies//@time`

Filter operator (`[ ]`)	Allows nodes to be filtered based on a matching criteria. The following example is used to retrieve all movie titles directed by Martin Scorsese:
	`//directors[last_name='Scorsese']`
	`    /movies/movie_Title`

Collection operator (`[ ]`)	Uses brackets just as the filter, but specifies a node based on ar ordinal value. Is used to distinguish among nodes with the sam name. This example returns the node for the second movie, *Raging Bull*: `//movies[2]` (Index is not 0 based.)
Union operator (`|`)	Returns the union of nodes found on specified paths. This exan ple returns the first and last name of each director: `//last_name | //first_name`

Note that the filter operator permits nodes to be selected by their content. The are a number of functions and operators that can be used to specify the matching c teria. Table 10-4 lists some of these.

Table 10-4 Functions and Operators used to Create an XPath Filter

XPath Documont and XPath

For applications that Only need to query an XML document. The XPath Documont is the recommoded Class If required for updating a tree and runs 20 to 30 porcent fastor than XML Document.

XML Reador to load all or part of a document Into it. This is done by Creating the reader. Positioning it to a desired Sub tree ; and then passing it to the XPathDocument Constructor.

XML Data Document and XPath:-

XML Data Document class allows you to take a Dataset (an object containing rows of data). and replica of It's a tree structure.

The tree not Only represents the DataSet, but It is Synchronized with it. This Means that Changes made to the DOM (or) DataSet are automatically reflected in the Order,

Because the XML Data Document is derived from XML Document. it Supports the basic Methods and Properties used to Manipulate XML data.

The most interesting of those is the GetRowFromElement Method that takes an XmlElement and Converts it to a Corresponding DataRow.

```
myRow = XmlDoc.GetRowFromElement((XmlElement)myNode);
```

Adding and Removing Nodes on a Tree:-

Besides a locating and reading data, Many application need to add, edit and delete information is an XML document tree. This is done Using ~~the~~ Methods that edit the Content of a node and add or delete nodes. After the Changes have been made to the tree, The Updated DOM is Saved to a file.

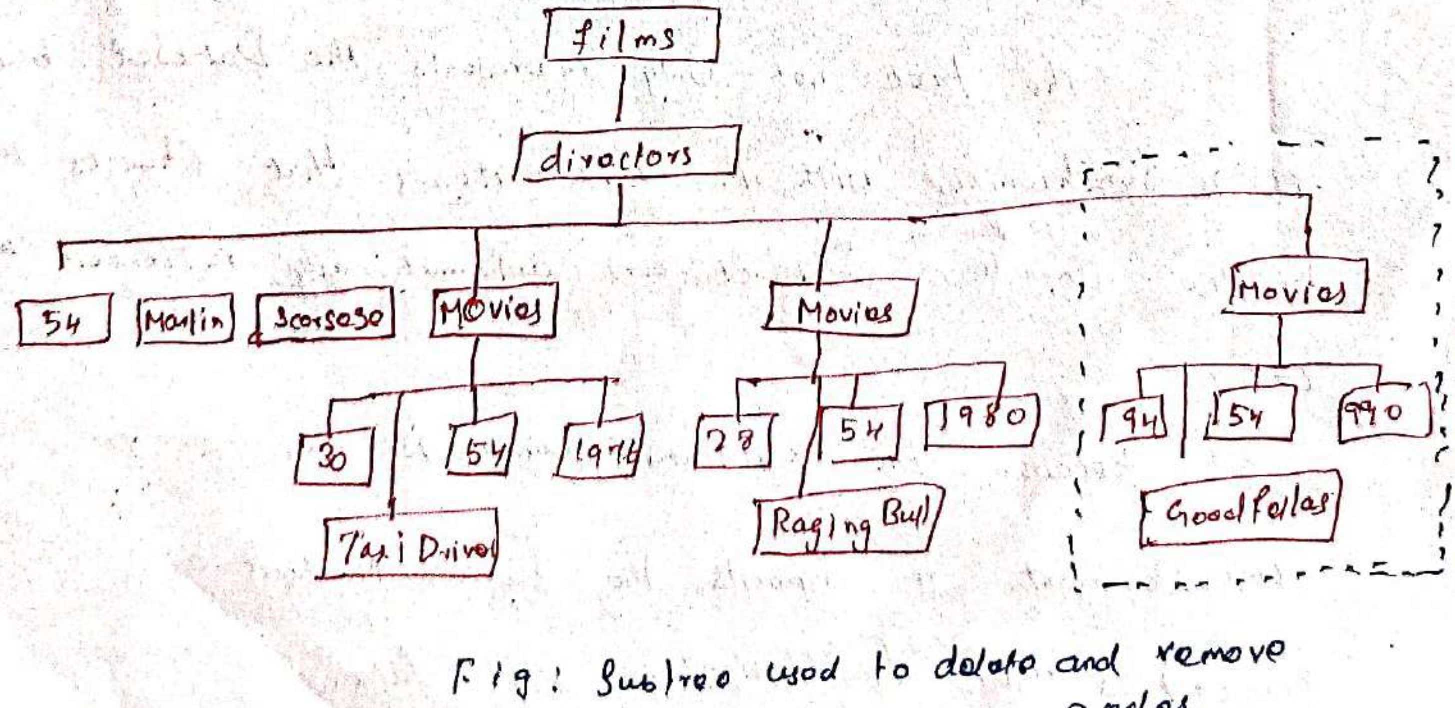


Fig: Subtree used to delete and remove nodes.

1. ADO.NET

ADO.NET is basod on a flexible Set of classes that allow data to be accessad from within the Managod Environment of .NET

These Classes are usod to access a Variaty of data Sources including relational databases. XML, files, Spread Sheets, and text Files

OVERVIEW OF THE ADO.NET ARCHITECTURE:-

The ADO.NET architecture is designed to make life easier for both the application developer and the Database Provider.

To the developer it presents a Set of abstract Classes that define a Common Set of Methods and Properties that can be usod to access any data Source. The data Source is treatod as an abstract entity

For Database Providers, ADO.NET Serves as a blue Print that describe the base API Classes and interface Specification providers must Supply with their product.

Many database products, Such as MySQL, and Oracle, have Custom .NET data provider implomontation available.

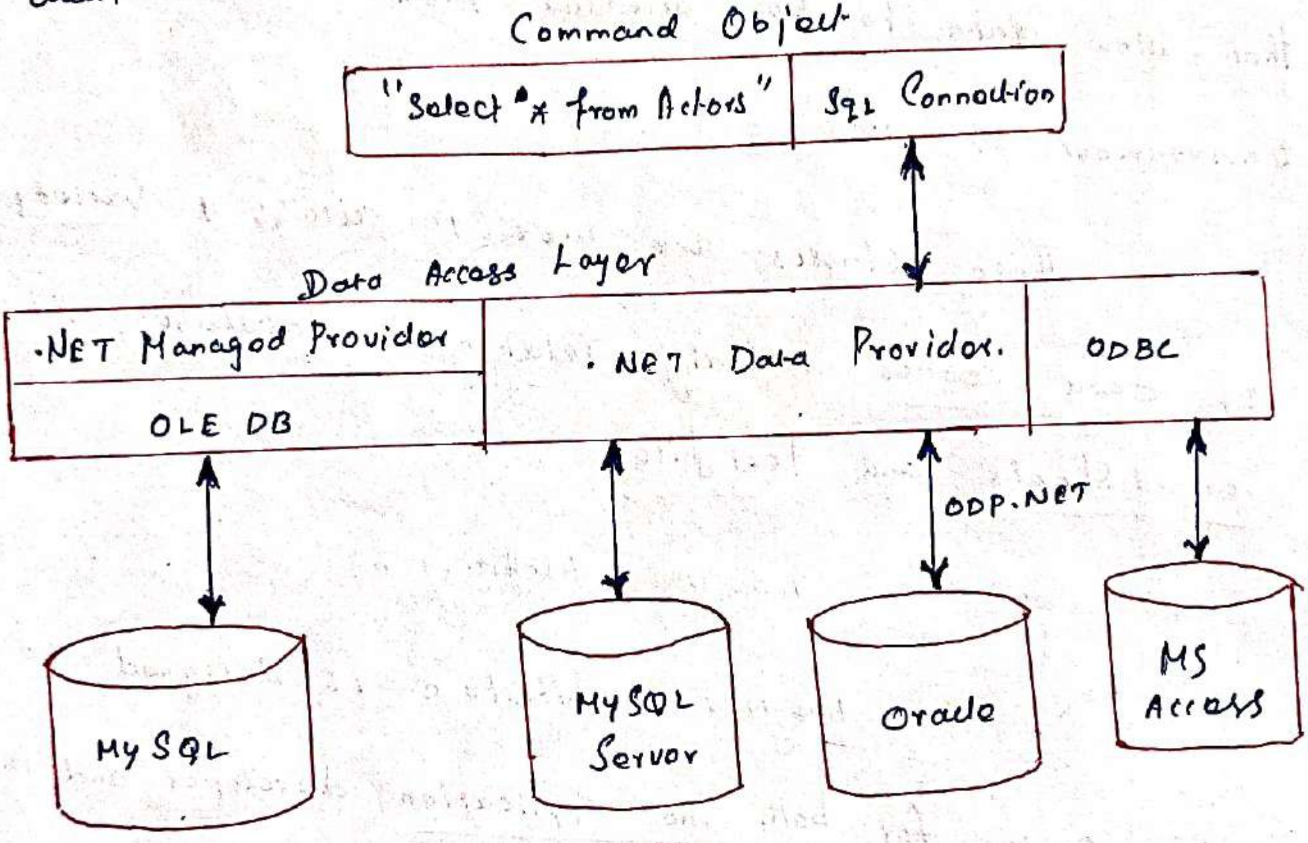


Fig: ADO.NET data access options.

OLE DB Data Provider in .NET

An OLE DB provider is the Code that Sits between the data Consumer and the native API of a data Source.

It is a COM based Solution in Which the data Consumer and provider are COM objects that Communicate through COM Interface.

.NET includes an OLEDB data provider that functions as a thin Wrapper to route calls into the native OLE DB

A Writing Code to use OLE DB is essentially the Same as Working with .NET data provider. In fact, now .NET classes provide a "factory" that can dynamically produce code for a Selected provider.

.NET Data Provider:-

.NET data provider provides the Same basic Service to the Client as the OLE DB provider. Exposing a data Source's API to a client, it's advantage is that it can directly access the native API of the data Source, rather than relying on an intermediate data access bridge.

Data provider Objects for Accessing Data

A Managed data provider exposes four Classes that enable a data Consumer to access the provider's data Source.

-> DB Connection - Establishes a Connection to the data Source

-> DB Command - Used to query or Send Command to the data Source.

-> DB Data Reader - Provides read-only and fwd-only access to the data Source.

-> DB Data Adapter

↳ Serves a channel through which DataSets connects to a provider.

Data Access Models: Connected and DisConnected

An Overview of Using ADO.NET to access data stored in relational tables. Through Simple Examples it presents the Classes and Concepts that distinguish the Connected and disconnected access models.

In this Section - as well as entire Chapter - Use data from the Films database defined.

It Consists of a Movies table Containing the top 100 Movie as Selected by the AFI (American Films Institute) in 1996. The data is downloadable as a Microsoft Access (.mdb) file and an XML text (.xml) file

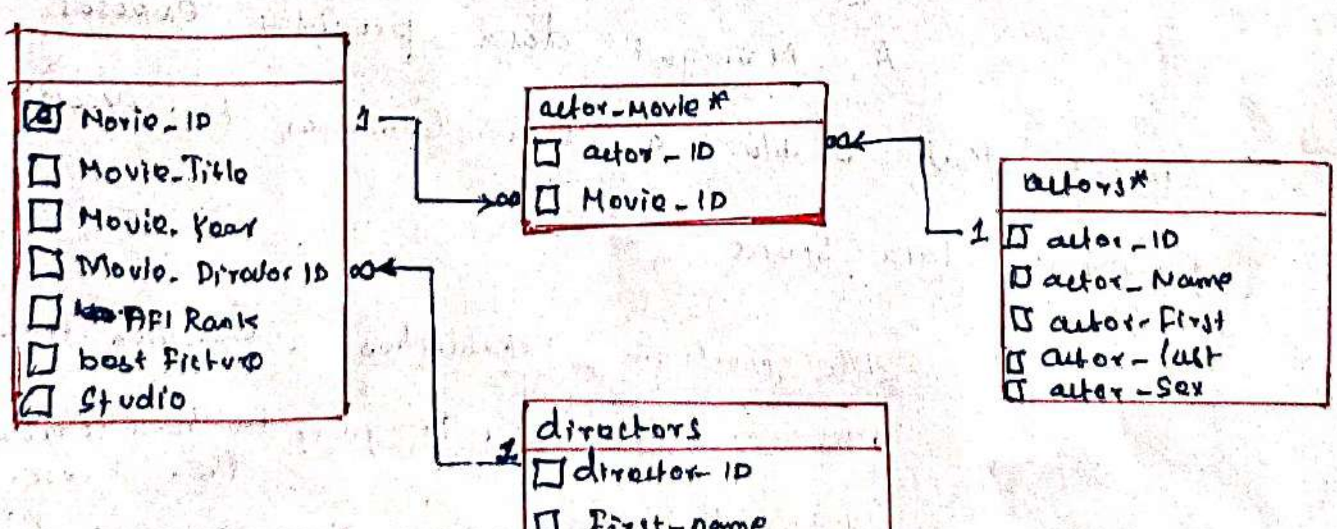


Fig:- Films database tables

Connected Model:-

In the ADO.NET Connected mode, an active Connections is maintained between an application DataReader Object and a data Source.

A row of data is returned from the data Source each time the object's Read Method is executed.

The most Important Charaboristics of the Connected model is that it reads data from a resultset (records returned by a SQL Command) One record at a time in a forward-only, read-only manner. It provides no direct Way to Update or add data.

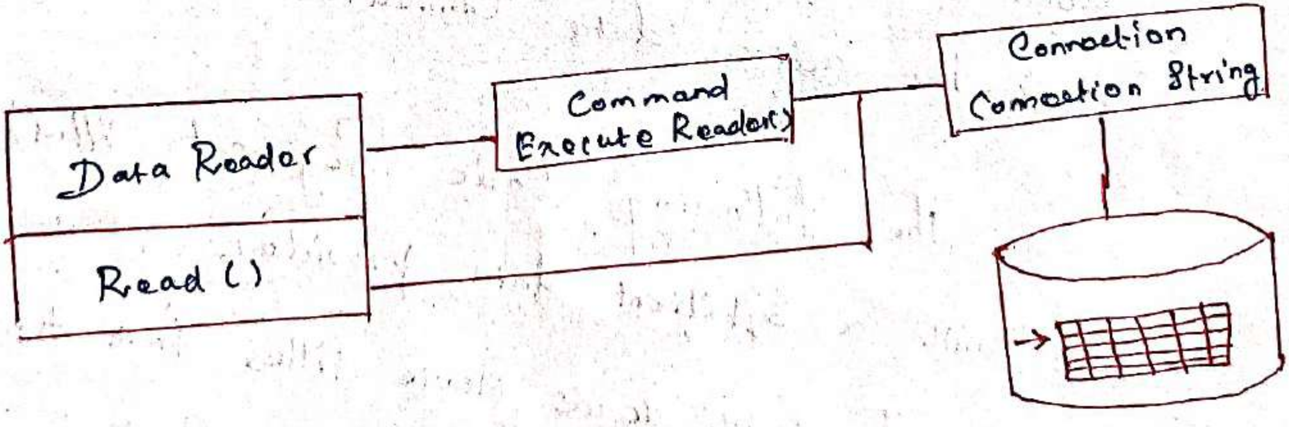


Fig:- Data Reader is used in ADO.NET Connected Mode.

Working with tho Data Roador in Four Steps:

→ Tho Comoction objact is Croated by passing a Connaction String to its Constructor.

→ A String Variable is assigned tho SQL Command that Specifies tho data to fetch.

→ A Command object is Createod. It's Overloads accept a Connoction Objact, a query String and a transaction objact.

→ The Data Roador Objact is Created by executing tho Command. ExecuteReador() method. The objact is then used to read the query results One line at a time over the active data Connection.

The following Code Segment illustrates thes Steps with a SqlClient data provider. The code reads Movie titles from the database and displays them in a listBox Control

Eg:-

```
//System.Data.sqlClient namespace is required
// (1) Create Connection
  Sql Connection  Conn = new SqlConnection (connStr);
  Conn.Open();
// (2) Query String
  String  sql = "SELECT Movie_title FROM Movies Order By
  Movie_Year";
// (3) Create Command Object
  SqlCommand  cmd = new SqlCommand(sql, conn);
  DbDataReader rdr;
// (4) ~~B~~ Create DataReader
  rdr = cmd.ExecuteReader(command Behaviour.closeConnection);
  While(rdr_Read())
  {
    listBox1.Items.Add(rdr("Movie-Title")); //Fill listBox
  }
  rdr.Close();        //Always close data reader
```

The Parameter to ExecuteReader Specifies that the Connection is closed when the data reader object is closed.

UNIT-V
.NET APPLICATION DOMAINS

.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.

Application Domains

.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).

Running an Application with a Specific Evidence and Zone

```
String myApplication = @"C:\MyApp.exe"; String[] argsToApp = null;
String myURL = @"http://www.readorrefer.in"; SecurityZone myZone =
SecurityZone.Internet; Evidence myEvidence = new Evidence();
myEvidence.AddHost(new Zone(myZone)); myEvidence.AddHost(new Url(myURL));
AppDomain app = AppDomain.CreateDomain(myApplication, myEvidence);
app.ExecuteAssembly(myApplication, myEvidence, argsToApp);
```

.NET REMOTING

The .NET Remoting provides an inter-process communication between Application Domains by using Remoting Framework.

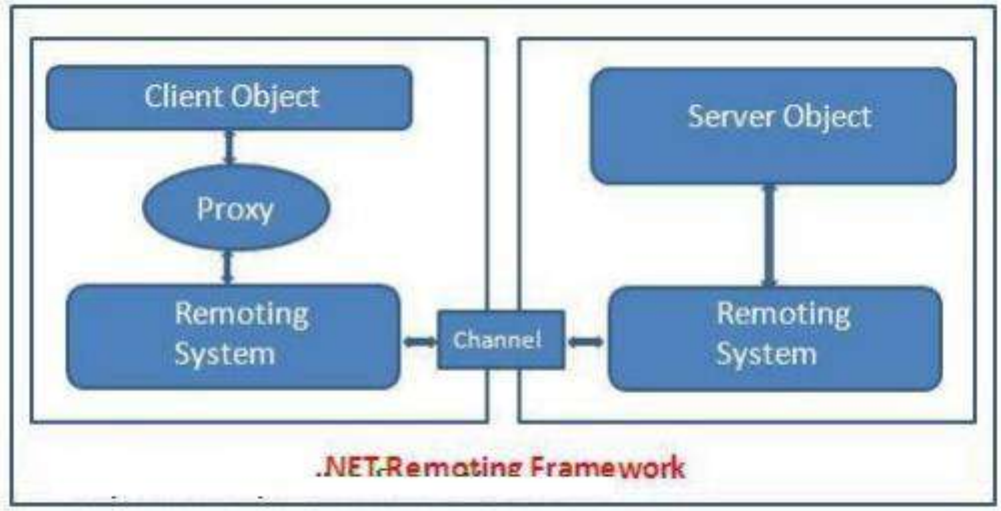
Remoting

The **.NET Remoting** provides an inter-process communication between Application Domains by using Remoting Framework. The applications can be located on the same computer , different computers on the same network, or on computers across separate networks. The .NET Remoting supports distributed object communications over the **TCP** and **HTTP** channels by using Binary or **SOAP** formatters of the data stream.

The main three components of a Remoting Framework are :

1. C# Remotable Object
2. C# Remote Listener Application - (listening requests for Remote Object)
3. C# Remote Client Application - (makes requests for Remote Object)

The Remote Object is implemented in a class that derives from ***System.MarshalByRefObject*** .



You can see the basic workflow of **.Net Remoting** from the above figure. When a client calls the Remote method, actually the client does not call the methods directly . It receives a proxy to the remote object and is used to invoke the method on the **Remote Object** . Once the proxy receives the method call from the Client , it encodes the message using appropriate formatter (**Binary Formatter** or **SOAP Formatter**) according to the Configuration file. After that it sends the call to the Server by using selected Channel (**TcpChannel** or **HttpChannel**). The Server side channel receives the request from the proxy and forwards it to the Server on Remoting system, which locates and invokes the methods on the Remote Object. When the execution of remote method is complete, any results from the call are returned back to the client in the same way.

Before an object instance of a Remotable type can be accessed, it must be created and initialized by a process known as Activation. **Activation** is categorized in two models , they are Client-activated Objects and Server-activated Objects.

C# Remote Activation

The real difference between client-activated and server-activated objects is that a server-activated object is not really created when a client instantiates it. Instead, it is created as needed. By default the .NET Framework ships with two formatters(Binary Formatter or SOAP Formatter) and two channels(TcpChannel ,HttpChannel).

C# Remote Channels
C# Remote Formatters
Formatters and Channel are configured by using Configuration files. It can be easily Configured by using XML-based files.
C# Remote Configuration

.NET LEASING AND SPONSORSHIP

.NET manages the lifecycle of objects using garbage collection. .NET keeps track of memory allocation and objects accessed by all the clients in the app domain.

Leasing and Sponsorship

.NET manages the lifecycle of objects using garbage collection. .NET keeps track of memory allocation and objects accessed by all the clients in the app domain. When an object becomes unreachable by its clients, the garbage collector eventually collects it. If the objects are in the same app domain as the clients, garbage collection functions fine. In fact, even in the case of a client in one app domain accessing an object in a different app domain in the same process, garbage collection still works, because all app domains in the same process share the same managed heap. In the case of remote objects accessed across processes and machines, however, the strategy breaks down because the object may not have any local clients. In this case, if garbage collection were to take place, the garbage collector would not find any references to the object and would deem it garbage, even though there are remote clients (on other machines, or even in separate processes on the same machine) who wish to use the object. The rest of this section addresses this challenge.

In the following discussion, a "remote object" is an object in a different process. The core piece of the .NET remoting architecture designed to address this problem is called leasing and sponsorship. The idea behind leasing is simple: each server object accessed by remote clients is associated with a lease object. The lease object literally gives the server object a lease on life. When a client creates a remote server object (that is, actually creates it, rather than connects to an existing instance), .NET creates a lease object and associates it with the server object. A special entity in .NET remoting called the lease manager keeps track of the server objects and their lease objects. Each lease object has an initial lease time. The clock starts ticking as soon as the first reference to the server object is marshaled across the app domain boundary, and the lease time is decremented as time goes by. As long as the lease time doesn't expire, .NET considers the server object as being used by its clients. The lease manager keeps a reference to the server object, which prevents the server object from being collected in case garbage collection is triggered. When the lease expires, .NET assumes that the server object has no remaining remote clients. .NET then disconnects the server object from the remoting infrastructure. The server object becomes a candidate for garbage collection and is eventually destroyed. After the object is disconnected, any client attempt to access it results in an exception of type RemotingException, letting the client know the object has been disconnected. This may appear strange at first, because the object may very well still be alive. .NET behaves this way because otherwise, the client's interaction with the remote object will be nondeterministic. If .NET allowed remote clients to access objects past their lease time, it would work some of the time but would fail in those cases in which garbage collection had already taken place.

.NET CODING DESIGN GUIDELINES

.NET Coding Design Guidelines: Â· Naming Guidelines Â· Class Member Usage Guidelines Â· Guidelines for Exposing Functionality to COM

NET Coding Design Guidelines

- Naming Guidelines
- Class Member Usage Guidelines
- Guidelines for Exposing Functionality to COM
- Error Raising & Handling Guidelines
- Array Usage Guidelines
- Operator Overloading Usage Guidelines
- Guidelines for Casting Types
- Common Design Patterns
- Callback Function Usage
- Time-Out Usage
- Security in Class Libraries
- Threading Design Guidelines
- Formatting Standards
- Commenting Code

- Code Reviews
- Additional Notes for VB .NET Developers

.NET ASSEMBLY

There are two kind of assemblies in .NET; Â· private Â· shared

Assemblies

The .NET assembly is the standard for components developed with the Microsoft.NET. Dot NET assemblies may or may not be executable, i.e., they might exist as the executable (.exe) file or dynamic link library (DLL) file. All the .NET assemblies contain the definition of types, versioning information for the type, meta-data, and manifest. The designers of .NET have worked a lot on the component (assembly) resolution.

There are two kind of assemblies in .NET;

- private
- shared

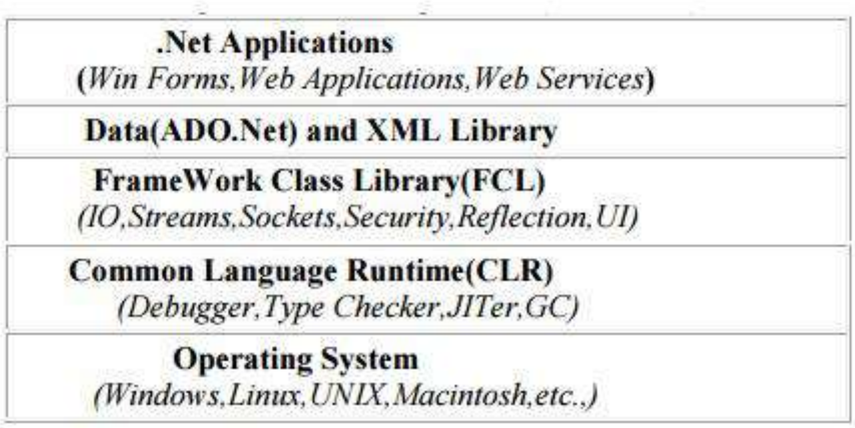
Private assemblies are simple and copied with each calling assemblies in the calling assemblies folder.

Shared assemblies (also called strong named assemblies) are copied to a single location (usually the Global assembly cache). For all calling assemblies within the same application, the same copy of the shared assembly is used from its original location. Hence, shared assemblies are not copied in the private folders of each calling assembly. Each shared assembly has a four part name including its face name, version, public key token and culture information. The public key token and version information makes it almost impossible for two different assemblies with the same name or for two similar assemblies with different version to mix with each other.

An assembly can be a single file or it may consist of the multiple files. In case of multi-file, there is one master module containing the manifest while other assemblies exist as non-manifest modules. A module in .NET is a sub part of a multi-file .NET assembly. Assembly is one of the most interesting and extremely useful areas of .NET architecture along with reflections and attributes, but unfortunately very few people take interest in learning such theoretical looking topics.

What are the basic components of .NET platform?

The basic components of .NET platform (framework) are:



.Net Applications
(Win Forms,Web Applications,Web Services)

Data(ADO.Net) and XML Library

FrameWork Class Library(FCL)
(IO,Streams,Sockets,Security,Reflection,UI)

Common Language Runtime(CLR)
(Debugger,Type Checker,JITer,GC)

Operating System
(Windows,Linux,UNIX,Macintosh,etc.,)

XML Web Service Application

On your local computer (localhost), start Visual Studio .NET. On the File menu, click New and then click Project. Under Project types click Visual Basic Projects, then click ASP.NET Web Service under Templates. Name the project TestService.

XML WEB SERVICE APPLICATION

- On your local computer (localhost), start Visual Studio .NET. On the File menu, click New and then click Project. Under Project types click Visual Basic Projects, then click ASP.NET Web Service under Templates. Name the project TestService.
- In Solution Explorer, change the name of Service1.asmx to Services.asmx.
- Open Services.asmx in the visual designer. In the Properties window, change the Name property of the Service1 class to Services.
- Save the project.

Create the XML Web Service Methods

- Open Services.asmx in the code editor.
- Add the following code within the Services class definition to create various Web methods:

```
    <WebMethod()> Public Function GetMessage() As String Return "Today is
the day"
    End Function <WebMethod()> _
.

    Public Function SendMessage(ByVal message As String) As String Return
        "Message received as: " & message
.

        End Function <WebMethod()> _
.
```

Public Function ReverseMessageFunction(ByVal message As String) As String Return StrReverse(message)

End Function

<WebMethod()> Public Sub ReverseMessageSub(ByRef message As String) message = StrReverse(message)

End Sub Save and build the project.

Test the Services with Visual Studio .NET

- In Solution Explorer, right-click Services.asmx and then click View in Browser.
- Follow these steps to use the built-in browser to test each Web method:NOTE: You cannot test the ReverseMessageSub procedure because it expects a ByRef argument.
    - Click the hyperlink for the method that you want to test.
    - Fill in any requested message parameter values.
    - Click Invoke.
    - View the resulting XML and close the results window.

□

- Click the Back button to return to the method list and repeat the steps for the remaining Web methods.

□

- Close the built-in browser.

Create the Test Client Application

- On the File menu, click Add Project, and then click New Project.
- Select Visual Basic Console Application, and then name the project TestHarness.
- On the Project menu, click Add Web Reference.
- In the Address field, type http://localhost/TestService/Services.asmx, and then click Go.

- Click Add Reference to finish creating the Web reference.

.

- In Solution Explorer, right-click localhost in the Web References folder, click Rename, and then change the name to WebService. This becomes the namespace that is used within the test application to refer to the Services class.

Create the Test Code

Open Module1.vb and locate the Sub Main procedure.

Paste the following code in the file to call the appropriate Web methods:

```
Dim strValue As String = "This is my message"
Dim myService As New WebService.Services()
Console.WriteLine(myService.GetMessage)
Console.WriteLine(myService.SendMessage(strValue))
Console.WriteLine(myService.ReverseMessageFunction(strValue))
myService.ReverseMessageSub(strValue) Console.WriteLine(strValue)
```

Test the Client Application

- Create a breakpoint on the following line:
- Console.WriteLine(myService.GetMessage)
- In Solution Explorer, right-click the TestHarness project, and then click Set as StartUp Project.
- On the Debug menu, click Start and wait for the program to enter debug mode.
- On the Debug menu, click Windows, and then click Locals. Use the Locals window to view the value of the strValue variable during the debugging to observe any changes that are made to the variable.
- On the Debug toolbar, use Step Into to step through each line of code from the TestHarness client into the XML Web service.
- Before you end the Main subroutine, confirm that the output in the console window is as expected.
- When the program ends, remove the breakpoint and close Visual Studio .NET.

Web Services Description Language (WSDL)

Web Services Description Language (WSDL) is a format for describing a Web Services interface. It is a way to describe services and how they should be bound to specific network addresses.

WSDL

- Web Services Description Language (WSDL)

➢ Web Services Description Language (WSDL) is a format for describing a Web Services interface. It is a way to describe services and how they should be bound to specific network addresses.

WSDL has three parts:

- ✓ Definitions
- ✓ Operations
- ✓ Service bindings

- ➢ Definitions are generally expressed in XML and include both data type definitions and message definitions that use the data type definitions.
- ➢ These definitions are usually based upon some agreed upon XML vocabulary. This agreement could be within an organization or between organizations.
- ➢ Vocabularies within an organization could be designed specifically for that organization. They may or may not be based on some industry-wide vocabulary.
- ➢ If data type and message definitions need to be used between organizations, then most likely an industry-wide vocabulary will be used.
- ➢ XML, however, is not necessary required for definitions. The OMG Interface Definition Language (IDL), for example, could be used instead of XML.
- ➢ If a different definitional format were used, senders and receivers would need to agree on the format as well as the vocabulary. Nevertheless, over time, XML-based vocabularies and messages are likely to dominate.
- ➢ XML Namespaces are used to ensure uniqueness of the XML element names in the definitions, operations, and service bindings.
- ➢ Operations describe actions for the messages supported by a Web service. There are four types of operations:
  - ✓ **One-way:** Messages sent without a reply required
  - ✓ **Request/response:** The sender sends a message and the received sends a reply.
  - ✓ **Solicit response:** A request for a response. (The specific definition for this action is pending.)
  - ✓ **Notification:** Messages sent to multiple receivers. (The specific definition for this action is pending.)
  - ➢ Operations are grouped into port types. Port types define a set of operations supported by the Web service.
  - ➢ Service bindings connect port types to a port. A port is defined by associating a network address with a port type. A collection of ports defines a service. This binding is commonly created using SOAP, but other forms may be used. These other forms could include

CORBA Internet Inter-ORB Protocol (IIOP), DCOM, .NET, Java Message Service (JMS), or WebSphere MQ to name a few.

SOAP (Simple Object Access Protocol)

SOAP (Simple Object Access Protocol) is a messaging protocol that allows programs that run on disparate operating systems (such as Windows and Linux) to communicate using Hypertext Transfer Protocol (HTTP) and its Extensible Markup Language (XML).

SOAP

- SOAP (Simple Object Access Protocol) is a messaging protocol that allows programs that run on disparate operating systems (such as Windows and Linux) to communicate using Hypertext Transfer Protocol (HTTP) and its Extensible Markup Language (XML).

- Since Web protocols are installed and available for use by all major operating system platforms, HTTP and XML provide an at-hand solution that allows programs running under different operating systems in a network to communicate with each other.
- SOAP specifies exactly how to encode an HTTP header and an XML file so that a program in one computer can call a program in another computer and pass along information.
- SOAP also specifies how the called program can return a response. Despite its frequent pairing with HTTP, SOAP supports other transport protocols as well.
- SOAP defines the XML-based message format that Web service-enabled applications use to communicate and inter-operate with each other over the Web.
- The heterogeneous environment of the Web demands that applications support a common data encoding protocol and message format. SOAP is a standard for encoding messages in XML that invoke functions in other applications.
- SOAP is analogous to Remote Procedure Calls (RPC), used in many technologies such as DCOM and CORBA, but eliminates some of the complexities of using these interfaces. SOAP enables applications to call functions from other applications, running on any hardware platform, regardless of different operating systems or programming languages.
- SOAP calls are much more likely to get through firewall servers, since HTTP is typically Port 80 compliant, where other calls may be blocked for security reasons. Since HTTP requests are usually allowed through firewalls, programs using SOAP to communicate can be sure that the program can communicate with programs anywhere.

- **Some of the advantages of leveraging SOAP include:**
  - ✓ It is platform and language independent.
  - ✓ SOAP provides simplified communications through proxies and firewalls, as mentioned above.
  - ✓ It has the ability to leverage different transport protocols, including HTTP and SMTP, as well as others.
- **Some disadvantages of leveraging SOAP include:**
  - ✓ SOAP is typically much slower than other types of middleware standards, including CORBA. This due to the fact that SOAP uses a verbose XML format. You need to fully understand the performance limitations before building applications around SOAP.
  - ✓ SOAP is typically limited to pooling, and not event notifications, when leveraging HTTP for transport. What's more, only one client can use the services of one server in typical situations.

  - ✓ Again, when leveraging HTTP as the transport protocol, there tends to be firewall latency due to the fact that the firewall is analyzing the HTTP transport. This is due to the fact that HTTP is also leveraged for Web browsing, and many firewalls do not understand the difference between the use of HTTP within a Web browser, and the use of HTTP within SOAP.
  - ✓ SOAP has different levels of support, depending upon the programming language supported. For example, SOAP support within Python and PHP is not as strong as it is within Java and .NET.