

**D.Y.Patil College Of Engineering &
Technology, Kolhapur**

Department of Computer Science & Engineering

Lab Manual

C# PROGRAMMING LAB

T.Y SEM – II, Year 2021-22

Department of Computer Science & Engineering, Kolhapur

Name of the course – C# Programming

Class – TY (CSE)

Practical -2 Hrs/Week

List of Experiment

Sr. No	Document Title	CO Mapped
01	Study of C# Language and .NET framework.	CO1
02	Implementation of Partial, Static & Abstract class in C#.	CO2
03	Develop DLL file and use it in application program.	CO1
04	Implementation of Inheritance, Extension Method & Interface in C#.	CO2
05	Implementation of Multidimensional & Jagged array in C#.	CO2
06	Implementation of Operator overloading in C#.	CO2
07	Implementation of string manipulation application using String & String builder class.	CO2
08	Develop an application using Regex.Matches method and Regular Expression pattern matching.	CO2
09	Implementation of Windows Form based application using different controls.	CO3
10	Implementation of Windows Form based MDI application with different controls.	CO3
11	Implementation of Windows Form based application with Database connectivity with all field validation.	CO3
12	Implement the console-based networking application to obtain network information.	CO4
13	Develop a Windows form application to download the file & process it using stream.	CO3
14	Design Simple ASP.NET web application.	CO5
15	Design simple login and registration page using client-side validation controls in ASP.NET	CO6

Subject Incharge

HOD (CSE Dept.)

Experiment No-01

Title :- Study of C# Language and .NET framework.

Aim : To study the components of .NET architecture and features of C# and .NET framework.

Objective :

- 1) Study the .Net framework Services.
- 2) Study the CLR, CTS, CLS and FCL
- 3) Study the advantages of C# language.

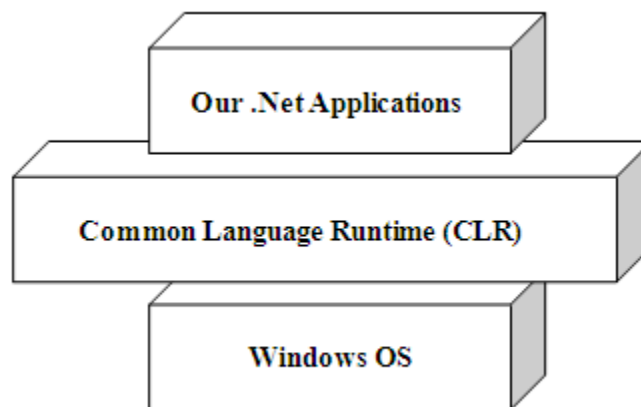
Theory:-

The .Net Architecture and .Net Framework

In the .Net Architecture and the .Net Framework there are different important terms and concepts which we will discuss one by one:-

The Common Language Runtime (CLR)

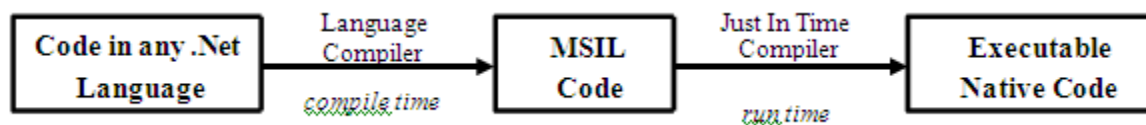
The most important concept of the .Net Framework is the existence and functionality of the .Net Common Language Runtime (CLR), also called .Net Runtime for short. It is a framework layer that resides above the OS and handles the execution of all the .Net applications. Our programs don't directly communicate with the OS but go through the CLR.



MSIL (Microsoft Intermediate Language) Code

When we compile our .Net Program using any .Net compliant language (such as C#, VB.Net or C++.Net) our source code does not get converted into the executable binary code, but to an intermediate code known as MSIL which is interpreted by the Common Language Runtime. MSIL is operating system and hardware independent code.

Upon program execution, this MSIL (intermediate code) is converted to binary executable code (native code). Cross language relationships are possible as the MSIL code is similar for each .Net language.



Just In Time Compilers (JITers)

When our IL compiled code needs to be executed, the CLR invokes the JIT compiler, which compile the IL code to native executable code (.exe or .dll) that is designed for the specific machine and OS. JITers in many ways are different from traditional compilers as they compile the IL to native code only when desired; e.g., when a function is called, the IL of the function's body is converted to native code just in time. So, the part of code that is not used by that particular run is never converted to native code. If some IL code is converted to native code, then the next time it's needed, the CLR reuses the same (already compiled) copy without re-compiling. So, if a program runs for some time (assuming that all or most of the functions get called), then it won't have any just-in-time performance penalty.

The Framework Class Library (FCL)

The .Net Framework provides a huge Framework (or Base) Class Library (FCL) for common, usual tasks. FCL contains thousands of classes to provide access to Windows API and common functions like String Manipulation, Common Data Structures, IO, Streams, Threads, Security, Network Programming, Windows Programming, WebProgramming, Data Access, etc. It is simply the largest standard library ever shipped with any development environment or programming language. The best part of this library is they follow extremely efficient OO design (design patterns) making their access and use very simple and predictable. You can use the classes in FCL in your program just as you would use any other class. You can even apply inheritance and polymorphism to these classes.

The Common Language Specification (CLS)

Earlier, we used the term '.Net Compliant Language' and stated that all the .Net compliant languages can make use of CLR and FCL. But what makes a language a '.Net compliant' language? The answer is the Common Language Specification (CLS). Microsoft has released a small set of specifications that each language should meet to qualify as a .Net Compliant Language. As IL is a very rich language, it is not necessary for a language to implement all the IL functionality; rather, it merely needs to meet a small subset of CLS to qualify as a .Net compliant language. This is the reason why so many languages (procedural and OO) are now running under the Net umbrella.

The Common Type System (CTS)

The .Net also defines a Common Type System (CTS). Like CLS, CTS is also a set of standards. CTS defines the basic data types that IL understands. Each .Net compliant language should map its data types to these standard data types. This makes it possible for the 2 languages to communicate with each other by passing/receiving parameters to and from each other. For example, CTS defines a type, Int32, an integral data type of 32 bits (4 bytes) which is mapped by C# through int and VB.Net through its Integer data type.

Garbage Collection (GC)

CLR also contains the Garbage Collector (GC), which runs in a low-priority thread and checks for un-referenced, dynamically allocated memory space. If it finds some data that is no longer referenced by any variable/reference, it re-claims it and returns it to the OS. The presence of a standard Garbage Collector frees the programmer from keeping track of dangling data.

The .Net Framework

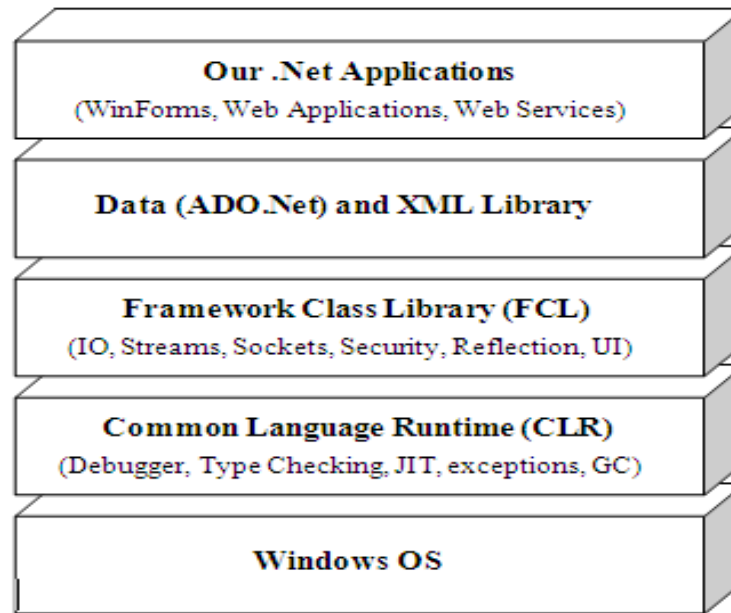


Figure - .Net Framework

The Visual Studio.Net IDE

Microsoft Visual Studio.Net is an Integrated Development Environment (IDE), which is the successor of Visual Studio 6. It eases the development process of the .Net Applications (VC#.Net, VB.Net, VC++.Net, JScript.Net, J#.Net, ASP.Net, and more). The revolutionary approach in this new improved version is that for all the Visual Studio.Net Compliant Languages use the same IDE, debugger, project and solution explorer, class view, properties tab, tool box, standard menu and toolbars.

CONCLUSION: -

Briefly reviewing important aspects of .NET Framework and C# relationship to it, additionally we check the role of CLR, CTS, CLS and MSIL in .NET framework.

Experiment No-02

Title:- Implementation of Partial, Static & Abstract class in C#.

Aim : Implementaion of C# programs to demonstarte the basic concept, object and class conecept in C#

Prerequisite: - Student should be familiar with C & object oriented concepts from Java

Objective :-

1. Study the obiect and its creation.
2. Study the class and its type.
3. Study the methos and its declaration.

Theory:-

Namespace:

Namespaces are C# program elements designed to help you organize your programs. They also provide assistance in avoiding name clashes between two sets of code. Implementing Namespaces in your own code is a good habit because it is likely to save you from problems later when you want to reuse some of your code. For example, if you created a class named Console, you would need to put it in your own namespace to ensure that there wasn't any confusion about when the System. Console class should be used or when your class should be used.

Namespaces don't correspond to file or directory names.

NamespaceCSS.cs

// Namespace Declaration

using System;

// The Namespace

namespace csharp_station

{ // Program start class

class NamespaceCSS

{ // Main begins program execution.

public static void Main()

```
// Write to console
```

```
    Console.WriteLine("This is the new C# Station Namespace.");  
} } }
```

The Main() Method

The programs start execution at a method named Main(). This must be a static method of a class (or struct), and must have a return type of either int or void.

Although it is common to specify the public modifier explicitly, because by definition the method must be called from outside the program, it doesn't actually matter what accessibility level you assign to the entry - point method — it will run even if you mark the method as private. Multiple Main() Methods. When a C# console or Windows application is compiled, by default the compiler looks for exactly one Main() method in any class matching the signature that was just described and makes that class method the entry point for the program. If there is more than one Main () method, the compiler will return an error message.

Console I/O

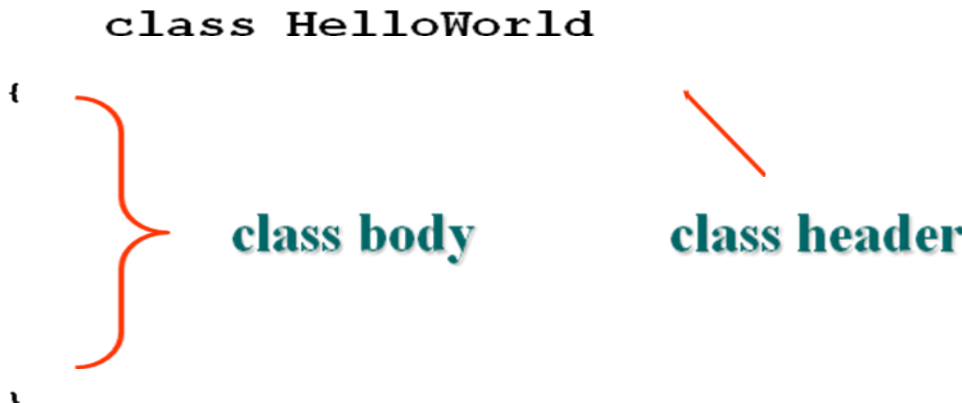
To read a line of text from the console window, you use the Console.ReadLine() method. This will read an input stream from the console window and return the input string. There are also two corresponding methods for writing to the console, which you have already used extensively:

- ❑ Console.Write() — Writes the specified value to the console window.
- ❑ Console.WriteLine() — This does the same, but adds a newline character at the end of the output.

Various forms (overloads) of these methods exist for all of the predefined types, so in most cases you don't have to convert values to strings before you display them.

Classes and Structs:

```
class HelloWorld
```



class body

class header

The class defines what data and functionality each particular object (called an *instance*) of that class can contain. Structs differ from classes in the way that they are stored in memory and accessed (classes are reference types stored in the heap; structs are value types stored on the stack), and in some of their features (for example, structs don't support inheritance). You will tend to use structs for smaller data types for performance reasons. In terms of syntax, however, structs look very similar to classes; the main difference is that you use the keyword struct instead of class to declare them.

Methods

In C# every function must be associated with a class or struct. Note that official C# terminology does in fact make a distinction between functions and methods. In C# terminology, the term “ function member ” includes not only methods, but also other nondata members of a class or struct. This includes indexers, operators, constructors, destructors, and also perhaps somewhat surprisingly properties. These are contrasted with data members: fields, constants, and events.

Declaring Methods

The syntax for defining a method in C# is

```
[modifiers] return_type MethodName([parameters])
{
    // Method body
}
```

Passing Parameters to Methods

In general, parameters can be passed into methods by reference or by value. In C#, all parameters are passed by value unless you specifically say otherwise. However, you need to be careful in understanding the implications of this for reference types. Because reference type variables hold only a reference to an object, it is this reference that will be copied, not the object itself. Hence, changes made to the underlying object will persist. Value type variables, in contrast, hold the actual data, so a copy of the data itself will be passed into the method.

ref Parameters

Passing variables by value is the default, but you can force value parameters to be passed by reference. To do so, use the ref keyword. If a parameter is passed to a method,

and if the input argument for that method is prefixed with the ref keyword, any changes that the method makes to the variable will affect the value of the original object:

out Parameters

C# requires that variables be initialized with a starting value before they are referenced. Although you could initialize your input variables with meaningless values before passing them into a function that will fill them with real, meaningful ones, this practice seems at best needless and at worst confusing.

Partial Classes

The partial keyword allows the class, struct, or interface to span across multiple files. Typically, a class will reside entirely in a single file. However, in situations where multiple developers need access to the same class, or more likely in the situation where a code generator of some type is generating part of a class, then having the class in multiple files can be beneficial. The way that the partial keyword is used is to simply place partial before class, struct, or Interface.

Static Classes

The static constructors and how they allowed the initialization of static member variables. If a class contains nothing but static methods and properties, the class itself can become static. A static class is functionally the same as creating a class with a private static constructor.

An instance of the class can never be created. By using the static keyword, the compiler can help by checking that instance members are never accidentally added to the class. If they are, a compile error happens. This can help guarantee that an instance is never created.

The syntax for a static class looks like this:

```
static class StaticUtilities
{
    public static void HelperMethod()
    {
        //statements
    }
}
```

Conclusion:C# adds some new features that are not present in the OOP model of some other languages and here we understand the use of namespace, object, class and their types in C#

Experiment No-03

Title :- Develop DLL file and use it in application program.

Aim : Create a DLL file and use that file in any application.

Objective : 1) Creation of DLL file.
2) Use of DLL file.

Theory:- A Dynamic Link library (DLL) is a library that contains functions and codes that can be used by more than one program at a time. Once we have created a DLL file, we can use it in many applications. The only thing we need to do is to add the reference/import the DLL File. Both DLL and .exe files are executable program modules but the difference is that we cannot execute DLL files directly.

Creating DLL File:

Step 1 - Open Visual Studio then select "File" -> "New" -> "Project..." then select "Visual C#" -> "Class library".

Step 2 - Change the class name ("class1.cs") to any appropriate name.

Step 3 - In that class, write methods for the demonstration purpose.

Step 4 - Build the solution. If the build is successful then you will see a .dll file in the "bin/debug" directory of your project.

Use of DLL File:

Step 1 - Open Visual Studio then select "File" -> "New" -> "Project..." then select "Visual C#" -> "Windows Forms application" or "Console Based Application".

Step 2 - Design the class.

Step 3 - Add a reference for the dll file, .dll, that we created earlier. Right-click on the project and then click on "Add reference".

Step 4 - Select the DLL file and add it to the project.

Step 5 - Add the namespace of your dll file.

Conclusion: Student successfully created a DLL file and use it.

Experiment No-04

Title:- Implementation of Inheritance , Extension method & Interface in C#.

Aim : Implement the C# program to demonstrate the use of inheritance and interface.

Objective :- 1. Study the inheritance and extension method in C# environment.
2. Study the interface and its declaration.

Theory:-

Inheritance:-

Implementation inheritance means that a type derives from a base type, taking the entire base type's member fields and functions. With implementation inheritance, a derived type adopts the base type's implementation of each function, unless it is indicated in the definition of the derived type that a function implementation is to be overridden.

If you want to declare that a class derives from another class, use the following syntax:

```
class MyDerivedClass : MyBaseClass
{
    // functions and data members here
}
```

If a class (or a struct) also derives from interfaces, the list of base class and interfaces is separated by commas:

```
public class MyDerivedClass : MyBaseClass, IInterface1, IInterface2
{
    // etc.
}
```

For a struct, the syntax is as follows:

```
public struct MyDerivedStruct : IInterface1, IInterface2
{
    // etc.
}
```

If you do not specify a base class in a class definition, the C# compiler will assume that System.Object is the base class. Hence, the following two pieces of code yield the same result:

```
class MyClass : Object // derives from System.Object
```

```

{
// etc.
}
and
class MyClass // derives from System.Object
{
// etc.
}

```

Calling Base Versions of Functions

C# has a special syntax for calling base versions of a method from a derived class:

base. <MethodName> ().

Abstract Classes and Functions

C# allows both classes and functions to be declared as abstract. An abstract class cannot be instantiated, whereas an abstract function does not have an implementation, and must be overridden in any non - abstract derived class. Obviously, an abstract function is automatically virtual. If any class contains any abstract functions, that class is also abstract and must be declared as such:

```

abstract class Building
{
public abstract decimal CalculateHeatingCost(); // abstract method
}

```

Sealed Classes

C# allows classes and methods to be declared as sealed . In the case of a class, this means that you can't inherit from that class. In the case of a method, this means that you can ' t override that method.

```

sealed class FinalClass
{
// etc
}
class DerivedClass : FinalClass // wrong. Will give compilation error
{

```

```
// etc
```

```
}
```

Interfaces

Interface inheritance means that a type inherits only the signatures of the functions and does not inherit any implementations. This type of inheritance is most useful when you want to specify that a type makes certain features available. For example, certain types can indicate that they provide a resource cleanup method called `Dispose()` by deriving from an interface, `System.IDisposable`. Implementation Inheritance which is intended to be implemented by classes to clean up code:

```
public interface IDisposable
{
    void Dispose();
}
```

You can never instantiate an interface; it contains only the signatures of its members. An interface has neither constructors. An interface definition is also not allowed to contain operator overloads, although that's not because there is any problem in principle with declaring them there isn't; it is because interfaces are usually intended to be public contracts, and having operator overloads would cause some incompatibility problems. It is also not permitted to declare modifiers on the members in an interface definition. Interface members are always implicitly public, and cannot be declared as virtual or static.

Extension Method:

Extension methods, as the name suggests, are additional methods. Extension methods allow you to inject additional methods without modifying, deriving or recompiling the original class, struct or interface. Extension methods can be added to your own custom class, .NET framework classes, or third-party classes or interfaces.

It is introduced in *C# 3.0*. C# extension method is a static method of a static class, where the "this" modifier is applied to the first parameter. The type of the first parameter will be the type that is extended.

Extension methods are only in scope when you explicitly import the namespace into your source code with a using directive.

Conclusion: C# offers rich support for both multiple interface and single implementation inheritance, Due to inheritance support the programmer will reuse the code and make the application portable.

Experiment No-05

Title:- Implementation of Multidimensional & Jagged array in C#.

Aim : Implement the program to demonstrate the array based manipulation on simple array, multidimensional array and jagged array.

Objective :-

1. Study the declaration and initialization of array
2. Study the multidimensional array & its declaration.
3. Study the jagged array and its implementation.

Theory:

Array Declaration:

An array is declared by defining the type of the elements inside the array followed by empty

brackets and a variable name;

```
int[] myArray;
```

Array Initialization

After declaring an array, memory must be allocated to hold all the elements of the array. An array is a reference type, so memory on the heap must be allocated. You do this by initializing the variable of the array using the new operator with the type and the number of elements inside the array.

```
myArray = new int[4];
```

Instead of using a separate line for the declaration and initialization, you can declare and initialize an array in a single line:

```
int[] myArray = new int[4];
```

You can also assign values to every array element using an array initialize.

```
int[] myArray = new int[4] {4, 7, 11, 2};
```

If you initialize the array using curly brackets, the size of the array can also be left out, because the compiler can count the number of elements itself:

```
int[] myArray = new int[] {4, 7, 11, 2};
```


Accessing Array Elements

You can access the array elements using an indexer. Arrays only support indexers that have integer parameters.

With the indexer, you pass the element number to access the array. The indexer always starts with a value of 0 for the first element.

the array `myArray` is declared and initialized with four integer values. The elements can be accessed with indexer values 0, 1, 2, and 3.

```
int[] myArray = new int[] {4, 7, 11, 2};  
int v1 = myArray[0]; // read first element
```

Multidimensional Arrays

A multidimensional array is indexed by two or more integers.

2 - dimensional array that has three rows and three columns. The first row has the values 1, 2, and 3, and the third row has the values 7, 8, and 9.

1, 2, 3

4, 5, 6

7, 8, 9

Declaring this 2 - dimensional array with C# is done by putting a comma inside the brackets. The array is initialized by specifying the size of every dimension (also known as rank). Then the array elements can be accessed by using two integers with the indexer:

```
int[,] twodim = new int[3, 3];
```

You can also initialize the 2 - dimensional array by using an array indexer if you know the value for the elements in advance.

```
int[,] twodim = {  
    {1, 2, 3},  
    {4, 5, 6},  
    {7, 8, 9}  
};
```

Jagged Arrays

A jagged array is more flexible in sizing the array. With a jagged array every row can have a different size. The jagged array shown contains three rows where the first row has two elements, the second row has six elements, and the third row has three elements.

Two-Dimensional Array Jagged Array

1 2 1 2

3 4 5 6 7 8

9 10 11

3

4 5 6

7 8 9

A jagged array is declared by placing one pair of opening and closing brackets after another. With the initialization of the jagged array, only the size that defines the number of rows in the first pair of brackets is set. The second brackets that define the number of elements inside the row are kept empty because every row has a different number of elements. Next, the element number of the rows can be set for every row:

```
int[][] jagged = new int[3][];
```

```
jagged[0] = new int[2] { 1, 2 };
```

```
jagged[1] = new int[6] { 3, 4, 5, 6, 7, 8 };
```

```
jagged[2] = new int[3] { 9, 10, 11 };
```

Conclusion:

The C# language support simple, multidimensional, and jagged arrays. Here we understand the different ways to initialize and declare the different types of array.

Experiment No-06

TITLE:- Implementation of Operator overloading in C#.

Aim : Impelement the program to demonstate the operator overloading.

Objective : 1. Study the different types of operator in C# environment.
2. Study and implement operator overloading in C#.

Prerequisite: - Student should know the concept of operator overloading in CPP.

Theory :-

Operator – An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. C# has rich set of built-in operators and provides the following type of operators:

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Misc Operators

Operator Overloading

We can redefine or overload most of the built-in operators available in C#. Thus a programmer can use operators with user-defined types as well. Overloaded operators are functions with special names the keyword operator followed by the symbol for the operator being defined. Like any other function, an overloaded operator has a return type and a parameter list.

For example, look at the following function:

```
public static Box operator+ (Box b, Box c)
{
    Box box = new Box();
    box.length = b.length + c.length;
    box.breadth = b.breadth + c.breadth;
    box.height = b.height + c.height;
    return box;
}
```

The above function implements the addition operator (+) for a user-defined class Box. It adds the attributes of two Box objects and returns the resultant Box object.

Overloadable and Non-Overloadable Operators

The following table describes the overload ability of the operators in C#:

Operators	Description
+, -, !, ~, ++, --	These unary operators take one operand and can be overloaded.
+, -, *, /, %	These binary operators take one operand and can be overloaded.
==, !=, <, >, <=, >=	The comparison operators can be overloaded
&&,	The conditional logical operators cannot be overloaded directly.

CONCLUSION:

Here we understand the different types of operator are available in C# environment and how to overload the operator to use in different manner.

Experiment No-07

Title:- Implementation of string manipulation application using String & String builder class.

Aim : Implement the program to demonstrate different string manipulation operations.

Objective:

1. Study the methods and property supported by string class.
2. Study the StringBuilder class and its supporting method & property.
3. Study how to format the string.

Prerequisite: - Student should know concept of String handling in C, CPP and Java

Theory:

System. String

C# treats strings as first-class types that are flexible, powerful, and easy to use. Each string object is an *immutable* sequence of Unicode characters. In other words, methods that appear to change the string actually return a modified copy; the original string remains intact. When you declare a C# string using the `string` keyword, you are in fact declaring the object to be of the type `System.String`, one of the built-in types provided by the .NET Framework Class Library. A C# string type *is* a `System.String` type.

It's immutable

You can never actually change the contents of a string, at least with safe code which doesn't use reflection. Because of this, you often end up changing the value of a string variable. For instance, the code `s = s.Replace ("foo", "bar");` doesn't change the contents of the string that `s` originally referred to - it just sets the value of `s` to a new string, which is a copy of the old string but with "foo" replaced by "bar".

The `String` class provides a host of methods for comparing, searching, and manipulating strings, as shown in table.

Method	Purpose
Compare	Compares the contents of strings, taking into account the culture (locale) in assessing equivalence between certain characters
CompareOrdinal	Same as Compare but doesn't take culture into account
Concat	Combines separate string instances into a single instance
CopyTo	Copies a specific number of characters from the selected index to an entirely new instance of an array
Format	Formats a string containing various values and specifiers for how each value should be formatted
IndexOf	Locates the first occurrence of a given substring or character in the string
IndexOfAny	Locates the first occurrence of any one of a set of characters in the string
Insert	Inserts a string instance into another string instance at a specified index
Join	Builds a new string by combining an array of strings
LastIndexOf	Same as IndexOf but finds the last occurrence
LastIndexOfAny	Same as IndexOfAny but finds the last occurrence
PadLeft	Pads out the string by adding a specified repeated character to the left side of the string
Split	Splits the string into an array of substrings, the breaks occurring wherever a given character occurs
Substring	Retrieves the substring starting at a specified position in the string
ToLower	Converts string to lowercase
ToUpper	Converts string to uppercase
Trim	Removes leading and trailing whitespace

Building Strings

As you have seen, String is an extremely powerful class that implements a large number of very useful methods. However, the String class has a shortcoming that makes it very inefficient for making repeated modifications to a given string — **it is actually an *immutable data type***, which means that once you initialize a string object, that string object can never change. The methods and operators that appear to modify the contents of a string actually create new strings, copying across the contents of the old string if necessary.

The following table lists the main StringBuilder methods.

Method	Purpose
Append()	Appends a string to the current string
AppendFormat()	Appends a string that has been worked out from a format specifier
Insert()	Inserts a substring into the current string
Remove()	Removes characters from the current string
Replace()	Replaces all occurrences of a character with another character or a substring with another substring in the current string
ToString()	Returns the current string cast to a <code>System.String</code> object (overridden from <code>System.Object</code>)

Format Strings

To implement a `ToString()` method in order to be able to display the contents of a given variable. However, quite often users might want the contents of a variable to be displayed in different, often culture - and locale - dependent, ways. The .NET base class, `System.DateTime`, provides the most obvious example of this. For example, you might want to display the same date as 10 June 2008, 10 Jun 2008, 6/10/08 (USA), 10/6/08 (UK), or 10.06.2008 (Germany).

The following table lists the common format specifiers for the numeric types, which were briefly discussed

Specifier	Applies To	Meaning	Example
C	Numeric types	Locale-specific monetary value	\$4834.50 (USA) £4834.50 (UK)
D	Integer types only	General integer	4834
E	Numeric types	Scientific notation	4.834E+003
F	Numeric types	Fixed-point decimal	4384.50
G	Numeric types	General number	4384.5
N	Numeric types	Common locale-specific format for numbers	4,384.50 (UK/USA) 4 384,50 (continental Europe)
P	Numeric types	Percentage notation	432,000.00%
X	Integer types only	Hexadecimal format	1120 (If you want to display 0x1120, you will have to write out the 0x separately)

Conclusion:

Here we understand the importance of string and how to use the string data type and manipulate it in your applications. When working with strings in the past, it was quite common to just slice and dice the strings as needed using concatenation. With the .NET Framework, you can use the `StringBuilder` class to accomplish a lot of this task with better performance.

Experiment No-08

TITLE:- Develop an application using Regex.Matches method and Regular Expression pattern matching.

Aim : Implement the program to demonstrate Regular Expression pattern matching.

Objective : 1. Study the regular expression class with supporting methods to validate the fields.

2. Implement an application using regular expression class.

Prerequisite: - Student should know concept of Exception Handling in C, CPP and Java

Theory:- A regular expression is a pattern that could be matched against an input text. The .Net framework provides a regular expression engine that allows such matching. A pattern consists of one or more character literals, operators, or constructs.

Constructs for Defining Regular Expressions

- Character escapes
- Character classes
- Anchors
- Grouping constructs
- Quantifiers
- Backreference constructs
- Alternation constructs
- Substitutions
- Miscellaneous constructs

The Regex Class

Sr. No.	Methods
1	public bool IsMatch(string input) Indicates whether the regular expression specified in the Regex constructor finds a match in a specified input string.
2	public bool IsMatch(string input, int startat) Indicates whether the regular expression specified in the Regex constructor finds a match in the specified input string, beginning at the specified starting position in the string.
3	public static bool IsMatch(string input, string pattern) Indicates whether the specified regular expression finds a match in the specified input string.
4	public MatchCollection Matches(string input) Searches the specified input string for all occurrences of a regular expression.
5	public string Replace(string input, string replacement) In a specified input string, replaces all strings that match a regular expression pattern with a specified replacement string.

The above Regex class is used for representing a regular expression. It has the above commonly used methods. For example, suppose we wanted to find words beginning with n. We could use the escape sequence \b, which indicates a word boundary (a word boundary is just a point where an alphanumeric character precedes or follows a whitespace character or punctuation symbol). We would write this:

```
string Pattern = @"\\bn";
```

```
MatchCollection Matches = Regex.Matches(Text, Pattern, RegexOptions.IgnoreCase |
RegexOptions.ExplicitCapture);
```

Notice the @ character in front of the string. We want the \b to be passed to the .NET regular expressions engine at runtime—we don't want the backslash intercepted by a well-meaning C# compiler that thinks it's an escape sequence intended for itself. If we want to find words ending with the sequence ion, then we write this:

```
string Pattern = @"\\ion\\b";
```

If we want to find all words beginning with the letter a and ending with the sequence ion, we will have to put a bit more thought into our code. We clearly need a pattern that begins with \ba and ends with ion\b, but what goes in the middle. We need to somehow tell the application that between the n and the ion there can be any number of characters as long as none of them are whitespace. In fact, the correct pattern looks like this:

```
string Pattern = @"\\ba\\S*ion\\b";
```

The following table lists the main special characters/escape sequences that we use

Symbol	Meaning	Example	Matches
^	Beginning of input text	^B	B, but only if first character in text
\$	End of input text	x\$	x, but only if last character in text
.	Any single character except the newline character (\n)	i.ation	isation, ization
*	Preceding character may be repeated 0 or more times	ra*t	rt, rat, raat, raaat, and so on
+	Preceding character may be repeated 1 or more times	ra+t	rat, raat, raaat and so on, (but not rt)
?	Preceding character may be repeated 0 or 1 times	ra?t	rt and rat only
\\s	Any whitespace character	\\sa	[space]a, \\ta, \\na (\\t and \\n have the same meanings as in C#)
\\S	Any character that isn't a whitespace	\\SF	aF, rF, cF, but not \\tf
\\b	Word boundary	ion\\b	Any word ending in ion
\\B	Any position that isn't a word boundary	\\BX\\B	Any x in the middle of a word

Conclusion: Here we study how to create pattern using regular expression and different methods and property supported by regex class use for pattern matching.

Experiment No-09

Title:- : Implementation of Windows Form based application using different controls.

Aim : Implement the Window Form Application to perform different tasks using different controls.

Objective :-

1. Study the different controls to design Windows Form.
2. Implement the window form and demonstrate the event handling for different controls.

Theory :-

The Visual Studio .NET on a Windows platform gives you a multitude of classes to easily create typical Windows GUI applications. If you elect to use these features, currently a C# application will only run on a Windows machine. There is similar functionality for C# GUI applications under Mono. . To create a Windows Forms Application, start Visual Studio .NET and create a new Visual C# Project. Make sure you select a Windows Application as the template.

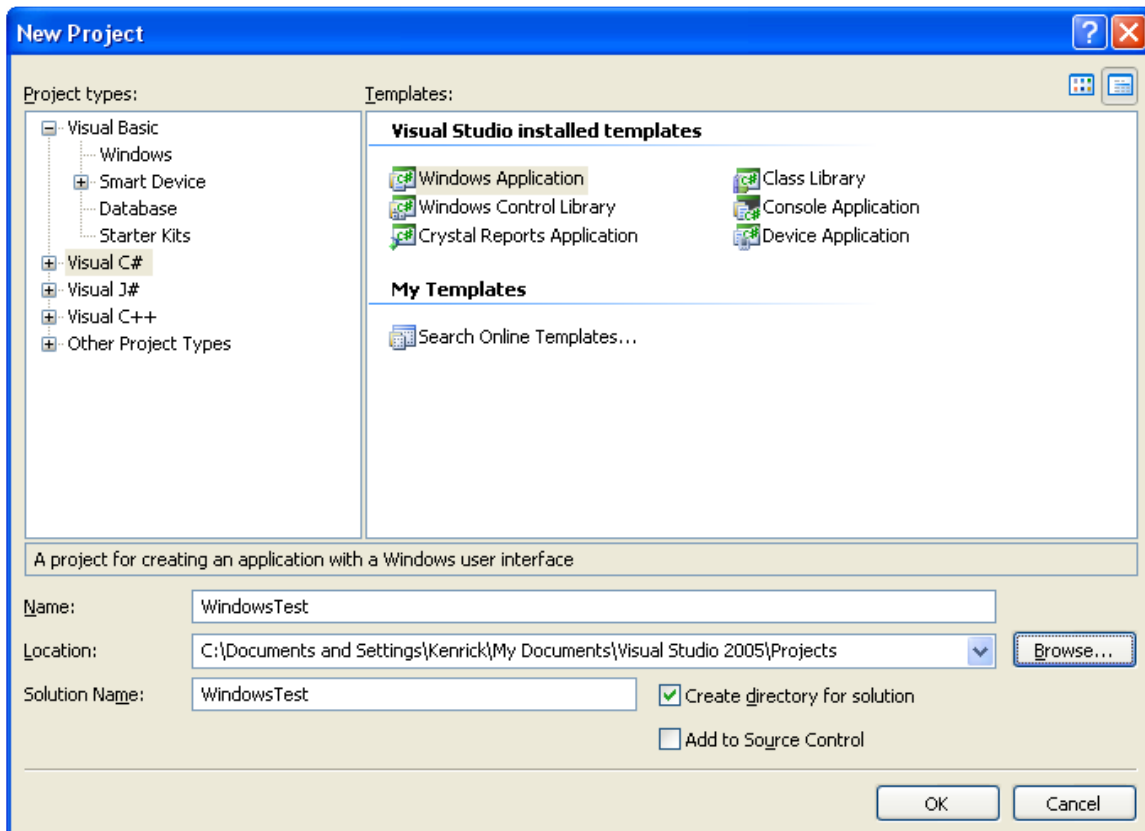
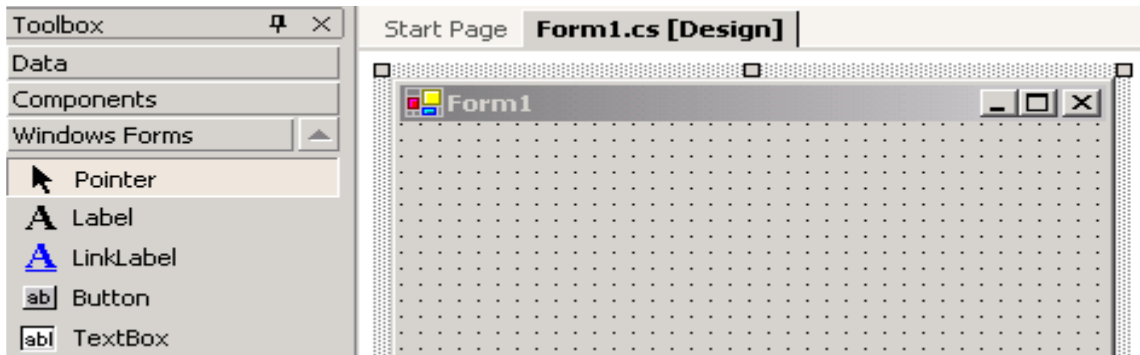
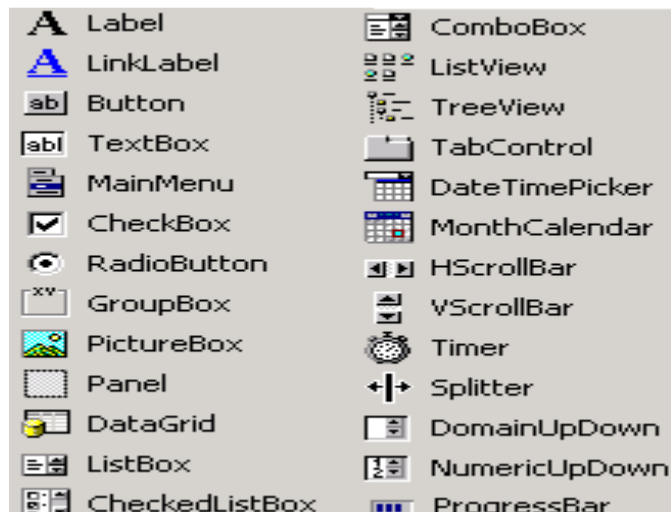


Figure – New Project Creation Wizard

You will be presented with a blank form. This document assumes that you have the Toolbox pane visible on the left of the screen and it is set to Windows Forms. This pane contains common controls which make up the elements on the form. The layout is shown below:

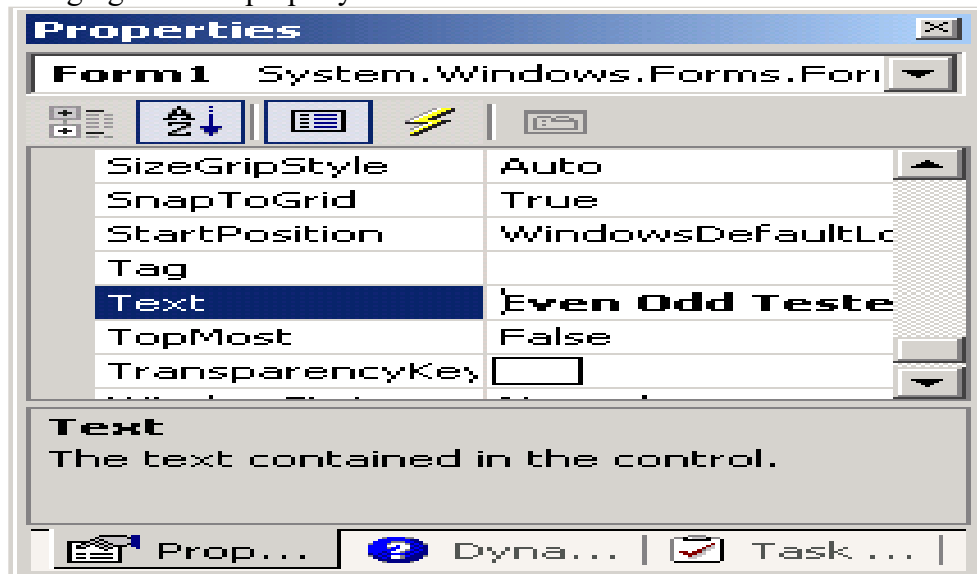


A control is a component with a visual representation. The Control class in the System.Windows.Forms namespace is the base class for the graphical objects on the screen. All controls are derived from the base control. Examples of controls include:



A Form is a type of Container Control. It can contain other controls and we can add controls to it. In this document we will only discuss a few controls: Buttons, Labels, TextBox, ProgressBar, and PictureBox. A Button is just like it sounds, a button we can press and when it is pressed some code is invoked. A Label is a way to output some text on the form. A TextBox can be used to output data and provide a way for the user to enter text to the program (although a label is better for data that is purely output only). A ProgressBar displays a graphical bar of progress for some slow event. Finally, a PictureBox is used to load and display an image. Please refer to the .NET documentation for details about the other controls. Let's make a simple Windows application that will allow the user to enter a number into a TextBox and press a button. Upon pressing the button we will output if the number is even or odd into a label. First, select the form by clicking on it. In the Properties Window, you can click and modify various properties of the form. By default, it is called Form1. Scroll through the properties; there are many available to change, such as the size, icon, width, locked, etc. One we can change is the

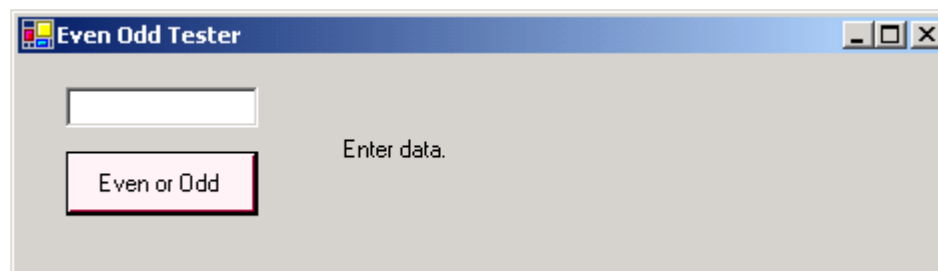
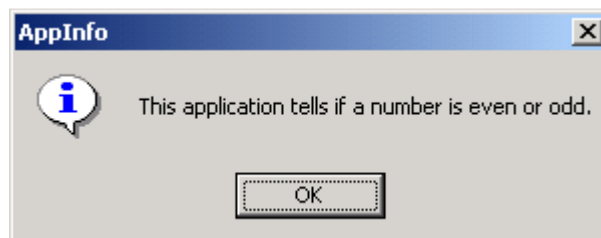
title bar shown with the form. Here we change the title bar of the form to “Even Odd Tester” by changing the Text property.



Any code we enter in here is executed when the form is loaded for the first time. In our case, this corresponds to when the application is initially started. Try adding the following code:

```
private void Form1_Load(object sender, EventArgs e)
{
    this.btnEvenOdd.BackColor = Color.LavenderBlush;
    MessageBox.Show("This app tells if a number is even or odd.",
        "AppInfo",
        MessageBoxButtons.OK,
        MessageBoxIcon.Information);
}
```

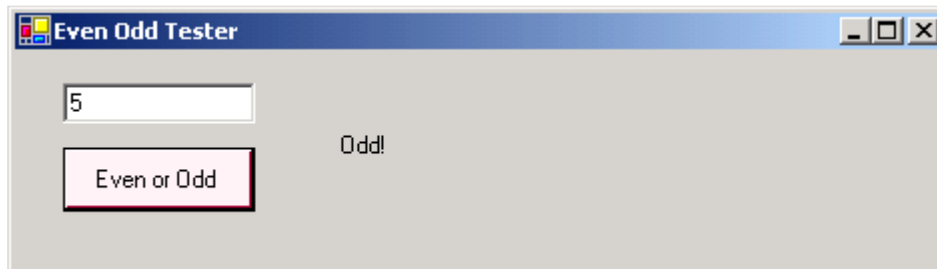
This code will set the background color of the button to LavenderBlush upon startup. It will also display a message box with the information “This app tells if a number is even or odd” with the title “AppInfo” and display an OK button and an “Information” icon:



Now, let's add some code back to the buttonClicked event. We would like to take whatever data is entered in the textbox, convert it to an Integer, and then change the label's text to indicate if the integer is even or odd. The following code will do all of this. In this case we have added a try/catch block in the event of an error converting the integer (but ignore what kind of error may have occurred):

```
private void btnEvenOdd_Click(object sender, EventArgs e)
{
    int i;
    try
    {
        i = int.Parse(textBoxData.Text);
        if ((i % 2) == 0)
        {
            labelResult.Text = "Even!";
        }
        else
        {
            labelResult.Text = "Odd!";
        }
    }
    catch (Exception err)
    {
        labelResult.Text = "Error, invalid number.";
    }
}
```

Here is a screen shot of an output case:



CONCLUSION:-

The Visual Studio .NET on a Windows platform gives you a multitude of classes to easily create typical Windows GUI applications and using that classes we implement attractive window form.

Experiment No-10

Title:- : Implementation of Windows Form based MDI application with different controls.

Aim : Implement the Windows Form based MDI application

Objective :- 1. Design Windows Form based MDI application using different controls.
2. Implement Windows Form based MDI application

Theory :-

Multiple Document Interface (MDI)

MDI-type applications are used when you have an application that can show either multiple instances of the same type of form or different forms that must be contained in some way. An example of multiple instances of the same type of form is a text editor that can show multiple edit windows at the same time. An example of the second type of application is Microsoft Access. You can have query windows, design windows, and table windows all open at the same time. The windows never leave the boundaries of the main Access application.

To create a MDI application you must have at least two forms in your project. One is the MdiParent and the other is the MdiChild. Let's look at an example of how MDI applications work. Create a new C# Windows Application project. Instead of allowing a form to be the start-up class, add a new class and call it StartUp. In the Startup class add a Main method. Be sure to set the StartUp object property in the Project Properties dialog box to the StartUp class. After adding the Main method your Startup class looks like this:

```
using System;
using System.Windows.Forms;
namespace SimpleMDIApp
{
    /// <summary>
    /// Summary description for StartUp.
    /// </summary>
    public class StartUp
    {
        [STAThread]
        static void Main()
        {
            Application.Run(new ParentForm());
        }
    }
}
```

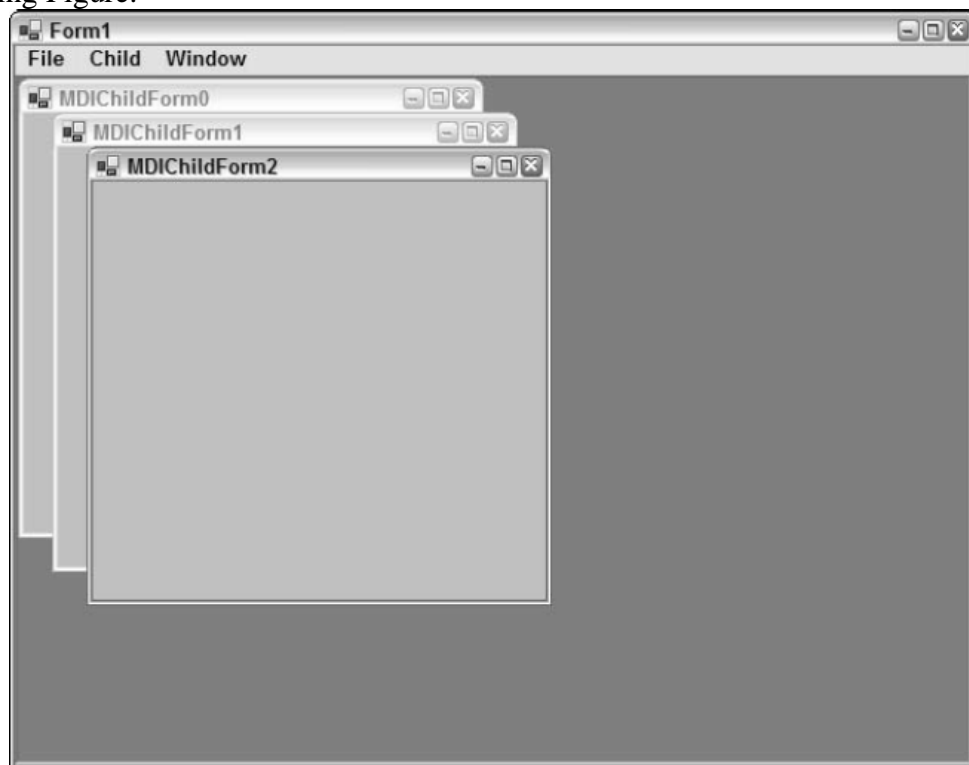
Next you must either add a new Form class and name it ParentForm or rename the class Form1 to ParentForm. If you rename Form1 be sure to delete the Main method from it since you will be starting the application from the StartUp class. You have to tell ParentForm that it is indeed a parent form for MDI children forms. Setting IsMdiContainer to true will do this. If you have the form in the designer you'll notice that the background turns a dark gray color. This is to let you know that this is a MDI parent form.

Next you must create child forms. Add a new form to the project and call it ChildForm. You can add a couple of controls to it if want. Add another new form and call it AnotherChildForm. You can also add controls to this form if you want. Currently these are standard forms that might appear in any project. There is nothing at design time that determines that these forms are children of a MDI parent form. This is done at runtime by setting the MdiParent property of the form.

To see how this works you must add a MainMenu control to the ParentForm. Add a File menu at the top level and below it add a New menu option. In the New Click event handler you can instantiate a new ChildForm and show it. Here is the code:

```
private void mnuFileNew_Click(object sender, System.EventArgs e)
{
    ChildForm frm = new ChildForm();
    frm.Name = string.Concat("MDIChildForm",
    this.MdiChildren.Length.ToString());
    frm.Text = frm.Name;
    frm.MdiParent = this;
    frm.Show();
}
```

A new ChildForm is created. Assign it to the object variable frm, and then assign a name to the new form by concatenating the string MDIChildForm with the length of the MdiChildren property of the parent form. The MdiChildren property is an Array of the current open MDI children form in the parent. Next, set the form Text equal to the new Name you generated. This is what will show in the caption bar of the form. The next line is where you set the MdiParent property of the child form to the parent form. Now the ChildForm is truly a MDI child. And the last thing you do is show the form. If you click the New menu choice three times, you should end up with a screen that resembles following Figure.



Custom Controls

There are a number of ways to create a control. You can start from scratch, deriving your class from either Control, ScrollableControl, or ContainerControl. You will have to override the Paint event and do all of your drawing, not to mention adding the functionality that your control is supposed to provide. If the control is supposed to be an enhanced version of a current control, the thing to do is to derive from the control that is being enhanced. You can add attributes to the custom control that will enhance the design time capabilities of the control like BindableAttribute, BrowsableAttribute, BrowsabilityAttribute, DefaultEventAttribute and DefaultPropertyAttribute.

User control

User controls are one of the more powerful features of Windows Forms. They allow encapsulating user interface designs into nice reusable packages that can be plugged into project after project. It is not uncommon for an organization to have a couple of libraries of frequently used user controls. Not only can user interface functionality be contained in user controls but common data validation can be incorporated in them as well. Things like formatting phone numbers or id numbers. A predefined list of items can be in the user control for fast loading of a list box or combo box. State codes or country codes fit into this category. Incorporating as much functionality that does not depend on the current application as possible into a user control makes the control that much more useful in the organization.

Conclusion:

Here we implement the Windows Form based MDI application using custom and user controls in which we show either multiple instances of the same type of form or different forms that must be contained.

Experiment No-11

Title:- : Implementation of Windows Form based application with Database connectivity with all field validation.

Aim : Implement the application which handle the database related operations.

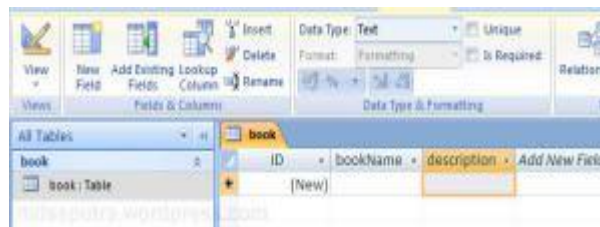
Objective :-

1. Study the different classes are use for database connectivity.
2. Implement the application to demonstrate the database connectivity and database realted operations.
3. Study the Data Access with ADO.NET

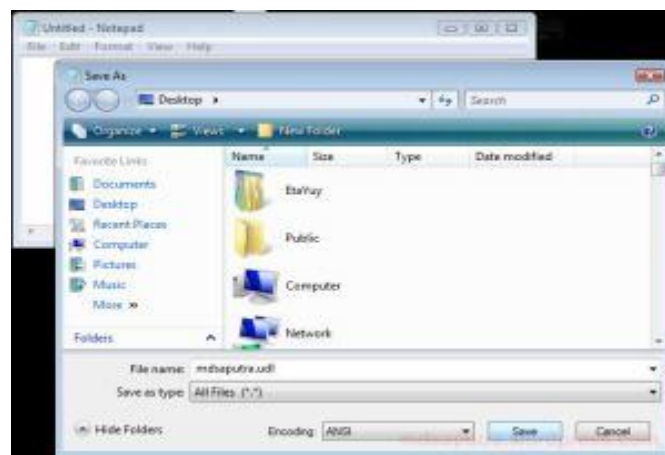
Theory: -

Connect to Access Database in C#

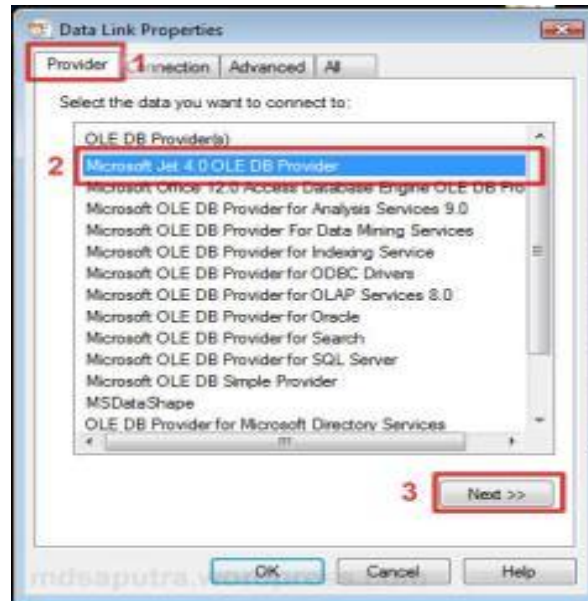
At first simply create Access database (e.g. book.mdb) like picture below, their column name and data type is ID (Autonumber), bookName (Text), description (Text).



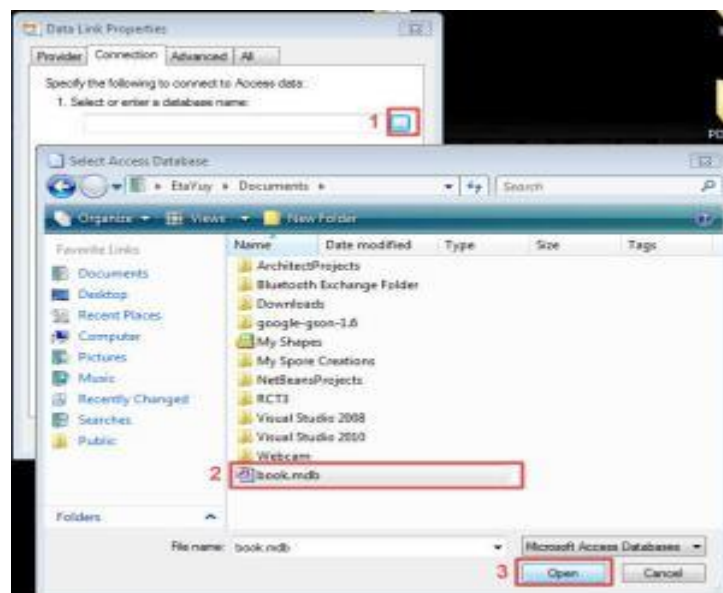
After that save your database and create connection parameter which will be use in the code. Now open notepad.exe , then simply save as it to mdsaputra.udl , don't forget to change Save as type into All Files (*.*)).



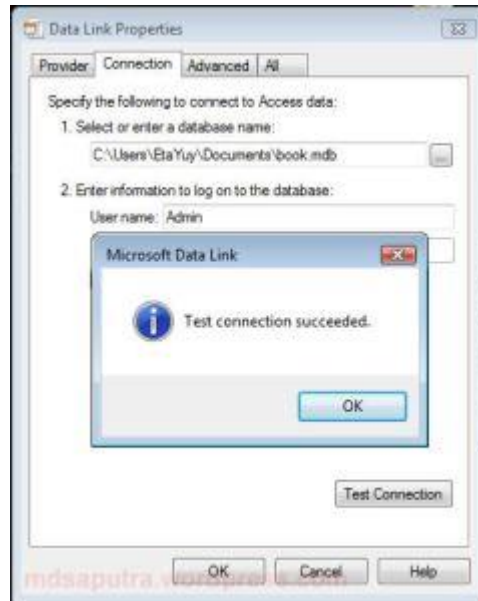
Now close your notepad and double click mdsaputra.udl which you just created. You are going to see wizard like picture below:



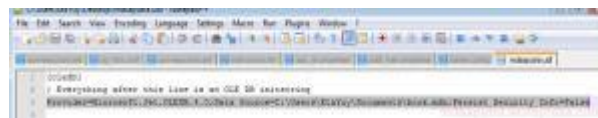
Click Provider Tab (1), select Microsoft Jet 4.0 OLE DB Provider (2) then click Next (3). Now you are in Connection Tab.



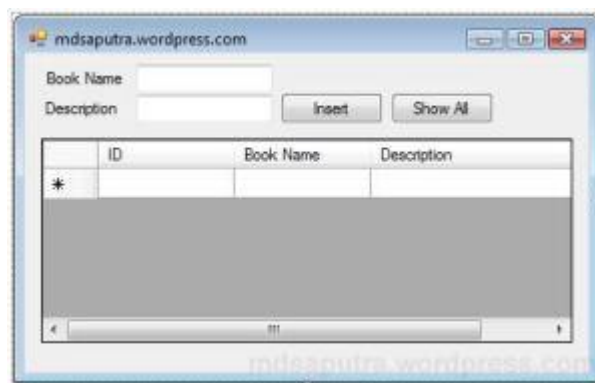
Click ... /browse button (1), select book.mdb (2), open (3) and then click Test Connection; if you do everything properly you must see Test Connection Succeed like picture below:



Click OK, and now reopen mdsaputra.udl with notepad, you gonna see some provider properties like shown below :



Text I highlight above is **provider properties** that we gonna need in our code, for now just close it. We are done with database and it connection things, now open your Visual Studio and create Visual C#, Windows Forms Application. Fill and arrange your form with Label, TextBox, Button and DataGridView like shown below.



Double click your Insert Button and Show All button to auto create Event Handler method. Now, inside your Form.cs add code that shown below:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
```

```

using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.OleDb;
namespace TutorialConnectToAccessDB
{
    public partial class FormMain : Form
    {
        private OleDbConnection bookConn;
        private OleDbCommand oleDbCmd = new OleDbCommand();
        //parameter from mdsaputra.udl
        private String connParam = @"Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\Users\EtaYuy\Documents\book.mdb;Persist Security Info=False";
        public FormMain()
        {
            //create connection using parameter from mdsaputra.udl
            bookConn = new OleDbConnection(connParam);
            InitializeComponent();
        }

        private void buttonInsert_Click(object sender, EventArgs e)
        {
            bookConn.Open();
            oleDbCmd.Connection = bookConn;
            oleDbCmd.CommandText = "insert into book (bookName, description) values ("
+ this.textBoxBookName.Text + "','" + this.textBoxDescription.Text + "');"
            int temp = oleDbCmd.ExecuteNonQuery();
            if (temp > 0)
            {
                textBoxBookName.Text = null;
                textBoxDescription.Text = null;
                MessageBox.Show("Record Successfully Added");
            }
            else
            {
                MessageBox.Show("Record Fail to Added");
            }
            bookConn.Close();
        }

        private void buttonShowAll_Click(object sender, EventArgs e)
        {
            dataGridView1.DataSource = null;
            dataGridView1.Rows.Clear();
            dataGridView1.Refresh();

            OleDbDataAdapter dAdapter = new OleDbDataAdapter("select * from book",
connParam);

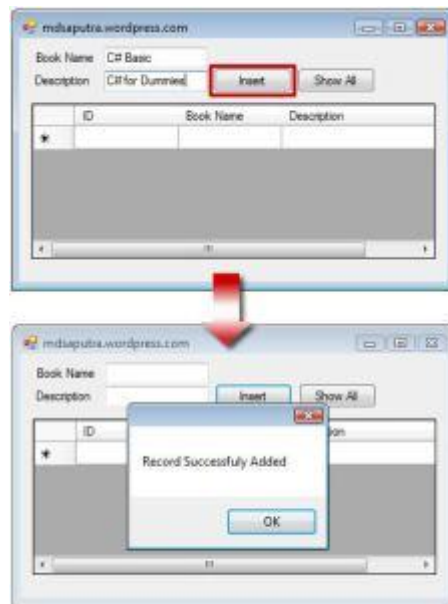
```

```

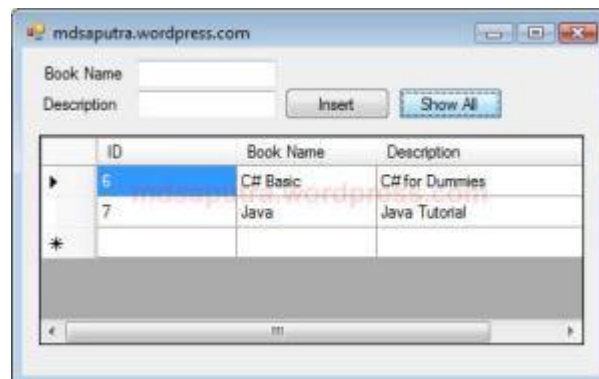
OleDbCommandBuilder cBuilder = new OleDbCommandBuilder(dAdapter);
DataTable dataTable = new DataTable();
DataSet ds = new DataSet();
dAdapter.Fill(dataTable);
for (int i = 0; i < dataTable.Rows.Count; i++)
{
    dataGridView1.Rows.Add(dataTable.Rows[i][0], dataTable.Rows[i][1],
dataTable.Rows[i][2]);
} } }

```

See highlighted code above, that connParameter assigned by provider properties from mdsaputra.udl. Now everything is done, if you following my instruction carefully, the program should be able Insert new Record into book.mdb like shown below,



And also can show all data you have been insert into book.mdb like picture below.



Data Access with .NET - ADO.NET

The ADO.NET is more than just a thin veneer over some existing API. The similarity to ADO is fairly minimal—the classes and methods of accessing data are completely different. ADO (ActiveX Data Objects) is a library of COM components that has had many incarnations over the last few years. Currently at version 2.7, ADO consists primarily of the Connection, Command, Recordset, and Field objects. Using ADO, a connection is opened to the database, some data is selected into a record set consisting of fields, that data is then manipulated and updated on the server, and the connection is closed. ADO also introduced a so-called disconnected record set, which is used when keeping the connection open for long periods of time is not desirable.

Namespaces

- System.Data—All generic data access classes
- System.Data.Common—Classes shared (or overridden) by individual data providers
- System.Data.Odbc—ODBC provider classes
- System.Data.OleDb—OLE DB provider classes
- System.Data.Oracle—Oracle provider classes
- System.Data.SqlClient—SQL Server provider classes
- System.Data.SqlTypes—SQL Server data types

Shared Classes

The following classes are contained in the System.Data namespace:

- DataSet—This object is designed for disconnected use and can contain a set of DataTables and include relationships between these tables.
- DataTable—A container of data that consists of one or more DataColumnns and, when populated, will have one or more DataRowns containing data.
- DataRow—A number of values, akin to a row from a database table, or a row from a spreadsheet.
- DataColumn—This object contains the definition of a column, such as the name and data type.
- DataRelation—A link between two DataTable classes within a DataSet class. Used for foreign key and master/detail relationships.

Database-Specific Classes

SqlCommand, OleDbCommand, OracleCommand, and OdbcCommand—Used as wrappers for SQL statements or stored procedure calls.

- ❑ SqlCommandBuilder, OleDbCommandBuilder, OracleCommandBuilder, and OdbcCommandBuilder—Used to generate SQL commands (such as INSERT, UPDATE, and DELETE statements) from a SELECT statement.
- ❑ SqlConnection, OleDbConnection, OracleConnection, OdbcConnection—Used to connect to the database. Similar to an ADO Connection.
- ❑ SqlDataAdapter, OleDbDataAdapter, OracleDataAdapter, OdbcDataAdapter—Used to hold select, insert, update, and delete commands, which are then used to populate a DataSet and update the Database.
- ❑ SqlDataReader, OleDbDataReader, OracleDataReader, OdbcDataReader—Used as a forward only, connected data reader.
- ❑ SqlParameter, OleDbParameter, OracleParameter, OdbcParameter—Used to define a parameter to a stored procedure.
- ❑ SqlTransaction, OleDbTransaction, OracleTransaction, OdbcTransaction—Used for a database transaction, wrapped in an object.

Commands

A command can be constructed by passing the SQL clause as a parameter to the constructor of the Command class, as shown in this example:

```
string source = "server=(local)\\NetSDK;" +
"integrated security=SSPI;" +
"database=Northwind";
string select = "SELECT ContactName,CompanyName FROM Customers";
SqlConnection conn = new SqlConnection(source);
conn.Open();
SqlCommand cmd = new SqlCommand(select, conn);
```

The <provider>Command classes have a property called CommandType, which is used to define whether the command is a SQL clause, a call to a stored procedure, or a full table statement.

Executing Commands

After you have defined the command, you need to execute it. There are a number of ways to issue the statement, depending on what you expect to be returned (if anything) from that command. The <provider>Command classes provide the following execute methods:

- ❑ ExecuteNonQuery()—Executes the command but does not return any output
- ❑ ExecuteReader()—Executes the command and returns a typed IDataReader
- ❑ ExecuteScalar()—Executes the command and returns a single value

In addition to these methods, the SqlCommand class also exposes the following method

- ❑ ExecuteXmlReader()—Executes the command and returns an XmlReader object, which can be used to traverse the XML fragment returned from the database.

Conclusion: Here, we implement the window form application with database connectivity in which we understand the different classes, data source which are used to perform database related operations.

Experiment No-12

Title:- : Implement the console based networking application to obtain network information.

Aim : Implement the application to obtain the network information and detect changes in network connectivity.

Objective :- 1. Study the use of NetworkInterface class to obtain network information.
2. Study the NetworkChange class to detect the change in network connectivity.

Theory : -

Obtain Information About the Local Network Interface

The static method GetAllNetworkInterfaces of the System.Net.NetworkInformation.NetworkInterface class to get an array of objects derived from the abstract class NetworkInterface. Each object represents a network interface available on the local machine. Use the members of each NetworkInterface object to retrieve configuration information and network statistics for that interface. The System.Net.NetworkInformation namespace provides easy access to information about network configuration and statistics. The primary means of retrieving network information are the properties and methods of the NetworkInterface class. You do not instantiate NetworkInterface objects directly. Instead, you call the static method NetworkInterface.GetAllNetworkInterfaces, which returns an array of NetworkInterface objects. Each object represents a single network interface on the local machine. You can then obtain network information and statistics about the interface using the NetworkInterface members in table.

Detect Changes in Network Connectivity

The NetworkChange class provides an easy-to-use mechanism that allows applications to be aware of changes to network addresses and general network availability. This allows your applications to adapt dynamically to the availability and configuration of the network. The NetworkAvailabilityChanged event fires when a change occurs to general network availability. An instance of the NetworkAvailabilityChangedEventHandler delegate is needed to handle this event and is passed a NetworkAvailabilityEventArgs object when the event fires. The NetworkAvailabilityEventArgs.IsAvailable property returns a bool indicating whether the network is available or unavailable following the change. The NetworkAddressChanged event fires when the IP address of a network interface changes. An instance of the NetworkAddressChangedEventHandler delegate is required to handle these events. No event-specific arguments are passed to the event handler, which must call NetworkInterface.GetAllNetworkInterfaces to determine what has changed and to take appropriate action.

Member	Description
Properties	
Description	Gets a string that provides a general description of the interface.
Id	Gets a string that contains the identifier of the interface.
IsReceiveOnly	Gets a bool indicating whether the interface can only receive or can both send and receive data.
Name	Gets a string containing the name of the interface.
NetworkInterfaceType	Gets a value from the <code>System.Net.NetworkInformation.NetworkInterfaceType</code> enumeration that identifies the type of interface. Common values include <code>Ethernet</code> , <code>FastEthernetT</code> , and <code>Loopback</code> .
OperationalStatus	Gets a value from the <code>System.Net.NetworkInformation.OperationalStatus</code> enumeration that identifies the status of the interface. Common values include <code>Down</code> and <code>Up</code> .
Speed	Gets a long that identifies the speed (in bits per second) of the interface as reported by the adapter, not based on dynamic calculation.
SupportsMulticast	Gets a bool indicating whether the interface is enabled to receive multicast packets.

Methods

GetIPProperties	Returns a <code>System.Net.NetworkInformation.IPInterfaceProperties</code> object that provides access to the TCP/IP configuration information for the interface. Properties of the <code>IPInterfaceProperties</code> object provide access to WINS, DNS, gateway, and IP address configuration.
-----------------	---

Member	Description
GetIPv4Statistics	Returns a <code>System.Net.NetworkInformation.IpV4InterfaceStatistics</code> object that provides access to the TCP/IP v4 statistics for the interface. The properties of the <code>IPv4InterfaceStatistics</code> object provide access to information about bytes sent and received, packets sent and received, discarded packets, and packets with errors.
GetPhysicalAddress	Returns a <code>System.Net.NetworkInformation.PhysicalAddress</code> object that provides access to the physical address of the interface. You can obtain the physical address as a byte array using the method <code>PhysicalAddress.GetAddressBytes</code> or as a string using <code>PhysicalAddress.ToString</code> .
Supports	Returns a bool indicating whether the interface supports a specified protocol. You specify the protocol using a value from the <code>System.Net.NetworkInformation.NetworkInterfaceComponent</code> enumeration. Possible values include <code>IPv4</code> and <code>IPv6</code> .

Conclusion – Here we understand how to obtain the network information and their change in network through console based application.

Experiment No-13

Title:- : Develop a Windows form application to download the file & process it using stream.

Aim : Implement the application to downloading content from the Web using a stream

Objective :-

1. Study the `WebRequest` and `WebResponse` class
2. Implement the window form to retrieve a file from a web site.

Theory : -

Download a File and Process It Using a Stream :

Use the `System.Net.WebRequest` class to create your request, the `System.Net.WebResponse` class to retrieve the response from the web server, and some form of reader (typically a `System.IO.StreamReader` for HTML or text data or a `System.IO.BinaryReader` for a binary file) to parse the response data.

The opening and downloading a stream of data from the Web using the `WebRequest` and `WebResponse` classes takes the following four basic steps:

1. Use the static method `Create` of the `WebRequest` class to specify the page you want. This method returns a `WebRequest`-derived object, depending on the type of URI you specify. For example, if you use an HTTP URI (with the scheme `http://` or `https://`), you will create an `HttpWebRequest` instance. If you use a file system URI (with the scheme `file://`), you will create a `FileWebRequest` instance. In the .NET Framework 2.0 and later, you can also use an FTP URL (with the scheme `ftp://`), which will create an `FtpWebRequest`.
2. Use the `GetResponse` method of the `WebRequest` object to return a `WebResponse` object for the page. If the request times out, a `System.Net.WebException` will be thrown. You can configure the timeout for the network request through the `WebRequest.Timeout` property in milliseconds (the default value is 100000).
3. Create a `StreamReader` or a `BinaryReader` that wraps the stream returned by the `WebResponse.GetResponseStream` method.
4. Perform any steps you need to with the stream contents.

Conclusion: Here, we study the use of `WebRequest` and `WebResponse` class to download the file and process it using stream.

Experiment No. 14

Title: Design Simple ASP.NET web application.

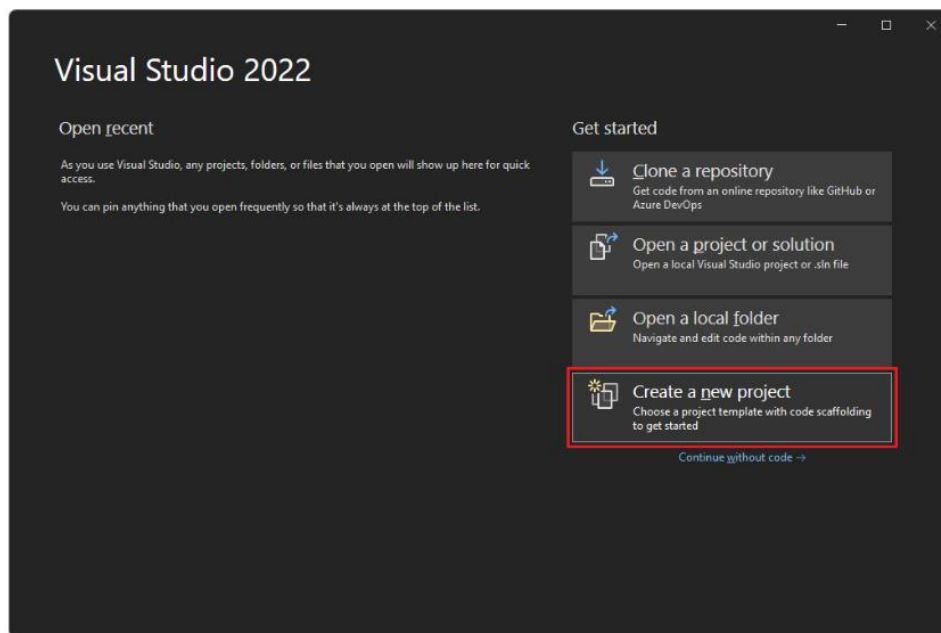
Aim: To design a simple ASP.NET web application.

Objective: 1) Create ASP.NET web application.

Theory:

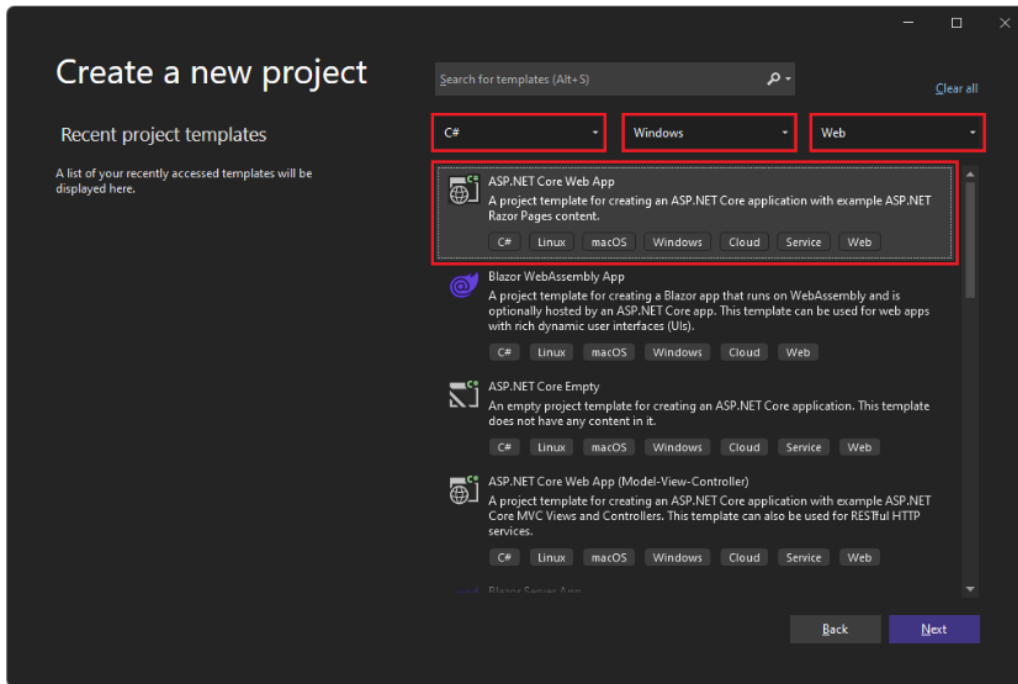
First, you'll create an ASP.NET Core project. The project type comes with all the template files you'll need to build a fully functional website.

1. On the start window, select **Create a new project**.

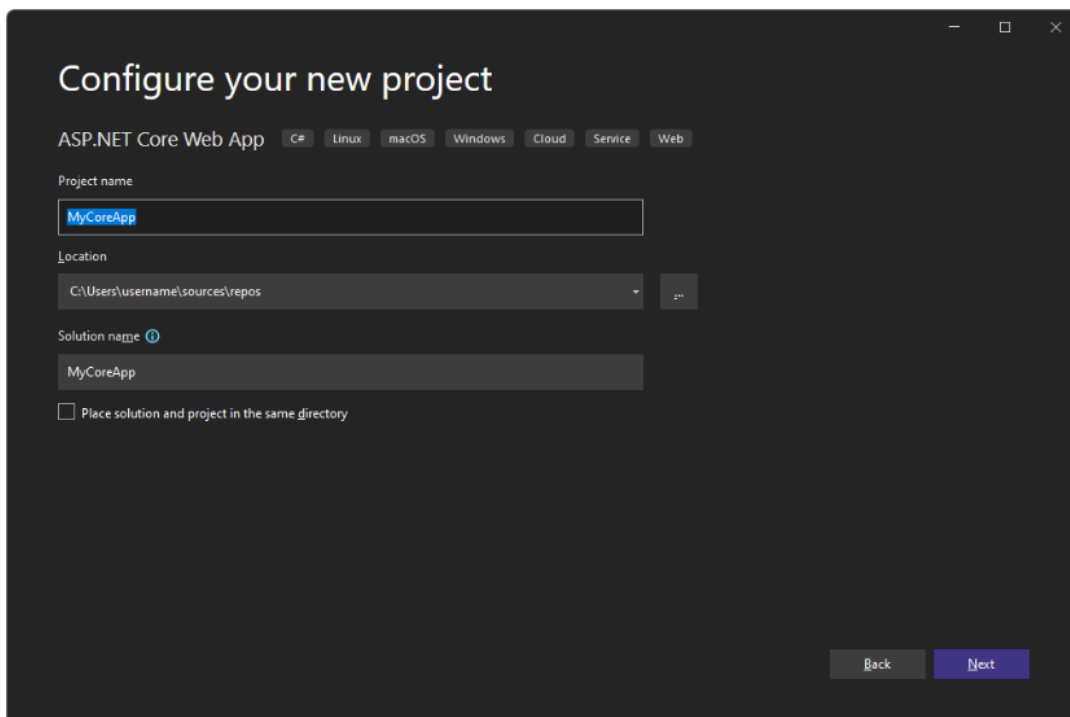


2. In the **Create a new project** window, select **C#** from the Language list. Next, select **Windows** from the platform list, and **Web** from the project types list.

After you apply the language, platform, and project type filters, select the **ASP.NET Core Web App** template, and then select **Next**.



3. In the **Configure your new project** window, enter **MyCoreApp** in the **Project name** field. Then, select **Next**.



4. In the **Additional information** window, verify that **.NET 6.0** appears in the **Target Framework** field. From this window, you can enable Docker support and add authentication support. The drop-down menu for **Authentication Type** has the following four options:

- **None.** No authentication.
- **Individual accounts.** These authentications are stored in a local or Azure-based database.
- **Microsoft identity platform.** This option uses Active Directory, Azure AD, or Microsoft 365 for authentication.
- **Windows.** Suitable for intranet applications.

Leave the **Enable Docker** box unchecked, and select **None** for Authentication Type. Next, select **Create**.

The screenshot shows a configuration window titled "Additional information" for an "ASP.NET Core Web App". At the top, there are tabs for "C#", "Linux", "macOS", "Windows", "Cloud", "Service", and "Web". The "Framework" dropdown is set to ".NET 6.0 (Long-term support)". The "Authentication type" dropdown is set to "None". There is a checked checkbox for "Configure for HTTPS" and an unchecked checkbox for "Enable Docker". The "Docker OS" dropdown is set to "Linux". At the bottom right, there are "Back" and "Create" buttons. Red rectangular boxes are drawn around the "Framework", "Authentication type", and "Enable Docker" options.

Conclusion: Student successfully create a web application.

Experiment No-15

Title:- : Design simple login and registration page using client-side validation controls in ASP.NET

Aim : Implement the application to login and registration page using client-side validation controls in ASP.NET

Objective :-

1. Study the controls in ASP.NET
2. Design & implement simple login and registration using client-side validation controls in ASP.NET

Theory :-

ASP.NET validation controls validate the user input data to ensure that useless, unauthenticated, or contradictory data don't get stored. ASP.NET validation controls provide functionality to perform validation using client script. By default, when client-side validation is being performed, the user cannot post the page to the server if there are errors on the page thus the user experience with the page is enhanced.

Client-side objects:

- Page_IsValid (Boolean variable) Indicates whether the page is currently valid. The validation scripts keep this up to date at all times.
- Page_Validators (Array of elements) This is an array containing all of the validators on the page.
- Page_ValidationActive
Boolean variable - Indicates whether validation should take place. Set this variable to False to turn off validation programmatically.
- IsValid
Boolean property - This is a property on each client validator indicating whether it is currently valid.
- Page_ValidationSummaries
Array of elements - This is an array containing all of the validation summaries on the page.

Client-Side APIs:

- ValidatorValidate(val) - Takes a client-validator as input. Makes the validator check its input and update its display.

- ValidatorEnable(val, enable) - Takes a client-validator and a Boolean value. Enables or disables a client validator. Being disabled will stop it from evaluating and it will always appear valid.
- ValidatorHookupControl(control, val) - Takes an input HTML element and a client-validator. Modifies or creates the element's change event so that it updates the validator when changed. This can be useful for custom validators that depend on multiple input values.

ASP.NET provides the following validation controls:

- RequiredFieldValidator
- RangeValidator
- CompareValidator
- RegularExpressionValidator
- CustomValidator
- ValidationSummary

Conclusion – we understand the client-side validation controls in ASP.NET and implement login and registration page using Client-side objects, API and validation controls.