.NET FRAMEWORK (18KP2CS06)

Unit- I: Preface: Introduction to Programming: Converting Source code to Machine Language Code- Explaining Program Development Cycle-Visual Basic: Getting Started with Visual Basic 2012- New Features-Keywords- Operator Precedence- Data types in Visual Basic 2012- Windows Forms: Introducing the Form Class- Performing Common Form Opertions- Creating Input boxes, Dialog boxes.

Unit -II: Windows Forms Controls –I: Using the Label ,Textbox, Button, RadioButton ,CheckBox, ComboBox, GroupBox- Windows Forms Controls-II: Using the ToolStrip, MenuStrip, StatusStrip controls- Working with Dialog Boxes- Windows Presentation Foundation: Exploring the Improvements in WPF 4.5- Describing types of WPF applications-Exploring WPF 4.5 Designer- Working with WPF Controls-Working with Resources and Style.

Text: ".NET 4.5 Programming Course Kit"- Vikas Gupta ,DreamsTech Press-Edition 2014

Subject Name: .NET Framework
Subject Code: 18KP2CS06
E-Content prepared Staff Name: J.Shanmugapriya, GL, Dept of Computer Science

UNIT I

Introduction to Programming

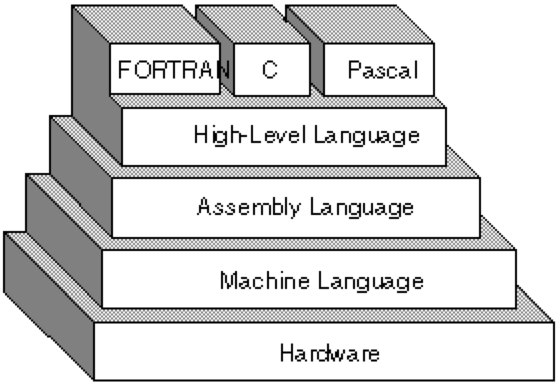
A computer is a calculating and computing device that is used to perform calculations and manipulations on various types of data, such as integers, strings, and float numbers. A computer performs all these tasks with the help of numbers of software, such as word processors, spreadsheets, and database applications. Software can be defined as a set of one or more programs, which are organized sets of certain statements and instructions that direct a computer to perform a specific task. These programs are converted into executable files with the help of a compiler or an interpreter.

A programming language is a formal language comprising a set of instructions that produce various kinds of output. Programming languages are used in computer programming to implement algorithms.

Each programming language has it own set of rules, known as syntax, which governs the structure of the statements in a program.

Programming languages can be broadly divided into the following two categories:

- High level language
- Low level language



High level language

A high-level language (HLL) is a programming language such as C, FORTRAN, or Pascal that enables a programmer to write programs that are more or less independent of a particular type of computer. Such languages are considered high-level because they are closer to human languages and further from machine languages. Programs written in a high-level language must be translated into machine language by a compiler or interpreter.

Program to add two numbers in BASIC

```
10 LET A =2
20 LET B=3
30 LET SUM = A+B
```

40 PRINT SUM
50 END

Low level language

A low-level language is a type of programming language that contains basic instructions recognized by a computer. Unlike high-level languages used by software developers, low-level code is often cryptic and not human-readable. Two common types of low-level programming languages are

- assembly language
- machine language.

Assembly language

Earlier than high level language were not invented, the programmers used to write programs in the Assembly language. similar to high level language, program written in assembly language is also not understandable by a computer. To make it understandable to a computer it needs to be e translated into machine language code with help of an assembler. A program written in the Assembly language consists of a set of instructions called mnemonics.

Machine language

The Machine language is only language computer can understand directly. A program written in machine language is a sequence of binary digits - 1s and 0s. the machine language is very difficult to understand and learn even for a advanced programmer.

Converting source code to machine language code

The programmers write programs by using a source language, which can be either a high-level language or an assembly language; however a computer cannot understand both these type of programming languages. Therefore, programs written in these types of languages need to be first converted into the machine language code. To convert the source code of a program into machine language code, a language converter is required.

HIGH LEVEL LANGUAGE TO MACHINE LANGUAGE CONVERSION

Conversion of code from high-level language to machine language is done with the help of the following two language converters:

Compiler:

Converts a complete program from a high-level language to machine language. If an error is caught by the compiler in the program during conversion, then the error is displayed to the user. On the other hand, if the compiler does not find any error in the program, then the compiler executes the program.

Interpreter :

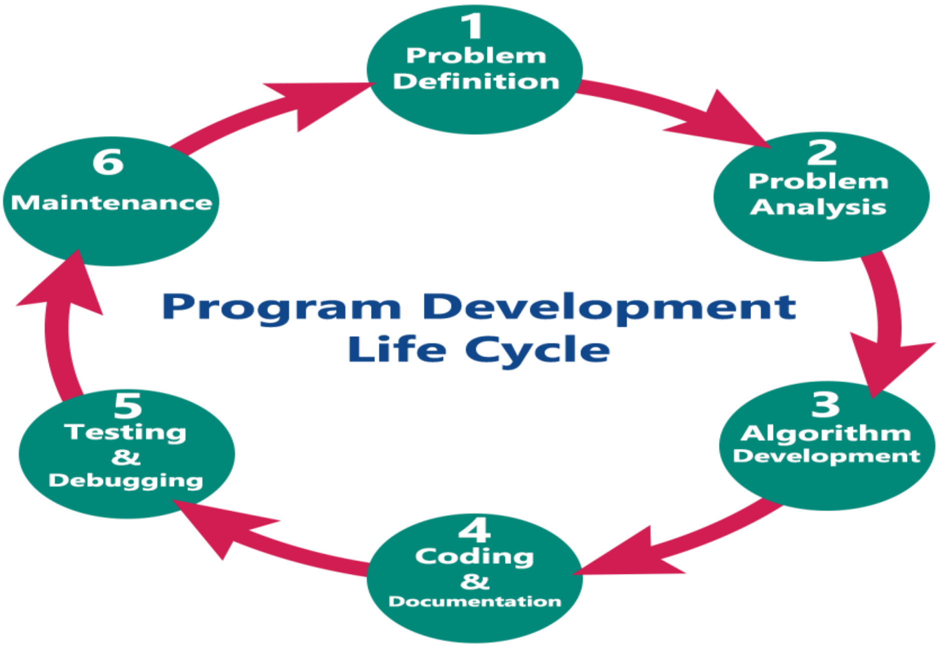
It converts the code written in a high-level language into machine language code, line by line. If the interpreter finds an error in the first line of code during conversion, it displays the error to the user. On the other hand, if the interpreter does not find any error in the first line of code, then the second line of the code is checked. The same process is followed for the remaining lines of the code as well.

Assembly language to Machine language conversion

The conversion of assembly language code into machine language code is done through another language converter, called assembler.

EXPLAINING PROGRAM DEVELOPMENT CYCLE

The program development cycle is a series of tasks are activities that take place during the development of a program.



The various stages of the program development cycle are as follows

1 .Analyze the problem

2. Developing a solution

3. coding the solution

4. testing the program

ANALYZING THE PROBLEM

Analyzing the problem involves determining what task the program needs to perform and what you need to do to enable that program to perform that task. Once we have clearly visualized the problem, we can move on to design the solution of the problem while analyzing a problem , we need to consider:

- The input supplied

- The process to obtain the and required output from the given input
- The output desired

In addition to determine the required output on the given input for your program, We also need to determine the variables and their types to represent input and output. A variable acts as a Placeholder parts during values used in a program.

Input

As explained one or more inputs are needed to be taken by a program to perform a task and return an output. For example, following inputs are required to display their registration information of the students of an educational Institute:

- Course code
- A character specifying whether or not more students need to be registered
  Let's assume that The Institute offers three courses with course code A, B and C. With this assumption come on the following variables are used to store the data used in this program:
- **course _code** for storing the value of course code
- **New_Reg**for Storing a character specifying whether or not more students need to be registered.
- **No_stud_A** for storing the number of students registered for course A
- **No_Stud_B** for storing the number of students registered for course B.
- **No_Stud_C** for storing the number of students registered for course C.

Process

To obtain the desired output from the given input, we need to process the data. the steps for processing the data in this program are as follows:

1. First, Check the course code for the first student.
2. If it is A, the entry will be made to the respective course, that is, course A. In this way, the course code for all the students is checked and the entries in the respective courses are made.
3. Finally, the total number of entries for the three courses are made.

Output

The output of the program includes:

- Total number of students registered for course A
- Total number of students registered for course B
- Total number of students registered for course C

DEVELOPING A SOLUTION

Once the problem has been analyzed, that is, the required output and the given input are determined, we an start developing a solution for the problem. The most important task in developing a solution is the development of logic that solves the problem. This requires creation of a step-by step procedure to solve the problem. This type of procedure is commonly termed as an algorithm. Once we created an algorithm to solve a problem, we can convert it into either a flowchart or pseudocode. A flowchart is a graphical representation of an algorithm. Pseudocode refers to short, readable, and formally-styled natural language code that is used to explain specific tasks within a program's algorithm.

Algorithm

An algorithm, as stated earlier, is a series of instruction written in any language spoken by human being, such as English. An algorithm describes a way to perform a particular programming task.

1. Initialize New_Reg to 'Y'
2. Initialize No_Stud_A to 0
3. Initialize No_Stud_B to 0
4. Initialize No_Stud_C to 0
5. Input Course_Code
6. Check the Course_Code. If it is A, then add 1 to No_Stud_A; otherwise, move to

   step 7.

7. If Course_Code is B, then add 1 to No_Stud_B; otherwise, move to step8.
8. If Course_Code is B, then add 1 to No_Stud_C;
9. Input New_Reg
10. If New_Reg is 'Y' , then repeat steps 5 to 9; otherwise, move to step 11.
11. Print the total number of students registered in each course, that is, No_Stud_A, No_Stud_B and No_Stud_C.

Pseudocode

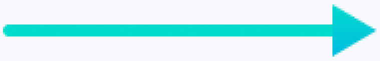
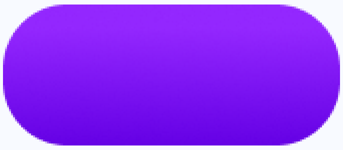
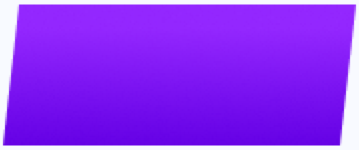
Pseudocode is structured set of English phrases that are used to describe the algorithms. It focuses on the logic of the algorithm and does not have any programming language specific keywords.

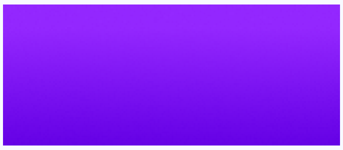
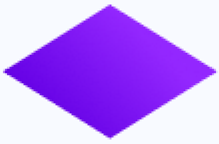
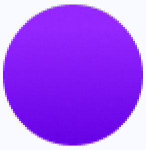
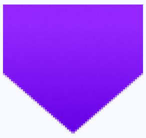
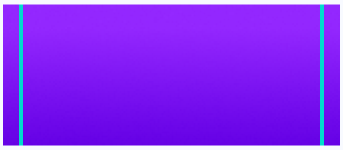
```
Initialize New_Reg to 'Y'

Initialize No_Stud_A to 0

Initialize No_Stud_B to 0

Initialize No_Stud_C to 0

While ( value of New_Reg is 'Y')

{

Course_code = get the course code from the user

    If Course_code= 'A' then

        No_Stud_A=  No_Stud_A + 1

    If Course_code= 'B' then
```

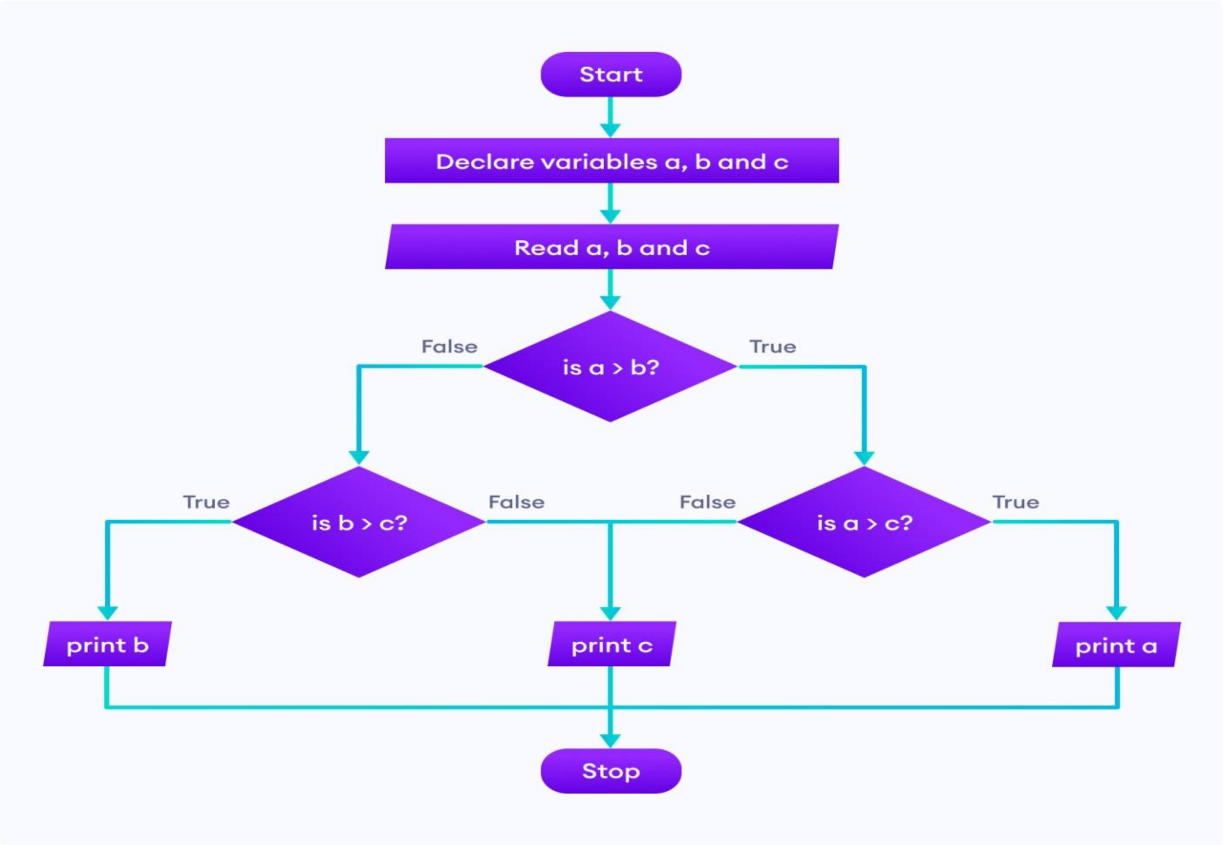
```
        No_Stud_B=  No_Stud_B + 1
    If Course_code= ‘C’ then
       No_Stud_C=  No_Stud_C + 1
New_Reg = Get the character from the user
}
Print No_Stud_A
Print No_Stud_B
Print No_Stud_C
```

Flowchart

A flowchart is a graphical representation of the various steps involved in an algorithm. It makes the flow of the program easy to understand. A flowchart uses different symbols to indicate different operations. The various symbols used in a flowchart are shown in fig:

Symbol	Purpose	Description
	Flow line	Indicates the flow of logic by connecting symbols.
	Terminal(Stop/Start)	Represents the start and the end of a flowchart.
	Input/Output	Used for input and output operation.

	Processing	Used for arithmetic operations and data-manipulations.
	Decision	Used for decision making between two or more alternatives.
	On-page Connector	Used to join different flowline
	Off-page Connector	Used to connect the flowchart portion on a different page.
	Predefined Process/Function	Represents a group of statements performing one processing task.



CODING THE SOLUTION

The third stage is coding the solution or writing the program, based on the flowchart and pseudocode. Software used for programming may consists of one or more of the following tools.

Code Editor:

A code editor is optimized for programming. For example, in a code editor, exeutabel statements might appear in one color and comments in another color to make the program easy to understand.

Compiler (or interpreter):

Converts the code in each source code file into machine language code and alces it in a file, known as object file.

Linker:

Combines all the object files into an executable program that an run directly on the target computer.

Debugger:

Helps us in finding errors in the programs.

Writing and executing a program is accomplished by performing the following steps:

1. Type the program
2. Compile the program
3. If there are any compile-time errors, correct them and again compile the program.
4. Run the program to obtain the desired results.
5. If there are any run- time errors, check the logic of the program and continue form step 2.

Compilers can only detect syntax error. The errors that are detected at run-time are known as run-time errors. It is important to know the difference between compile-time and run-time errors.

- compile-time error: occurs if there is a mistake in the syntax of the program. A program will not compile successfully if there are syntax errors.
- Run-time error: Occurs if there is a mistake in the logic of the program. A program that was compiled successfully may have run-time errors. Run- time errors occurs when we run the program.

TESTING THE PROGRAM

Testing is the last stage in the program development cycle. Once we have developed a program, we should test it to ensure that it is free of bug and is capable of solving the given problem.

EXPLORING VISUAL STUDIO 2012 IDE

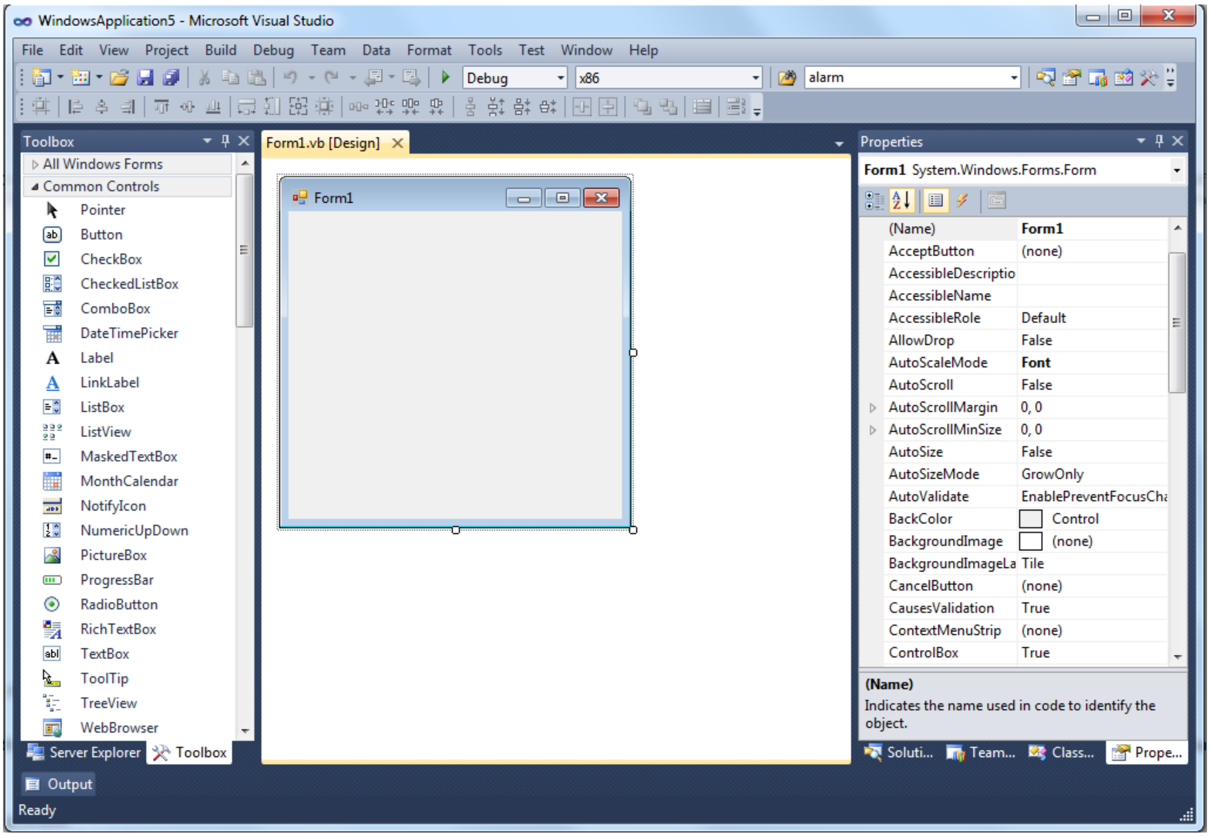
Visual studio 2012 IDE is a comprehensive environment for the development and execution of .NET applications. It consists of menu bar, toolbar, and several windows that assist to design the UI of .NET applications as well as execute the applications.

Some of the important components of Visual Studio 2012 IDE are as follows:

- Start page
- Menu bar and Toolbar
- Toolbox
- Solution Explorer
- Properties window
- Designer and code editor
- Server Explorer
- Output window
- Object Browser
- Class View window

Start Page

As the name suggests, Start Page is the first page that appears whenever we open Visual Studio 2012.



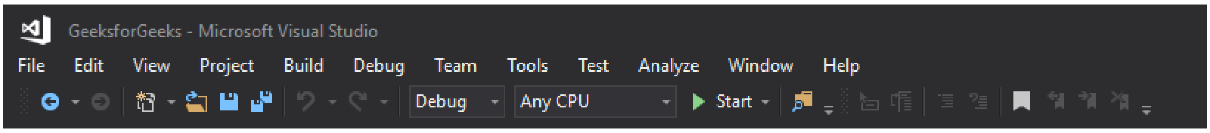
Form1 System.Windows.Forms.Form

(Name)	Form1
AcceptButton	(none)
AccessibleDescriptio	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
AutoScaleMode	Font
AutoScroll	False
AutoScrollMargin	0, 0
AutoScrollMinSize	0, 0
AutoSize	False
AutoSizeMode	GrowOnly
AutoValidate	EnablePreventFocusCha
BackColor	Control
BackgroundImage	(none)
BackgroundImageLa	Tile
CancelButton	(none)
CausesValidation	True
ContextMenuStrip	(none)
ControlBox	True

(Name)
Indicates the name used in code to identify the object.

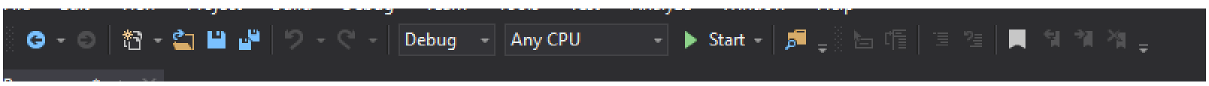
Menu Bar and Tool Bar:

Various Menus in Visual Studio: A user can find a lot of menus on the top screen of Visual Studio as shown below


GeeksforGeeks - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug
Any CPU
Start

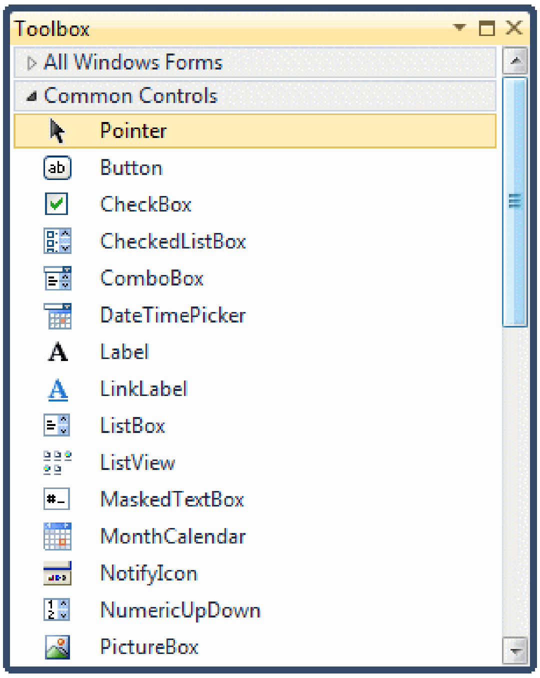
1. Create, Open and save projects commands are contained by **File** menu.
2. Searching, Modifying, Refactoring code commands are contained by the **Edit** menu.
3. **View** Menu is used to open the additional tool windows in Visual Studio.
4. **Project** menu is used to add some files and dependencies in the project.
5. To change the settings, add functionality to Visual Studio via extensions, and access various Visual Studio tools can be used by using **Tools** menu.

- The below menu is known as the **toolbar** which provide the quick access to the most frequently used commands. You can add and remove the commands by going to **View** → **Customize**



Toolbox

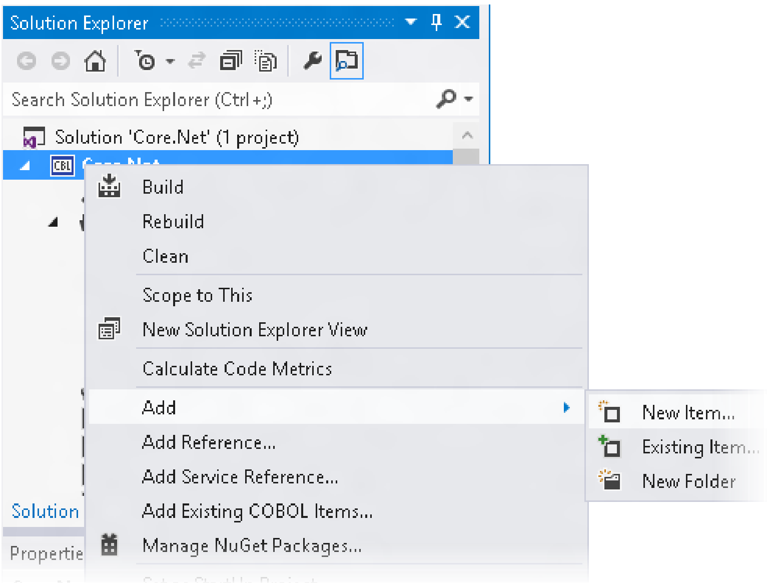
Toolbox is a window that contains icons for various items and controls that we can add to a .NET application to design the UI of an application. The icons in Toolbox are logically grouped under different tabs, such as Stand, Containers, and Menus & Toolbars



Solution Explorer:

Visual Studio provides a Solution Explorer window that enables you to explore and manage your solutions and projects. To open the window select **View** > **Solution Explorer**.

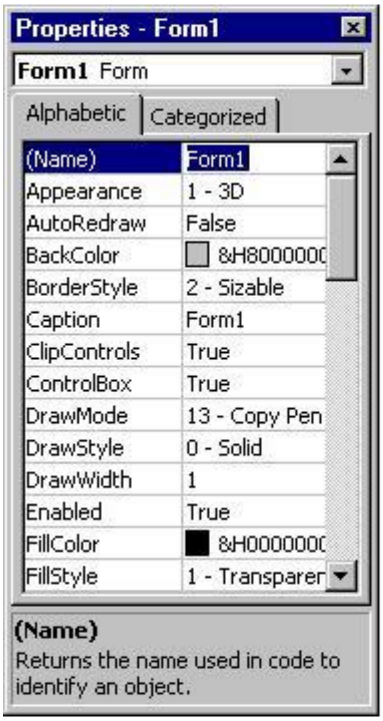
Solution Explorer displays the projects that form your solution, the files and folders in a project as they appear on the physical hard drive, and any assemblies, COM objects or files the project references. The context menus within Solution Explorer provide a variety of commands that help you manage your projects.



Properties Window

Use this window to view and change the design-time properties and events of selected objects that are located in editors and designers. You can also use the **Properties** window to edit and view file, project, and solution properties. You can find **Properties** Window on the **View** menu. You can also open it by pressing **F4** or by typing **Properties** in the search box.

The **Properties** window displays different types of editing fields, depending on the needs of a particular property. These edit fields include edit boxes, drop-down lists, and links to custom editor dialog boxes. Properties shown in gray are read-only.



Properties - Form1

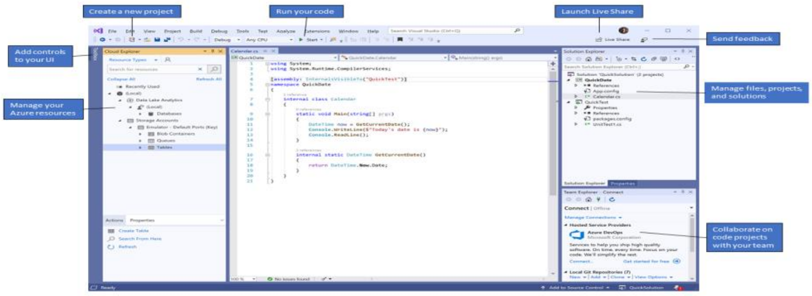
Form1 Form

(Name)	Form1
Appearance	1 - 3D
AutoRedraw	False
BackColor	☐ &H8000000(
BorderStyle	2 - Sizable
Caption	Form1
ClipControls	True
ControlBox	True
DrawMode	13 - Copy Pen
DrawStyle	0 - Solid
DrawWidth	1
Enabled	True
FillColor	■ &H0000000(
FillStyle	1 - Transparer

(Name)
Returns the name used in code to identify an object.

Designer and Code Editor

The designer helps in designing the UI of application, while the code editor helps in adding the code or functionality for the application. The available designers and code editors in Visual Studio 2012 depend on the type of application and the file that we are working with.

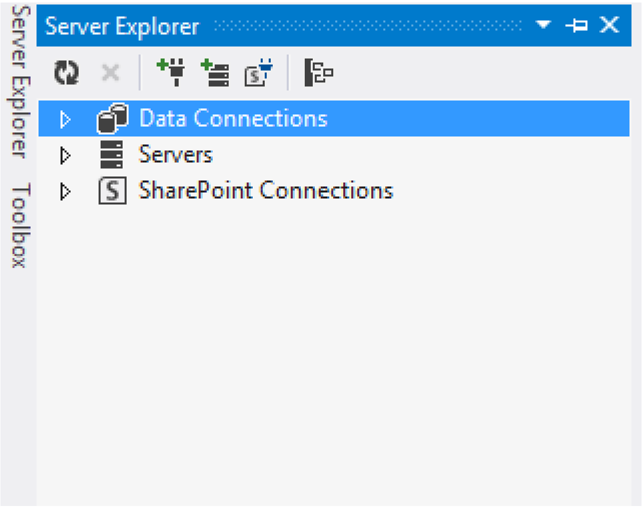


Create a new project
Run your code
Launch Live Share
Send feedback
Add controls to your UI
Manage your Azure resources
Manage files, projects, and solutions
Collaborate on code projects with your team

```
using System;
using System.Runtime.CompilerServices;

[assembly: InternalsVisibleTo("QuickTest")]
namespace QuickDate
{
    internal class Calendar
    {
        static void Main(string[] args)
        {
            DateTime now = GetCurrentDate();
            Console.WriteLine($"Today's date is {now}");
            Console.ReadLine();
        }

        internal static DateTime GetCurrentDate()
        {
            return DateTime.Now.Date;
        }
    }
}
```

Server Explorer

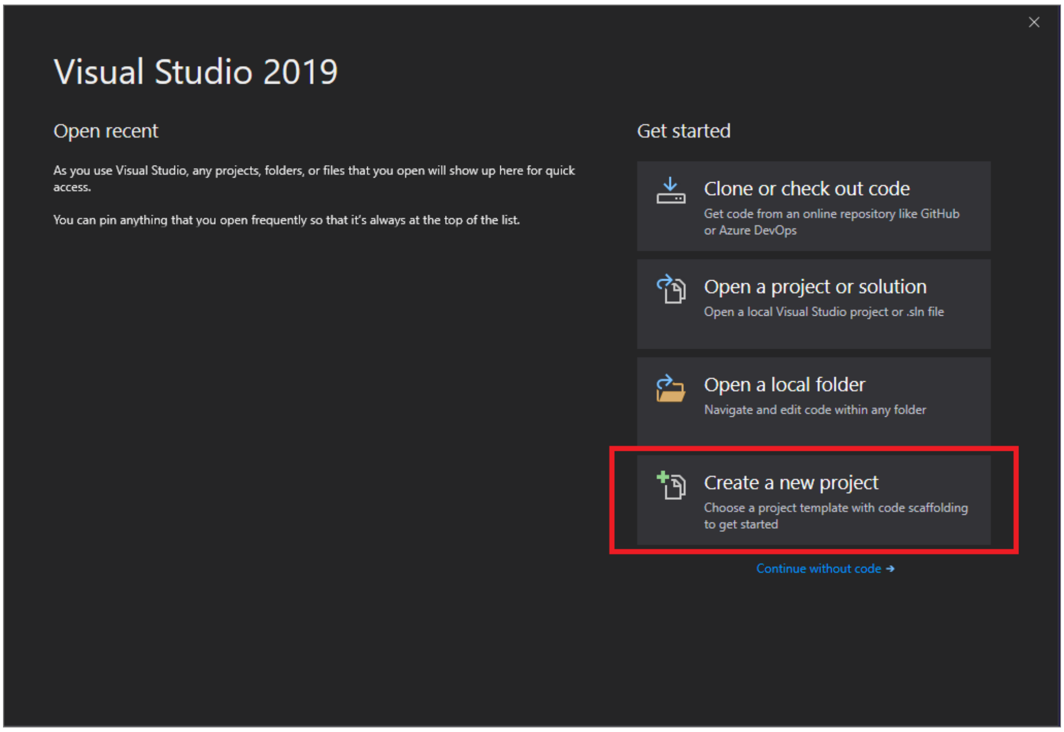
As said, the Server Explorer provides quick access to the connected database servers. With the Server Explorer, you can set up queries for use in your program. A default instance of the Server Explorer looks like the following:



- Data Connections
- Servers
- SharePoint Connections

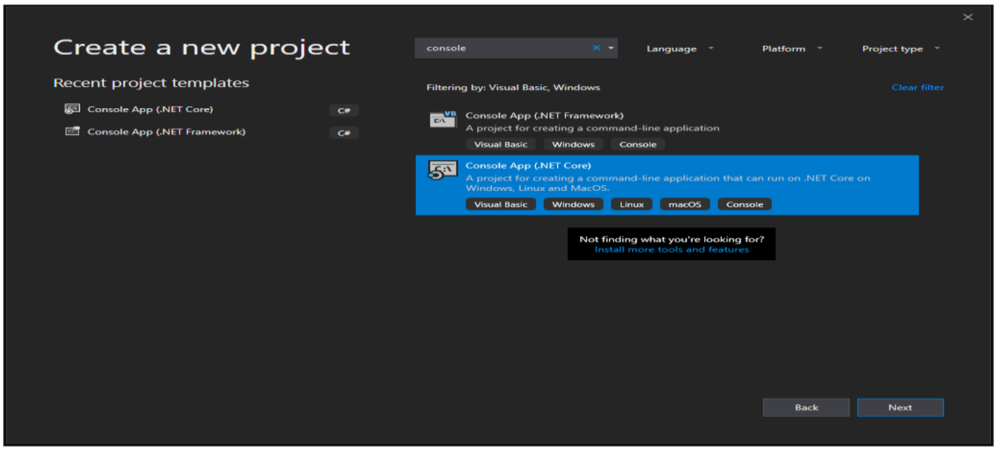
Developing a Console Applicaion

1. Open Visual Studio 2012.
2. On the start window, choose **Create a new project**.


Visual Studio 2019
Open recent
As you use Visual Studio, any projects, folders, or files that you open will show up here for quick access.
You can pin anything that you open frequently so that it's always at the top of the list.
Get started
Clone or check out code
Get code from an online repository like GitHub or Azure DevOps
Open a project or solution
Open a local Visual Studio project or .sln file
Open a local folder
Navigate and edit code within any folder
Create a new project
Choose a project template with code scaffolding to get started
Continue without code →

3. On the **Create a new project** window, enter or type *console* in the search box. Next, choose **Visual Basic** from the Language list, and then choose **Windows** from the Platform list.

   After you apply the language and platform filters, choose the **Console App (.NET Core)** template, and then choose **Next**.



Create a new project

Recent project templates

Filtering by: Visual Basic, Windows

Console App (.NET Framework)
A project for creating a command-line application

Console App (.NET Core)
A project for creating a command-line application that can run on .NET Core on Windows, Linux and MacOS.

Not finding what you're looking for?
Install more tools and features

Then, in the Visual Studio Installer, choose the **.NET Core cross-platform development** workload.

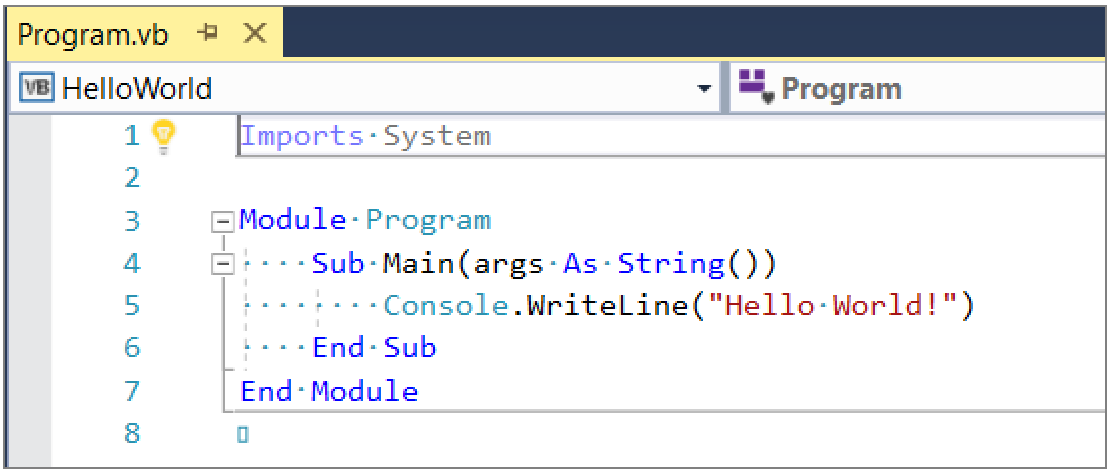
After that, choose the **Modify** button in the Visual Studio Installer. You might be prompted to save your work; if so, do so. Next, choose **Continue** to install the workload. Then, return to step 2 in this "**Create a project**" procedure.

4. In the **Configure your new project** window, type or enter *WhatIsYourName* in the **Project name** box. Then, choose **Create**.

   Visual Studio opens your new project.

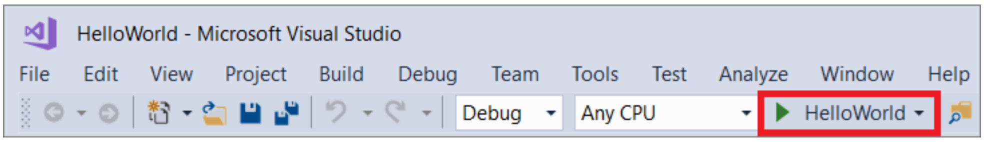
Create the application

After you select your Visual Basic project template and name your project, Visual Studio creates a simple "Hello World" application for you. It calls the WriteLine method to display the literal string "Hello World!" in the console window.



```vb
Imports System

Module Program
    Sub Main(args As String())
        Console.WriteLine("Hello World!")
    End Sub
End Module

```

If you click the **HelloWorld** button in the IDE, you can run the program in Debug mode.



HelloWorld - Microsoft Visual Studio

When you do this, the console window is visible for only a moment before it closes. This happens because the Main method terminates after its single statement executes, and so the application ends.

Add some code

Let's add some code to pause the application and then ask for user input.

1. Add the following code immediately after the call to the WriteLine method:

   ```
   Console.Write("Press any key to continue...")
   Console.ReadKey(true)
   ```

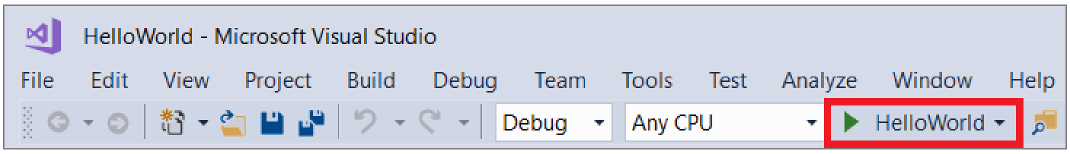
   This pauses the program until you press a key.

2. On the menu bar, select **Build** > **Build Solution**.

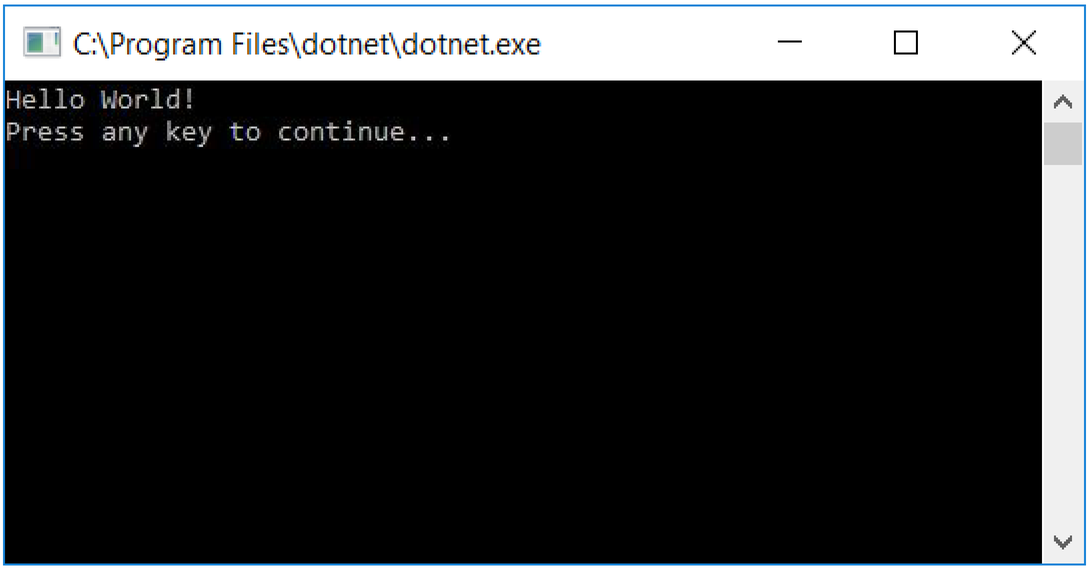
   This compiles your program into an intermediate language (IL) that's converted into binary code by a just-in-time (JIT) compiler.

Run the application

1. Click the **HelloWorld** button on the toolbar.



2. Press any key to close the console window.



C:\Program Files\dotnet\dotnet.exe

```
Hello World!
Press any key to continue...
```

Chapter 1- New features of 2012

Async and Await:

- Async feature allows developers to write asynchronous code.
- Async method is used to execute a time consuming thread without forcing the callers thread to wait.
- It helps them write the asynchronous code as easily as they write the synchronous code.
- It is useful institution where user interface is not responding or the server is not scalable.
- Await expression to suspend the execution of a thread until the awaited thread completes its task.

Iterators:

- An iterator returns an element of a collection by using the yield statement.
- Whenever yield statement is encounter the location in the code is retained.
- Next time, the execution is started from this location the iteration function is called by using the For Each...Next statement.

Call hierarchy

- The call hierarchy feature allows a developer to view all the calls made to and from a particular method or property or constructor.
- This feature provides better understanding of how the code flows and Alice evaluation of the effects of modifications to code.

Caller information

- The caller information feature allows a developer to easily retrieve the information about the caller of a method by using the caller info attributes.
- This information is useful for tracing debugging and creating diagnostic tools.

Visual basic 2012 keywords

- Every programming language uses a set of predefined words in coding.
- keyword has a specific predefined meaning for the compiler.
- Visual basic 2012 provides two types of keywords they are reserve and unreserved .
- Reserved keywords for those keywords that cannot be used as names for programming elements such as variables methods and classes, while and reserved keywords for those keywords that can be bused as names for programming elements.

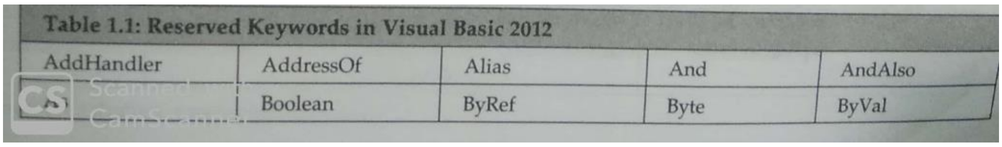


Table 1.1: Reserved Keywords in Visual Basic 2012

AddHandler	AddressOf	Alias	And	AndAlso
[illegible]	Boolean	ByRef	Byte	ByVal

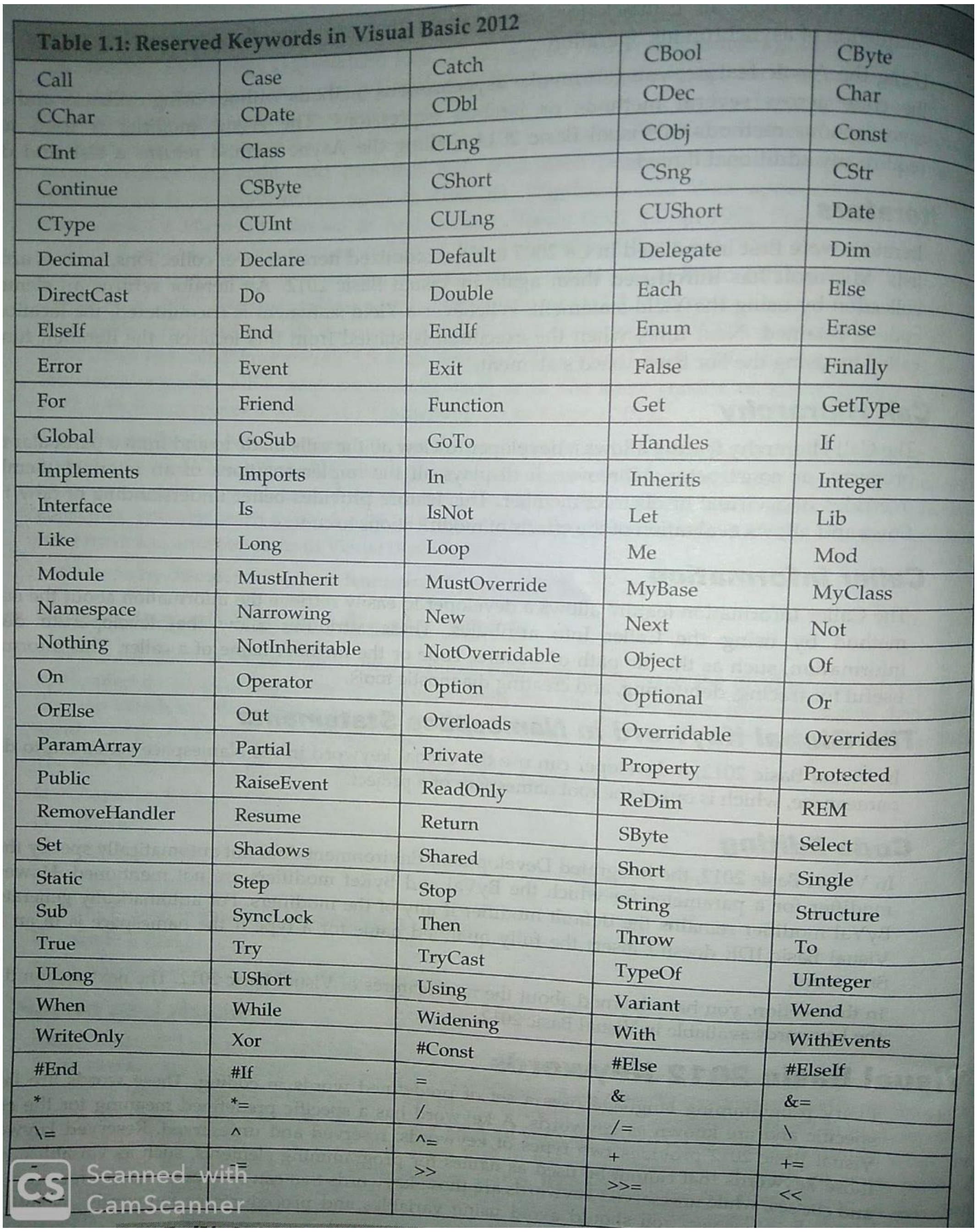


Table 1.1: Reserved Keywords in Visual Basic 2012

Call	Case	Catch	CBool	CByte
CChar	CDate	CDbl	CDec	Char
CInt	Class	CLng	CObj	Const
Continue	CSByte	CShort	CSng	CStr
CType	CUInt	CULng	CUShort	Date
Decimal	Declare	Default	Delegate	Dim
DirectCast	Do	Double	Each	Else
ElseIf	End	EndIf	Enum	Erase
Error	Event	Exit	False	Finally
For	Friend	Function	Get	GetType
Global	GoSub	GoTo	Handles	If
Implements	Imports	In	Inherits	Integer
Interface	Is	IsNot	Let	Lib
Like	Long	Loop	Me	Mod
Module	MustInherit	MustOverride	MyBase	MyClass
Namespace	Narrowing	New	Next	Not
Nothing	NotInheritable	NotOverridable	Object	Of
On	Operator	Option	Optional	Or
OrElse	Out	Overloads	Overridable	Overrides
ParamArray	Partial	Private	Property	Protected
Public	RaiseEvent	ReadOnly	ReDim	REM
RemoveHandler	Resume	Return	SByte	Select
Set	Shadows	Shared	Short	Single
Static	Step	Stop	String	Structure
Sub	SyncLock	Then	Throw	To
True	Try	TryCast	TypeOf	UInteger
ULong	UShort	Using	Variant	Wend
When	While	Widening	With	WithEvents
WriteOnly	Xor	#Const	#Else	#ElseIf
#End	#If	=	&	&=
*	*=	/	/=	\
\=	^	^=	+	+=
-	-=	>>	>>=	<<
[illegible]				

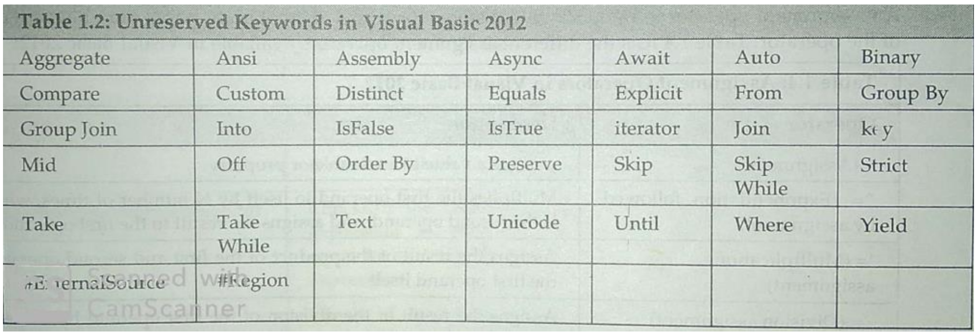


Table 1.2: Unreserved Keywords in Visual Basic 2012

Aggregate	Ansi	Assembly	Async	Await	Auto	Binary
Compare	Custom	Distinct	Equals	Explicit	From	Group By
Group Join	Into	IsFalse	IsTrue	iterator	Join	key
Mid	Off	Order By	Preserve	Skip	Skip While	Strict
Take	Take While	Text	Unicode	Until	Where	Yield
#ExternalSource	#Region					

Operators

An operator is a symbol that is used to perform an operation on one or more expressions called operands.

- arithmetic operators
- assignment operators
- comparison operators
- concatenation operators
- logical and bitwise operators
- miscellaneous operators

Arithmetic operators

The operators used to perform arithmetic operations such as subtraction multiplication and division are called arithmetic operators.

Operators	Meaning	Example
^	Raises one operand to the power of another	x ^ y (x to the power y)
+	Adds two operands	x + y
–	Subtracts second operand from the first	x – y
*	Multiplies both operands	x * y
/	Divides one operand by another and returns a floating-point result	x / y
\	Divides one operand by another and returns an integer result	x \ y
MOD	Modulus Operator and the remainder of a result after an integer division	x MOD y (remainder of x/y)

Assignment Operators

Assignment operators are used for assigning values to variables in VB.NET.

Operators	Example	Equivalent to
=	x = 4	x = 4
+=	x += 4	x = x + 4
-=	x -= 4	x = x – 4
*=	x *= 4	x = x * 4
/=	x /= 4	x = x / 4
\=	x \= 4	x = x \ 4
^=	x ^= 4	x = x ^ 4
<<=	x << = 4	x = x << 4
>>=	x >> = 4	x = x >> 4
&=	x &= 4	x = x & 4

Comparison Operators

Comparison operators are basically used to compare different values. These operators normally return Boolean values either True or False depending upon the condition.

Operators	Meaning	Example
=	Equality Check -Returns True if both values are the same	x == y
<>	Non-Equality Returns True if both values are unequal	x < > y
>	Greater than Check-Returns true if the first value specified is greater than the second	x > y
<	Less than-Returns true if the first value specified is less than second	x < y x
>=	Checks for two conditions, If the first value is greater than or equal to the second value it returns true	>= y
<=	Checks for two conditions, If the first value is less than or equal to the second value it returns true	x <= y
Is	Compares two Object Variable for Reference, True If the same object reference	
IsNot	Compares two Object Variable for Reference, False If the same object reference	
Like	compares a string against a pattern.	

Concatenation Operators

The process of combining two text strings into one string is called string concatenation and operators used to perform string concatenation are called concatenation operators.

Logical and Bitwise Operators

Logcal operators can be defined as operators that are used with expressions and produce a Boolean value.

Operators	Meaning	Example
And	Logical as well as bitwise AND operator. Returns True If both the operands are true	x And y
	Does not perform short-circuiting, i.e., it evaluates both the expressions	
Or	Logical as well as bitwise OR operator. Returns True If any of the two operands is true. It does not perform short-circuiting.	x Or y
Not	Logical as well as bitwise NOT operator. If True, then this operator will make it false.	Not y
Xor	Logical as well as bitwise Logical Exclusive OR operator. Returns True if both expressions are the same; otherwise False.	x Xor y
AndAlso	Logical AND operator. Works only on Boolean data. Performs short-circuiting.	x AndAlso y
OrElse	Logical OR operator. Works only on Boolean data. Performs short-circuiting.	x OrElse y
IsFalse	Determines whether an expression is False	
IsTrue	Determines whether an expression is False	

Miscellaneous Operators

Operators	Example	Equivalent to

AddressOf	Returns the address of a procedure.	AddHandler Button1.Click, AddressOf Button1_Click
Await	It is applied to an operand in an asynchronous method or lambda expression to suspend execution of the method until the awaited task completes.	Dim result As res = Await AsyncMethodThatReturnsResult() Await AsyncMethod()
GetType	It returns a Type object for the specified type.	MsgBox(GetType(Integer).ToString())
Function Expression	It declares the parameters and code that define a function lambda expression.	Dim add5 = Function(num As Integer) num + 5 ‘prints 10 Console.WriteLine(add5(5))
If	It uses short-circuit evaluation to conditionally return one of two values.	Dim num = 5 Console.WriteLine(If(num >= 0, “Positive”, “Negative”))

Operator Precedence

- When several operations occur in an expression, each part is evaluated and resolved in a predetermined order called *operator precedence*. The arithmetic and concatenation operators have an order of precedence that is described below, and all have higher precedence than the comparison and logical operators.
- Comparison operators have higher precedence than the logical operators, but lower precedence than the arithmetic and concatenation operators. All comparison operators have equal precedence; that is, they are evaluated in the order, left to right, in which they appear. Arithmetic, concatenation and logical/bitwise Operators are evaluated in the following order of precedence:

Arithmetic/Concatenation Operators

- Exponentiation (^)
- Negation (–)
- Multiplication and division (*****, /)
- Integer division (\)

Modulus arithmetic (**Mod**)
Addition and subtraction (+, –), String concatenation (+)
String concatenation (**&**)

Comparison Operators

- Equality (=)
- Inequality (<>)
- Less than, greater than (<,>)
- Greater than or equal to (>=)
- Less than or equal to (<=)
- Like
- Is
- TypeOf...Is

Logical/Bitwise Operators

- Negation (**Not**)
- Conjunction (**And, AndAlso**)
- Disjunction (**Or, OrElse, Xor**)

Data types in Visual Basic 2012

Data types refer to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.

Data Types Available in VB.Net

VB.Net provides a wide range of data types. The following table shows all the data types available −

Data Type	Storage Allocation	Value Range
Boolean	Depends on implementing platform	True or False
Byte	1 byte	0 through 255 (unsigned)
Char	2 bytes	0 through 65535 (unsigned)
Date	8 bytes	0:00:00 (midnight) on January 1, 0001 through 11:59:59 PM on December 31, 9999

Decimal	16 bytes	0 through +/-79,228,162,514,264,337,593,543,950,335 (+/-7.9...E+28) with no decimal point; 0 through +/-7.9228162514264337593543950335 with 28 places to the right of the decimal
Double	8 bytes	-1.79769313486231570E+308 through -4.94065645841246544E-324, for negative values 4.94065645841246544E-324 through 1.79769313486231570E+308, for positive values
Integer	4 bytes	-2,147,483,648 through 2,147,483,647 (signed)
Long	8 bytes	-9,223,372,036,854,775,808 through 9,223,372,036,854,775,807(signed)
Object	4 bytes on 32-bit platform 8 bytes on 64-bit platform	Any type can be stored in a variable of type Object
SByte	1 byte	-128 through 127 (signed)
Short	2 bytes	-32,768 through 32,767 (signed)
Single	4 bytes	-3.4028235E+38 through -1.401298E-45 for negative values; 1.401298E-45 through 3.4028235E+38 for positive values
String	Depends on implementing platform	0 to approximately 2 billion Unicode characters
UInteger	4 bytes	0 through 4,294,967,295 (unsigned)
ULong	8 bytes	0 through 18,446,744,073,709,551,615 (unsigned)
User-Defined	Depends on implementing platform	Each member of the structure has a range determined by its data type and independent of the ranges of the other members

UShort	2 bytes	0 through 65,535 (unsigned)

The Type Conversion Functions in VB.Net

Sr.No.	Functions & Description
1	CBool(expression) Converts the expression to Boolean data type.
2	CByte(expression) Converts the expression to Byte data type.
3	CChar(expression) Converts the expression to Char data type.
4	CDate(expression) Converts the expression to Date data type
5	CDbl(expression) Converts the expression to Double data type.
6	CDec(expression) Converts the expression to Decimal data type.
7	CInt(expression) Converts the expression to Integer data type.
8	CLng(expression) Converts the expression to Long data type.
9	CObj(expression) Converts the expression to Object type.
10	CSByte(expression) Converts the expression to SByte data type.
11	CShort(expression) Converts the expression to Short data type.

12	CSng(expression) Converts the expression to Single data type.
13	CStr(expression) Converts the expression to String data type.
14	CUInt(expression) Converts the expression to UInt data type.
15	CULng(expression) Converts the expression to ULng data type.
16	CUShort(expression) Converts the expression to UShort data type.

Visual Basic Statements

A statement is a complete instruction in Visual Basic programs. It may contain keywords, operators, variables, literal values, constants and expressions.

Statements could be categorized as −

- Declaration statements − these are the statements where you name a variable, constant, or procedure, and can also specify a data type.
- Executable statements − these are the statements, which initiate actions. These statements can call a method or function, loop or branch through blocks of code or assign values or expression to a variable or constant. In the last case, it is called an Assignment statement.

Using the If...Else Statement:

Syntax

```
If boolean_expression Then
// Statements to Execute if boolean expression is True
Else
// Statements to Execute if boolean expression is False
End If
```

The statements inside of **If** condition will be executed only when the “**bool_expression**” returns **true** otherwise the statements inside of **Else** condition will be executed.
Example:

```
Dim x As Integer = 20
If x >= 10 Then
   Console.WriteLine("x is Greater than or equals 10")
```

```
Else
    Console.WriteLine("x is less than or equals to 10")
End If
```

Using the Select...Case Statement:

- **Select...Case** statement is useful to execute a single case statement from the group of multiple case statements based on the value of a defined expression.

- By using **Select...Case** statement in Visual Basic, we can replace the functionality of if…else if statement to provide better readability for the code.
- Generally, the Select...Case statement is a collection of multiple case statements and it will execute only one single case statement based on the matching value of the defined expression.

Syntax

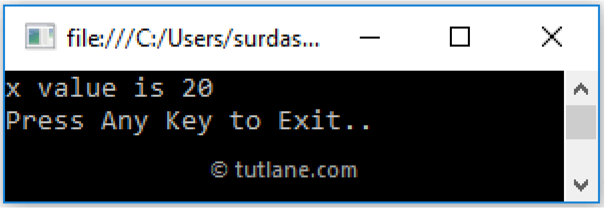
```
Select Case variable/expresison
Case value1
// Statements to Execute
Case value2
//Statements to Execute
....
....
Case Else
// Statements to Execute if No Case Matches
End Select
```

Here, the **Select** statement will evaluate the **expression** / **variable** value by matching with **Case** statement values (value1, value2, etc.). If **variable/expression** value matches with any of the **case** statement, then the statements inside of the particular **case** will be executed.

In case, if none of the **case** statements are matches with the defined **expression** / **variable** value, then the statements inside of **Else** block will be executed and it's more like **Else** block in if...else statement.

Example

```
Module Module1
    Sub Main()
        Dim x As Integer = 20
        Select Case x
            Case 10
                Console.WriteLine("x value is 10")
            Case 15
                Console.WriteLine("x value is 15")
            Case 20
                Console.WriteLine("x value is 20")
            Case Else
                Console.WriteLine("Not Known")
        End Select
        Console.WriteLine("Press Enter Key to Exit..")
        Console.ReadLine()
    End Sub
End Module
```



```
x value is 20
Press Any Key to Exit..
```

Using the For...Next Statement:

- **For** loop is useful to execute a statement or a group of statements repeatedly until the defined condition returns true.
- Generally, For loop is useful iterate and execute a certain block of statements repeatedly until the specified number of times.

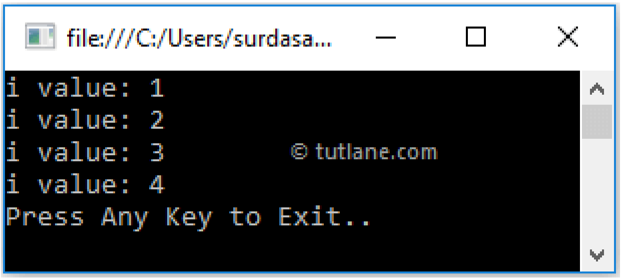
Syntax

```
For variable As [Data Type] = start To end

// Statements to Execute

Next
```

Here, the variable parameter is required in the For statement and it must be numeric. The **Data Type** is optional and it is useful to define the data type for the variable. The **start** and **end** parameters are required to define the initial and final value of a variable.

Example

```
Module Module1
    Sub Main()
        For i As Integer = 1 To 4
            Console.WriteLine("i value: {0}", i)
        Next
        Console.WriteLine("Press Enter Key to Exit..")
        Console.ReadLine()
    End Sub

End Module
```



```
i value: 1
i value: 2
i value: 3
i value: 4
Press Any Key to Exit..
```

Using For Each...Next Statement:

Repeats a group of statements for each element in a collection.

Syntax

```
For Each element [ As datatype ] In group
    [ statements ]
    [ Continue For ]
    [ statements ]
    [ Exit For ]
    [ statements ]
Next [ element ]
```

Using the While… End While statement

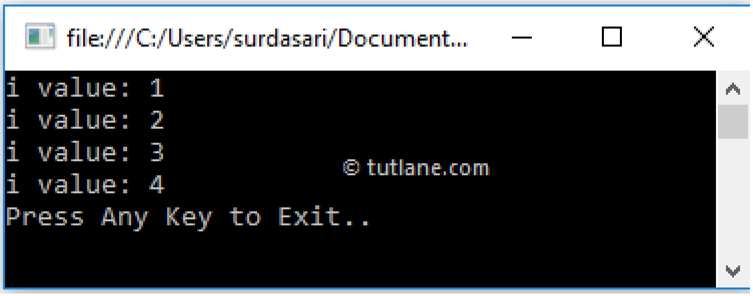
While loop is useful to execute the block of statements as long as the specified condition is **true.**

Syntax:

```
While boolean_expression
// Statements to Execute
End While
```

Example

```
Module Module1
    Sub Main()
        Dim i As Integer = 1
        While i <= 4
            Console.WriteLine("i value: {0}", i)
            i += 1
        End While
        Console.WriteLine("Press Enter Key to Exit..")
        Console.ReadLine()
    End Sub

End Module
```



```
i value: 1
i value: 2
i value: 3
i value: 4
Press Any Key to Exit..
```

Using Do...Loop Statement

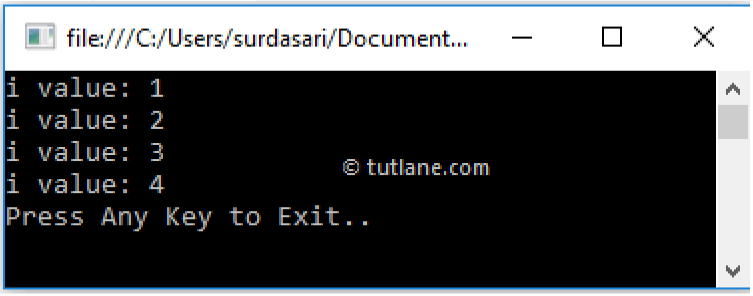
Generally, in Visual Basic the do-while loop is same as while loop but only the difference is while loop will execute the statements only when the defined condition returns true but the do-while loop will execute the statements at least once because first it will execute the block of statements and then it will checks the condition.

Syntax

```
Do
// Statements to Execute
Loop While boolean_expression
```

Example

```
Module Module1
   Sub Main()
      Dim i As Integer = 1
      Do
         Console.WriteLine("i value: {0}", i)
         i += 1
      Loop While i <= 4
      Console.WriteLine("Press Enter Key to Exit..")
      Console.ReadLine()
   End Sub
End Module
```



```
i value: 1
i value: 2
i value: 3
i value: 4
Press Any Key to Exit..
```

Variables

A variables is an identifier that denotes a storage area in the memory.

Syntax

```
Dim [Variable Name] As [Data Type]

Dim [Variable Name] As [Data Type] = [Value]
```

The variable name is to tell the compiler about the type of data the variable can hold.

Item	Description
Dim	It is useful to declare and allocate the storage space for one or more variables.
[Variable Name]	It’s the name of the variable to hold the values in our application.
As	The As clause in the declaration statement allows you to define the data type.
[Data Type]	t’s a type of data the variable can hold such as integer, string, decimal, etc.
[Value]	Assigning a required value to the variable.

Constants

- Generally, in visual basic the constant field values are set at compile-time and those values will never be changed.In visual basic, Const keyword is to declare the constant field, then that field value cannot be changed throughout the application. It’s mandatory to initialize constant fields with required values during the declaration itself otherwise compile-time errors in visual basic application.

Syntax

```
field_name As data_type = "value"
```

Arrays

- In visual basic, **Arrays** are useful to store multiple elements of the same data type at contiguous memory locations and arrays will allow us to store the fixed number of elements sequentially based on the predefined number of items.
- In visual basic, **Arrays** can be declared by specifying the type of elements followed by the brackets **()** like as shown below.

```
Dim array_name As [Data_Type]();
```

Enumerations

- In visual basic, **Enum** is a keyword and it is useful to declare an enumeration. In visual basic, the enumeration is a type and that will contain a set of named constants as a list.

- By using enumeration, we can group constants that are logically related to each other. For example, the days of week can be grouped together by using enumeration in visual basic.

Syntax

```
Enum enum_name
' enumeration list
End Enum
```

Example

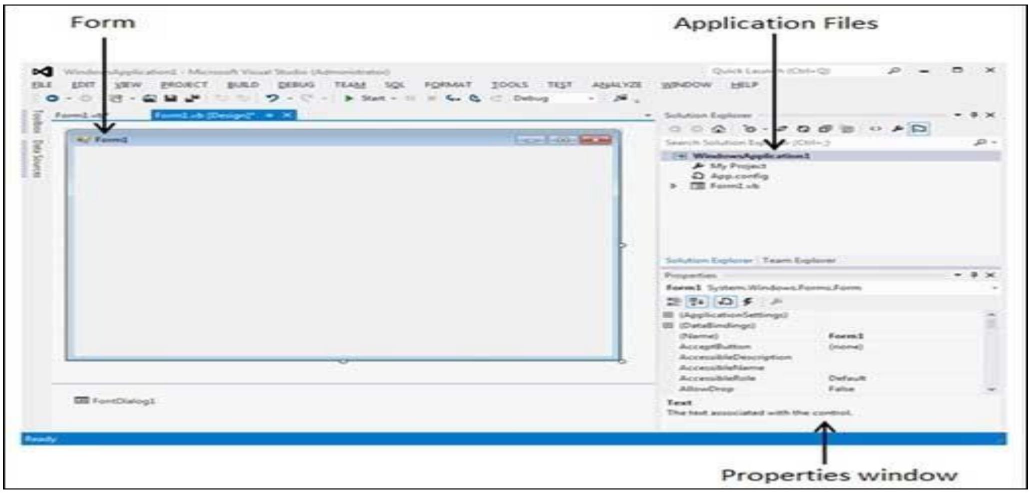
```
Enum Week
        Sunday
        Monday
        Tuesday
        Wednesday
        Thursday
        Friday
        Saturday
End Enum
```

In visual basic, by default the first named constant in enumerator has a value of **0**, and the value of each successive item in enumerator will be increased by **1**. For example, in the above enumeration, Sunday value is **0**, Monday is **1**, Tuesday is **2**, and so forth.

Introducing the Form Class:

Let's start with creating a Window Forms Application by following the following steps in Microsoft Visual Studio - File → New Project → Windows Forms Applications

Finally, select OK, Microsoft Visual Studio creates your project and displays following window Form with a name Form1.

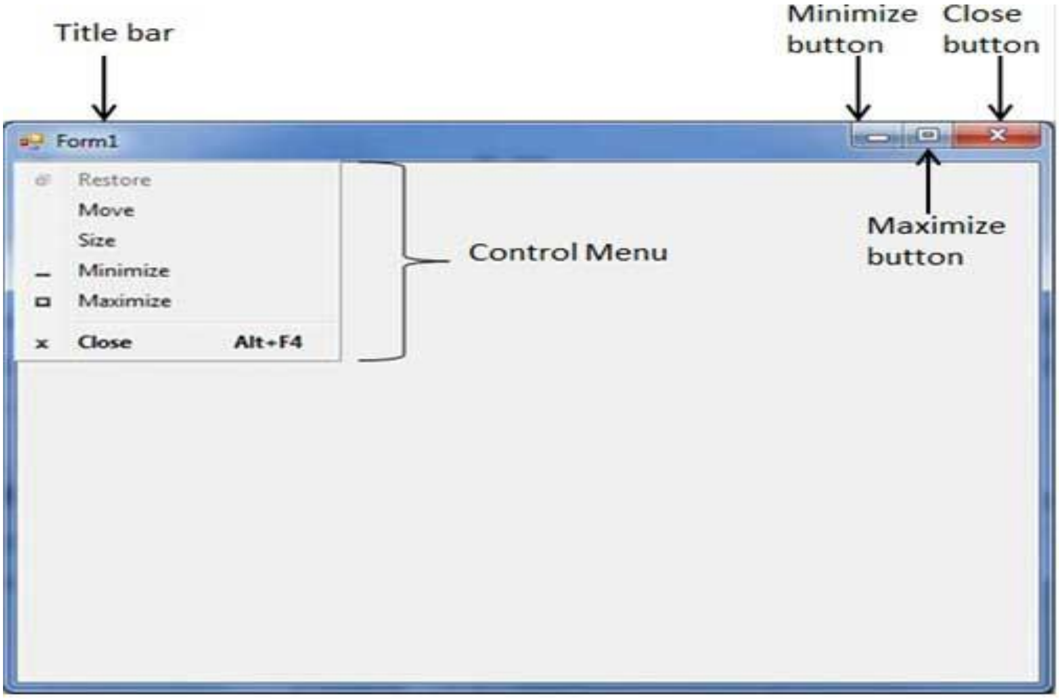


Visual Basic Form is the container for all the controls that make up the user interface. Every window you see in a running visual basic application is a form, thus the terms form and window describe the same entity.

Visual Studio creates a default form for you when you create a Windows Forms Application.

Every form will have title bar on which the form's caption is displayed and there will be buttons to close, maximize and minimize the form shown below –

If you click the icon on the top left corner, it opens the control menu, which contains the various commands to control the form like to move control from one place to another place, to maximize or minimize the form or to close the form.



Form Properties

Following table lists down various important properties related to a form. These properties can be set or read during application execution. You can refer to Microsoft documentation for a complete list of properties associated with a Form control –

S.No.	Properties	Description
1	AcceptButton	The button that's automatically activated when you press Enter, no matter which control has the focus at the time. Usually the OK button on a form is set as AcceptButton for a form.

2	CancelButton	The button that's automatically activated when you hit the Esc key. Usually, the Cancel button on a form is set as CancelButton for a form.
3	AutoScale	This Boolean property determines whether the controls you place on the form are automatically scaled to the height of the current font. The default value of this property is True. This is a property of the form, but it affects the controls on the form.
4	AutoScroll	This Boolean property indicates whether scroll bars will be automatically attached to the form if it is resized to a point that not all its controls are visible.
5	AutoScrollMinSize	This property lets you specify the minimum size of the form, before the scroll bars are attached.
6	AutoScrollPosition	The AutoScrollPosition is the number of pixels by which the two scroll bars were displaced from their initial locations.
7	BackColor	Sets the form background color.
8	BorderStyle	The BorderStyle property determines the style of the form's border and the appearance of the form − ● None − Borderless window that can't be resized. ● Sizable − This is default value and will be used for resizable window that's used for displaying regular forms. ● Fixed3D − Window with a visible border, "raised" relative to the main area. In this case, windows can't be resized. ● FixedDialog − A fixed window, used to create dialog boxes. ● FixedSingle − A fixed window with a single line border. ● FixedToolWindow − A fixed window with a Close button only. It looks like the toolbar displayed by the drawing and imaging applications. ● SizableToolWindow − Same as the FixedToolWindow but resizable. In addition, its caption font is smaller than the usual.
9	ControlBox	By default, this property is True and you can set it to False to hide the icon and disable the Control menu.

10	Enabled	If True, allows the form to respond to mouse and keyboard events; if False, disables form.
11	Font	This property specify font type, style, size
12	HelpButton	Determines whether a Help button should be displayed in the caption box of the form.
13	Height	This is the height of the Form in pixels.
14	MinimizeBox	By default, this property is True and you can set it to False to hide the Minimize button on the title bar.
15	MaximizeBox	By default, this property is True and you can set it to False to hide the Maximize button on the title bar.
16	MinimumSize	This specifies the minimum height and width of the window you can minimize.
17	MaximumSize	This specifies the maximum height and width of the window you maximize.
18	Name	This is the actual name of the form.
19	StartPosition	This property determines the initial position of the form when it's first displayed. It will have any of the following values − ● CenterParent − The form is centered in the area of its parent form. ● CenterScreen − The form is centered on the monitor. ● Manual − The location and size of the form will determine its starting position. ● WindowsDefaultBounds − The form is positioned at the default location and size determined by Windows. ● WindowsDefaultLocation − The form is positioned at the Windows default location and has the dimensions you've set at design time.
20	Text	The text, which will appear at the title bar of the form.
21	Top, Left	These two properties set or return the coordinates of the form's top-left corner in pixels.

22	TopMost	This property is a True/False value that lets you specify whether the form will remain on top of all other forms in your application. Its default property is False.
23	Width	This is the width of the form in pixel.

Form Methods

The following are some of the commonly used methods of the Form class.

Sr.No.	Method Name & Description
1	Activate Activates the form and gives it focus.
2	ActivateMdiChild Activates the MDI child of a form.
3	AddOwnedForm Adds an owned form to this form.
4	BringToFront Brings the control to the front of the z-order.
5	CenterToParent Centers the position of the form within the bounds of the parent form.
6	CenterToScreen Centers the form on the current screen.
7	Close Closes the form.
8	Contains Retrieves a value indicating whether the specified control is a child of the control.
9	Focus Sets input focus to the control.
10	Hide Conceals the control from the user.

11	Refresh Forces the control to invalidate its client area and immediately redraw itself and any child controls.
12	Scale(SizeF) Scales the control and all child controls by the specified scaling factor.
13	ScaleControl Scales the location, size, padding, and margin of a control.
14	ScaleCore Performs scaling of the form.
15	Select Activates the control.
16	SendToBack Sends the control to the back of the z-order.
17	SetAutoScrollMargin Sets the size of the auto-scroll margins.
18	SetDesktopBounds Sets the bounds of the form in desktop coordinates.
19	SetDesktopLocation Sets the location of the form in desktop coordinates.
20	SetDisplayRectLocation Positions the display window to the specified value.
21	Show Displays the control to the user.
22	ShowDialog Shows the form as a modal dialog box.

Form Events

Following table lists down various important events related to a form.

Sr.No.	Event	Description
1	Activated	Occurs when the form is activated in code or by the user.
2	Click	Occurs when the form is clicked.
3	Closed	Occurs before the form is closed.
4	Closing	Occurs when the form is closing.
5	DoubleClick	Occurs when the form control is double-clicked.
6	DragDrop	Occurs when a drag-and-drop operation is completed.
7	Enter	Occurs when the form is entered.
8	GotFocus	Occurs when the form control receives focus.
9	HelpButtonClicked	Occurs when the Help button is clicked.
10	KeyDown	Occurs when a key is pressed while the form has focus.
11	KeyPress	Occurs when a key is pressed while the form has focus.
12	KeyUp	Occurs when a key is released while the form has focus.
13	Load	Occurs before a form is displayed for the first time.
14	LostFocus	Occurs when the form loses focus.
15	MouseDown	Occurs when the mouse pointer is over the form and a mouse button is pressed.
16	MouseEnter	Occurs when the mouse pointer enters the form.
17	MouseHover	Occurs when the mouse pointer rests on the form.

18	MouseLeave	Occurs when the mouse pointer leaves the form.
19	MouseMove	Occurs when the mouse pointer is moved over the form.
20	MouseUp	Occurs when the mouse pointer is over the form and a mouse button is released.
21	MouseWheel	Occurs when the mouse wheel moves while the control has focus.
22	Move	Occurs when the form is moved.
23	Resize	Occurs when the control is resized.
24	Scroll	Occurs when the user or code scrolls through the client area.
25	Shown	Occurs whenever the form is first displayed.
26	VisibleChanged	Occurs when the Visible property value changes.

Performing Common Form Opeprations:

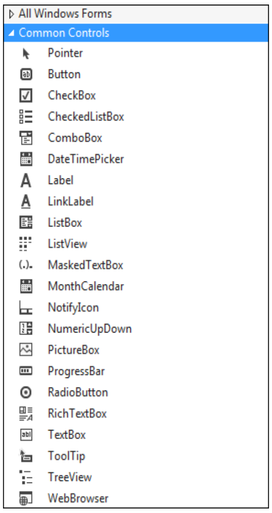
- Setting the title of a form
- Setting the initial position and state of a form
- Hiding the Minimize,Maximize , and Close button

FormStartPosition Enumeration

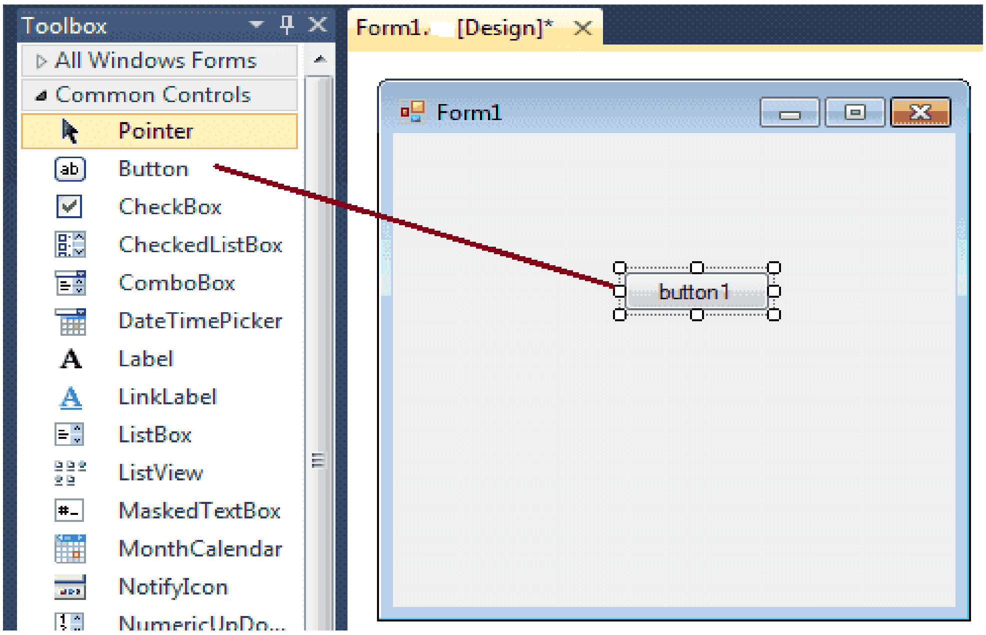
Member Name	Description
CenterParent	The form is centered within the bounds of its parent form.
CenterScreen	The form is centered on the current display, and has the dimensions specified in the form's size.
Manual	The position of the form is determined by the Location property.
WindowsDefaultBounds	The form is positioned at the Windows default location and has the bounds determined by Windows default.

WindowsDefaultLocation	The form is positioned at the Windows default location and has the dimensions specified in the form's size.

Adding Controls to a Form

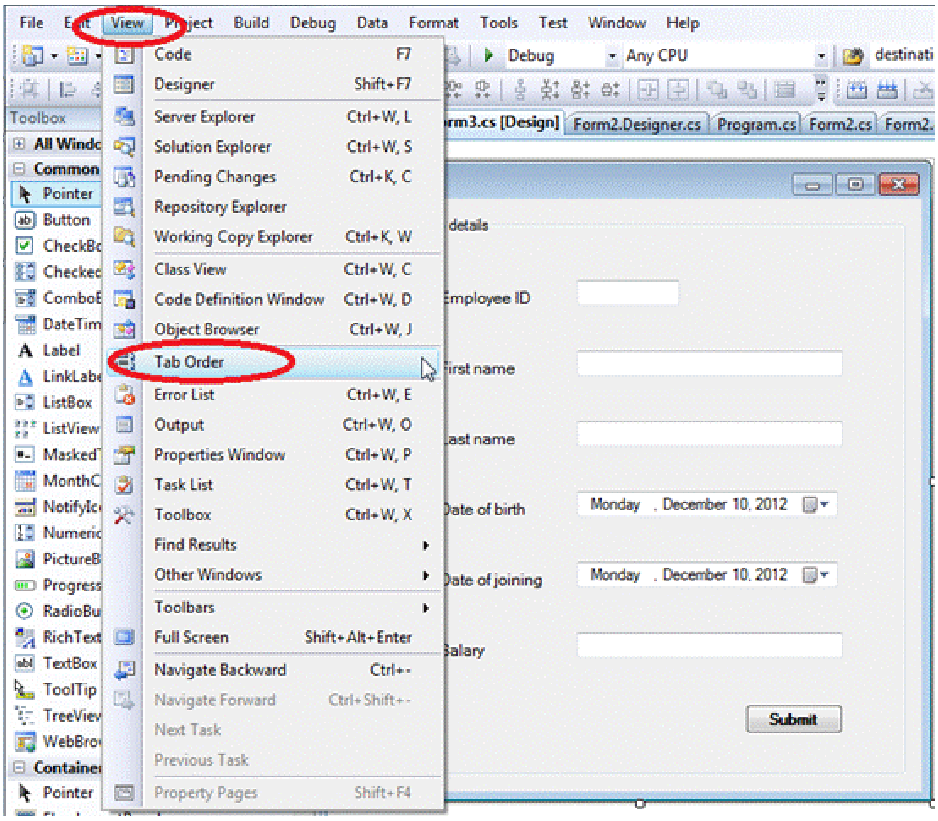


Each time you create a Windows application, Visual Studio will display a default blank form, onto which you can drag and drop controls from the Visual Studio Toolbox window.



Setting TabOrder of Controls

Add your controls on the and form and go to design view of the form and select **View -> Tab Order**



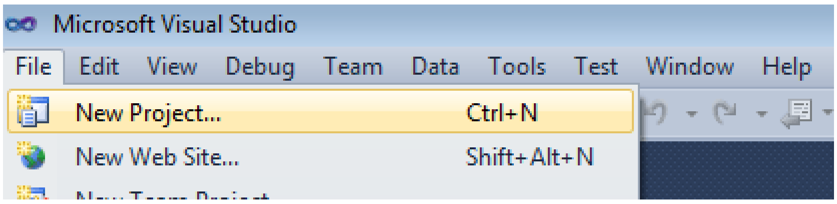
Docking and Anchoring Controls

- The Anchor and Dock properties of a form are two separate properties. Anchor refers to the position a control has relative to the edges of the form. A textbox, for example, that is anchored to the left edge of a form will stay in the same position as the form is resized.
- Docking refers to how much space you want the control to take up on the form. If you dock a control to the left of the form, it will stretch itself to the height of the form, but its width will stay the same.
- Docking is similar to Anchoring, but this time the control fills a certain area of the form.

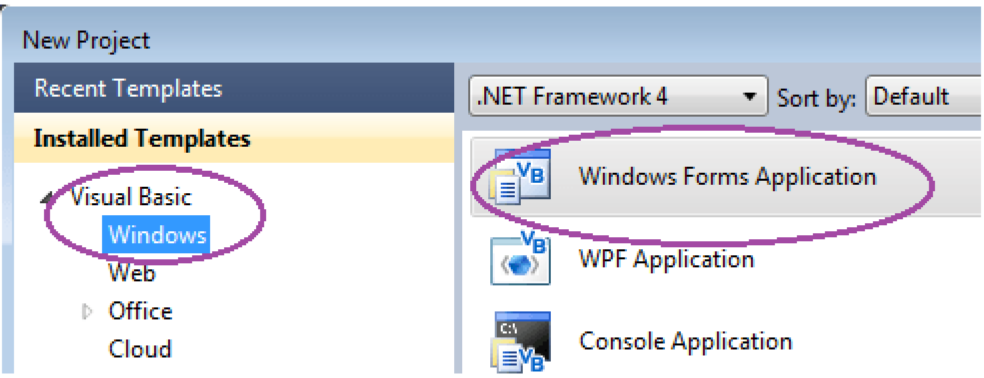
Adding Forms to Windows Form Application

- The first step is to start a new project and build a form. Open your Visual Studio and select File->NewProject and select Visual Basic from the New project dialog box and select Windows Froms Application. Enter your project name instead of WindowsApplication1 in the bottom of dialouge box and click OK button. The following picture shows how to create a new Form in Visual Studio.

- To work on multiple forms, create the new project and add a new form



Select project type from New project dialog Box.



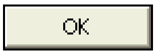
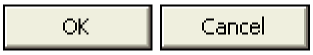
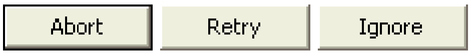
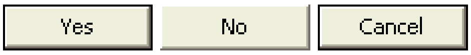
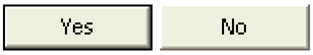
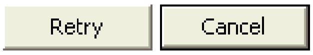
Creating Message Boxes

A message box is a special dialog box used to display a piece of information to the user. As opposed to a regular form, the user cannot type anything in the dialog box. To support message boxes, the Visual Basic language provides a function named **MsgBox**. To support message boxes, the .NET Framework provides a class named.

To display a simple message box, you can use the MsgBox() function with the following formula:

MsgBox(Message) the parentheses, pass a string. Example:

```
:       Private Sub btnMessage_Click(ByVal sender As System.Object, _
                    ByVal e As System.EventArgs) _
                    Handles btnMessage.Click
    MsgBox("Welcome to Microsoft Visual Basic")

End Sub
```

To Display	**MsgBoxStyle**	**Integral Value**
	**OKOnly**	0
	**OKCancel**	1
	**AbortRetryIgnore**	2
	**YesNoCancel**	3
	**YesNo**	4
	**RetryCancel**	5

MessageBoxButtons Enumeration

AbortRetryIgnore	The message box contains Abort ,Retry,and Ignore buttons
OK	The message box contains an OK button
OkCancel	The message box contains OK and Cancel buttons
RetryCancel	The message box contains Retry and Cancel buttons
YesNo	The message box contains Yes and No buttonx
YesNoCancel	The message box with Yes, No and Cancel buttons

The Members of MessageBoxIcon Enumeraion

Member	Description
Asterisk	The message box contains a symbol consisting of a lowercase letter i in a circle.

Error	The message box contains a symbol consisting of white X in a circle with a red background.
Exclamation	The message box contains a symbol consisting of an exclamation point in a triangle with a yellow background
Hand	The message box contains a symbol consisting of a white X in a circle with a red background.
Information	The message box contains a symbol consisting of a lowercase letter i in a circle.
None	The message box contains no symbols
Question	The message box contains a symbol consisting of a question mark in a circle. The question mark message icon is no longer recommended because it does not clearly represent a specific type of message and because the phrasing of a message as a question could apply to any message type. In addition, users can confuse the question mark symbol with a help information symbol. Therefore, do not use this question mark symbol in your message boxes. The system continues to support its inclusion only for backward compatibility.
Stop	The message box contains a symbol consisting of white X in a circle with a red background.
Warning	The message box contains a symbol consisting of an exclamation point in a triangle with a yellow background

The Members of MessageDefaultButton Enumeration

Member	Description
Button1	The first button on the message box is the default button.
Button2	The second button on the message box is the default button.
Button3	The third button on the message box is the default button.

The Members of MessageBoxOptions Enumeration

Member	Description
RightAllign	The message box text is right-aligned.
RtlReading	Specifies that the message box text is displayed with right to left reading order.
ServiceNotification	The message box is displayed on the active desktop. The caller is a

	service notifying the user of an event. Show displays a message box on the current active desktop, even if there is no user logged on to the computer
DefaultDesktopOnly	The message box is displayed on the active desktop. This constant is similar to ServiceNotification, except that the system displays the message box only on the default desktop of the interactive window station. The application that displayed the message box loses focus, and the message box is displayed without using visual styles

The Members of DialogResult Enumeration

Member	Description
Abort	Returns Abort
Cancel	Returns Cancel
Ignore	Returns Ignore
No	Returns No
None	Returns nothing from the dialog box

The Members of DialogResult Enumeration

Member	Description
Ok	Return Ok
Retry	Return Retry
Yes	Return Yes

Creating Input Boxes

The **Input Box**. When an **input box** displays, it presents a request to the user who can then provide a value. After using the **input box**, the user can change his or her mind and press Esc or click Cancel. If the user provided a value and want to acknowledge it, he or she can click OK or press Enter.

Creating Dialog Boxes

- Add controls such as Label, Dialog boxes are used to interact with the user and retrieve information. In simple terms, a dialog box is a form with its FormBorderStyle enumeration property set to FixedDialog.
- You can construct your own custom dialog boxes by using the Windows Forms Designer in Visual Studio. Textbox, and Button to customize dialog boxes to your specific needs. The .NET Framework also includes predefined dialog boxes, such as File Open and message boxes, which you can adapt for your own applications.

Handling Events

VB.Net is an event-driven language. There are mainly two types of events −

- Mouse events
- Keyboard events

Handling Mouse Events

Mouse events occur with mouse movements in forms and controls. Following are the various mouse events related with a Control class −

- MouseDown − it occurs when a mouse button is pressed
- MouseEnter − it occurs when the mouse pointer enters the control
- MouseHover − it occurs when the mouse pointer hovers over the control
- MouseLeave − it occurs when the mouse pointer leaves the control
- MouseMove − it occurs when the mouse pointer moves over the control
- MouseUp − it occurs when the mouse pointer is over the control and the mouse button is released
- MouseWheel − it occurs when the mouse wheel moves and the control has focus

The event handlers of the mouse events get an argument of type MouseEventArgs. The MouseEventArgs object is used for handling mouse events. It has the following properties −

- Buttons − indicates the mouse button pressed
- Clicks − indicates the number of clicks
- Delta − indicates the number of detents the mouse wheel rotated
- X − indicates the x-coordinate of mouse click
- Y − indicates the y-coordinate of mouse click

Handling Keyboard Events

Following are the various keyboard events related with a Control class −

- KeyDown − occurs when a key is pressed down and the control has focus
- KeyPress − occurs when a key is pressed and the control has focus
- KeyUp − occurs when a key is released while the control has focus

The event handlers of the KeyDown and KeyUp events get an argument of type KeyEventArgs. This object has the following properties −

- Alt − it indicates whether the ALT key is pressed
- Control − it indicates whether the CTRL key is pressed
- Handled − it indicates whether the event is handled
- KeyCode − stores the keyboard code for the event
- KeyData − stores the keyboard data for the event
- KeyValue − stores the keyboard value for the event
- Modifiers − it indicates which modifier keys (Ctrl, Shift, and/or Alt) are pressed
- Shift − it indicates if the Shift key is pressed

The event handlers of the KeyDown and KeyUp events get an argument of type KeyEventArgs. This object has the following properties −

- Handled − indicates if the KeyPress event is handled
- KeyChar − stores the character corresponding to the key pressed

UNIT- II

Windows Forms Controls-I

Introducing the Control Class

Windows Form Controls

All the windows Forms controls inherit the common properties, methods and events of the Control class.

Properties of Control Class

Property	Description
Anchor	Obtains or sets the edges of the parent control to which the child control is bound and determines how the control is resized with respect to the size of its parent control
BackColor	Obtains or sets the background color of the control
BackgroundImage	Obtains or sets the background

Text Box

Text Box is used to accept textual input from the user. The user can add strings, numerical values and a combination of those, but Images and other multimedia content are not supported.

Label

It is used to show any text to the user, typically the text in a label does not change while the application is running.

Button

It is used as a standard Windows Button. In most cases, the Button Control is used to generate a click event, its name, size and appearance are not changed in the runtime.

ListBox

As the name suggests, this control works as a way to display a list of items on the application. Users can select any options from the list.

Radio button:

The RadioButton control is used to provide a set of mutually exclusive options. The user can select one radio button in a group. If you need to place more than one group of radio buttons in the same form, you should place them in different container controls like a GroupBox control.

Combobox:

The ComboBox control is used to display a drop-down list of various items. It is a combination of a text box in which the user enters an item and a drop-down list from which the user selects an item.

Listbox:

The ListBox represents a Windows control to display a list of items to a user. A user can select an item from the list. It allows the programmer to add items at design time by using the properties window or at the runtime.

Properties of the ListBox Control

The following are some of the commonly used properties of the ListBox control −

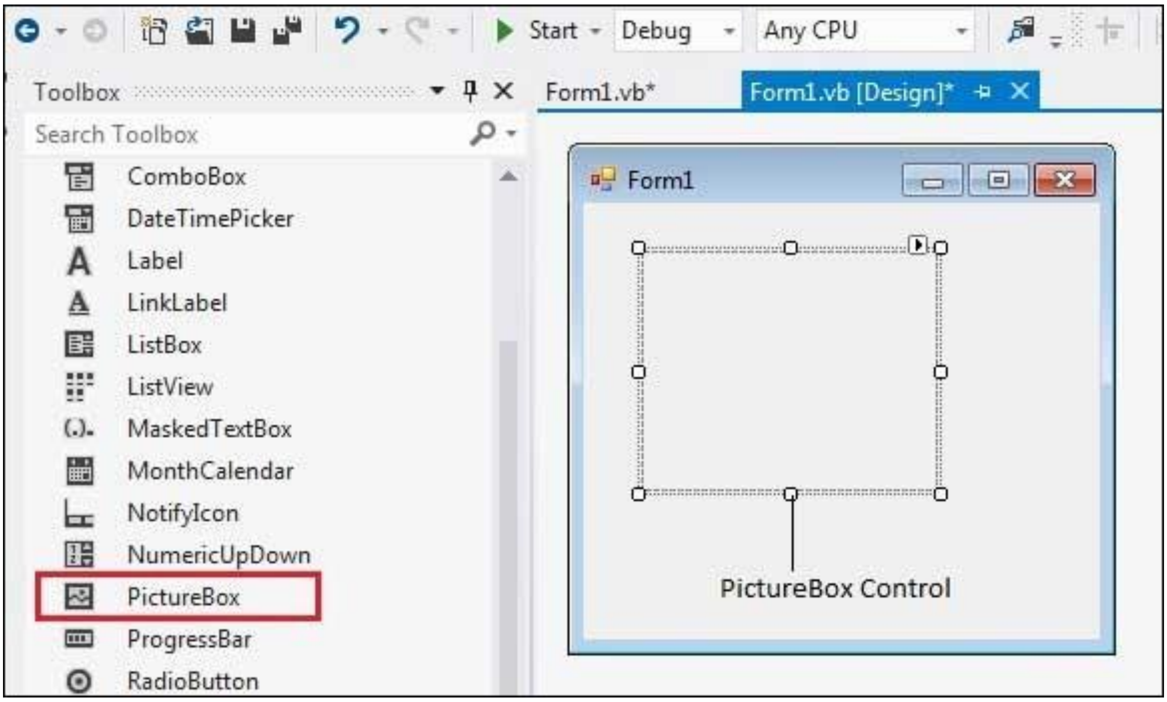
Sr.No.	Property & Description
1	**AllowSelection** Gets a value indicating whether the ListBox currently enables selection of list items.
2	**BorderStyle** Gets or sets the type of border drawn around the list box.
3	**ColumnWidth** Gets of sets the width of columns in a multicolumn list box.
4	**HorizontalExtent** Gets or sets the horizontal scrolling area of a list box.
5	**HorizontalScrollBar** Gets or sets the value indicating whether a horizontal scrollbar is displayed in the list box.
6	**ItemHeight** Gets or sets the height of an item in the list box.
7	**Items** Gets the items of the list bo

Panel control:

The **Panel control** is a container of other **controls**. The **Panel control** is displayed by default without any borders at run time. Drag and drop **Panel control** from toolbox on the window Form.

Picturebox control:

The PictureBox control is used for displaying images on the form. The Image property of the control allows you to set an image both at design time or at run time.



ProgressBar Control:

It represents a Windows progress bar control. It is used to provide visual feedback to your users about the status of some task. It shows a bar that fills in from left to right as the operation progresses.

WINDOWS FORMS CONTROLS-II

The Windows Forms ToolStrip control and its associated classes provide a common framework for combining user interface elements into toolbars, status bars, and menus. ToolStrip controls offer a rich design-time experience that includes in-place activation and editing, custom layout, and rafting, which is the ability of toolbars to share horizontal or vertical space. Although ToolStrip replaces and adds functionality to the control in previous versions, ToolBar is retained for both backward compatibility and future use if desired.

Features of the ToolStrip Controls

Use the ToolStrip control to:

- Present a common user interface across containers.
- Create easily customized, commonly employed toolbars that support advanced user interface and layout features, such as docking, rafting, buttons with text and images,

drop-down buttons and controls, overflow buttons, and run-time reordering of ToolStrip items.

- Support overflow and run-time item reordering. The overflow feature moves items to a drop-down menu when there is not enough room to display them in a ToolStrip.
- Support the typical appearance and behaviour of the operating system through a common rendering model.
- Handle events consistently for all containers and contained items, in the same way you handle events for other controls.
- Drag items from one ToolStrip to another or within a ToolStrip.
- Create drop-down controls and user interface type editors with advanced layouts in a ToolStripDropDown.

Properties of the Tool strip class

- TabStop
- TextDirection
- Methods of the ToolStrip Class
- GetItemAt

The ToolStrip control contains the following types of controls that are referred as too strip items:

- ToolStripButton
- ToolStripLabel
- ToolStripTextBox
- ToolStripComboBox
- ToolStripSeparator
- ToolStripDropDownButton

Properties of the ToolStrip Items

- Alignment
- AutoSize
- AutoToolTip
- CanSelect
- Image
- Name
- Text
- TextDirection
- ToolTipBox

Methods of the ToolStrip Class

- GetCurrentParent
- PerformClick
- Select

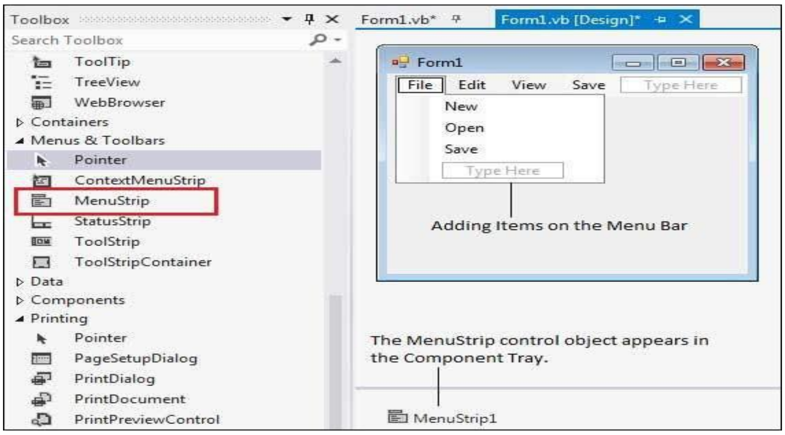
Events of the ToolStrip Items

- Click
- DisplayStylecchanged
- TextChanged
- Using the MenuStrip Contorl
- ToolStripLabel

Using MenuStripControl

- The **MenuStrip** control represents the container for the menu structure.
- The MenuStrip control works as the top-level container for the menu structure. The ToolStripMenuItem class and the ToolStripDropDownMenu class provide the functionalities to create menu items, sub menus and drop-down menu

The following diagram shows adding a MenuStrip control on the form –



Adding Items on the Menu Bar

The MenuStrip control object appears in the Component Tray.

The following are some of the commonly used properties of the MenuStrip control –

Sr.No.	Property & Description
1	**CanOverflow** Gets or sets a value indicating whether the MenuStrip supports overflow functionality.
2	**ShowItemToolTips** Gets or sets a value indicating whether ToolTips are shown for the MenuStrip.

3	**Stretch** Gets or sets a value indicating whether the MenuStrip stretches from end to end in its container.

Events of the MenuStrip Control

The following are some of the commonly used events of the MenuStrip control −

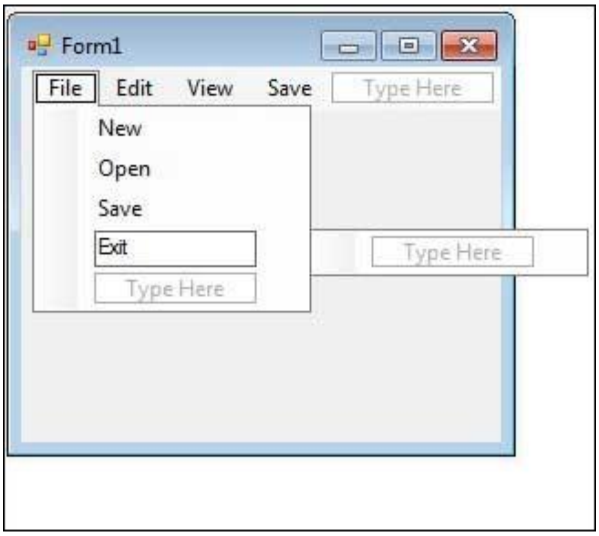
Sr.No.	Event & Description
1	**MenuActivate** Occurs when the user accesses the menu with the keyboard or mouse.
2	**MenuDeactivate** Occurs when the MenuStrip is deactivated.

Example

In this example, let us add menu and sub-menu items.

Take the following steps −

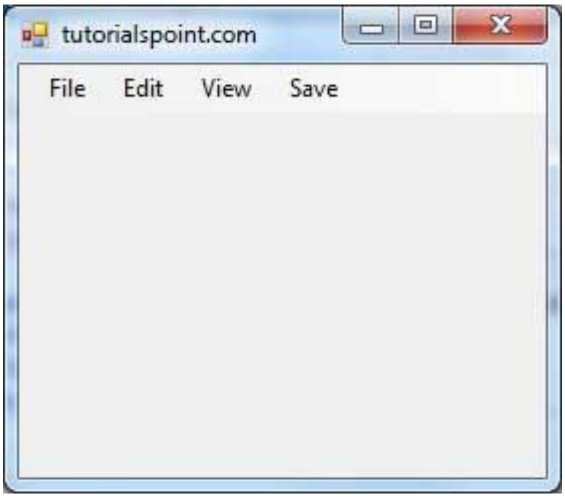
- Drag and drop or double click on a MenuStrip control, to add it to the form.
- Click the Type Here text to open a text box and enter the names of the menu items or sub-menu items you want. When you add a sub-menu, another text box with 'Type Here' text opens below it.
- Complete the menu structure shown in the diagram above.
- Add a sub menu **Exit** under the **File** menu.



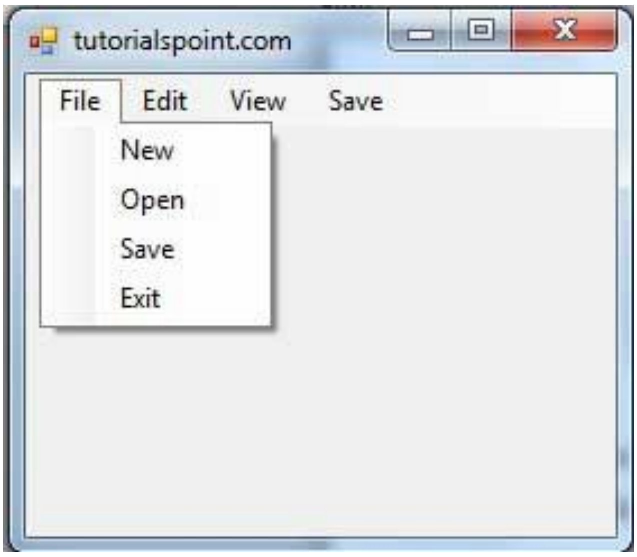
- Double-Click the Exit menu created and add the following code to the **Click** event of **ExitToolStripMenuItem** −

```
Private Sub ExitToolStripMenuItem_Click(sender As Object, e As
EventArgs) _
   Handles ExitToolStripMenuItem.Click
   End
End Sub
```

When the above code is executed and run using **Start** button available at the Microsoft Visual Studio tool bar, it will show the following window:



Click on the File -> Exit to exit from the application −



Using the StatusStrip Control

The StatusStrip control allows us to display a variety of information such as name or description of another control, the progress of a ongoing task and the instructions for users to perform a particular task.

Properties of the StatusStrip Class

- Dock
- LayoutStyle
- ShowItemToolTips
- SizingGripStretch
- Stretch

WORKING WITH DIALOGBOXES

- There are many built-in dialog boxes to be used in Windows forms for various tasks like opening and saving files, printing a page, providing choices for colors, fonts, page setup, etc., to the user of an application. These built-in dialog boxes reduce the developer's time and workload.
- All of these dialog box control classes inherit from the **CommonDialog** class and override the
- RunDialog function of the base class to create the specific dialog box.
- The RunDialog function is automatically invoked when a user of a dialog box calls its ShowDialo*g* function
- The **ShowDialog** method is used to display all the dialog box controls at run-time. It returns a value of the type of **DialogResult** enumeration.

The values of DialogResult enumeration are:

Abort - returns DialogResult. Abort value, when user clicks an Abort button.
Cancel- returns DialogResult.Cancel, when user clicks a

Cancel button.
Ignore - returns DialogResult.Ignore, when user clicks an Ignore button.
No - DialogResult.No, when user clicks a No button.
None - returns nothing and the dialog box continues running.
OK - DialogResult.OK, when user clicks an OK button
Retry - returns DialogResult.Retry , when user clicks an Retry button
Yes - returns DialogResult.Yes, when user clicks an Yes button.

Some commonly used dialog box controls are given as follows:

1. **FolderBrowserDialog Control**

   It allows users to work the Browse For Folder dialog box where they can browse through folders, create new folders, and select folders to view the containing files.

2. **OpenFileDialog Control**

   It prompts the user to open a file and allows the user to select a file to open.

   Properties of the OpenFileDialog class

   - ✓ CheckFilesExists
   - ✓ ShowReadOnly
   - ✓ ReadOnlyChecked
   - ✓ SafeFileName
   - ✓ SaffeFileName

3. **SaveFileDialog Control:**

   It prompts the user to select a location for saving a file and allows the user to specify the name of the file to save data

4. **FontDialog Control:**

   It prompts the user to choose a font from among those installed on the local computer and lets the user select the font, font size, and color.

   Properties of the FontDialog class are

   - Color
   - Font
   - ShowColor
   - ShowEffects
   - MaxSize
   - MinSize

5. **ColorDialog Control:**

It represents a common dialog box that displays available colors along with controls that enable the user to define custom colors. Properties of the ColorDialog class are

- AnyColor
- CustomColors
- FullOpen
- ShowColorOnly
- AllowFullOpen

6. **PrintDialog Control**

It lets the user to print documents by selecting a printer and choosing which sections of the documentto print from a Windows Forms application.

7. **PrintDocument Control:**

Represents the document to print. This control allows us to set various options and settings to print a document.

Events of the PrintDocument Class are

- BeginPrint
- EndPrint
- PrintPage
- QueryPageSettings

WINDOWS PRESENTAATION FOUNDATION

- WPF stands for Windows Presentation Foundation. It is a powerful framework for building Windows applications. It provides various controls, such as labels, radio buttons, tool strips, and menu strips, to design the user interface of our application.

- .NET frame work has incorporated a new techonology known as Windows Presentation Foundation(WPF),which was previously known as Avalon to address the issues of employing different technologies to work with 2D and 3D graphics and Multimedia in Windows Forms Application.

- WPF is a combination of various technologies such as GDI+ , windows media player, and DirectX to facilitate the development of high end desktop applications. With WPF we can easily and conveniently develop graphic rich desktop applications because of its support for multimedia such as text, images audio, video and 2D ,and 3D graphics that was not available for windows forms applications.

EXPLORING THE IMPROVEMENTS IN WPF4.5

WPF forms an essential component of the .NET framework that enables .NET application developers to create WPF applications with graphic rich UI's.Windows applications that have a modern graphical UI. The improvements in WPF 4.5 are as follows:

- **The Ribbon Control:** The Ribbon control enables us to place some controls on a horizontal bar called ribbon which is present at top of the edge of an application window.

- The **Ribbon button** functionally works nearly similarly the same as a normal WPF button. This is for an event and communicating with business logic. It is responsible for providing an interface for the actions exposed by our application.

- Support for validating and Data Synchronously and Asynchronously- Implementing the INotifyDataErrorInfor interface.

- Support for Binding to Types that implement ICustomTypeProvider –Implement ing the ICustomTypeProvider interface.

- Support for Fetching Binding Information**:** -WPF 4.5 API's help to fetch data binding information .

- Data Binding to Static Properties**:**
    - WPF4.5 enables to establish data binding using static properties of a class. A static property is shared by each instance of a class and can be used as the source of data binding. An event can then be defined that is raised whenever the value of the static property changes

- Support for Accessing Collections on a Non-UI Thread
- Support for Automatically Updating a Source of a Data Binding
- New VirtualizingPanel Features
- Support for implementing Weak Reference to an Event
- New methods Defined in the Dispatcher Class
- Support for the Use of Markup Extension for Events
- Support for rearrangement of Data on Modification
- Improved performance when Displaying Large sets of Grouped data
- ✓ Support for the validation of DataContext object

Explaining WPF 4.5 Architecture

WPF 4.5 has both managed and unmanaged components. Managed components refer to those components that services of Common Language Runtime(CLR). Such as garbage collection and type safety with the .NET environment. While unmanaged components are those components that do not use the CLR services.

Essential components of WPF architecture are

- The Presentation Framework Component
- The PresentationCore Component
- Windows Base
- TheMilCore Component

Describing types of WPF Applications

There are various applications development features such as controls, graphics ,layout, and data binding that are supported by WPF. It is a next generation of UI framework that enables us to develop application with a rich user experience WPF application.

WPF application are mainly of the following two types:

- StandaloneWPF applications
- XAML Browser Applications(XBAP's)

StandaloneWPF application:

WPF standalone applications are similar to windows Forms applications .

XBAP'S Applications-

- ✓ These are WPF applications that are hosted on a Web Server and run in a Web Browser, such as Windows Explorer or Mozilla FireFox. An XBAP is similar to a website as it consists of pages similar to Web pages in a website that allows to navigate in the Web browser.
- ✓ XBAP's Application can perform by setting Margin lines and Margin Stubs, Sanaplines.

Using XAML in WPHF:

- WPF supports XAML , which is a declarative markup language based on Extensible Markup Language(XML). It allows us to easily and quickly define the elements for designing the UI of WPF applications.
- We can then use the UI elements defined in XAML to specify the logic for these elements in the code-behind file, which can be used to design the UI of a WPF application at run time.
- As a result, WPF allows both segregation and integration of the UI and the application logic in WPF applications.
- Visual studio 2012 extends support for XAML by incorporating XAML IntelliSense and allowing the debugging and complication of XAML content.

Following are the XAML components that help us to work with XAML :

- XAML Elements and Attributes
- Name space and XAML
- Markup Extension

WORKING WITH WPF CONTROLS

A control is an UI element that allows interaction between the application and its users.WPF has a rich set of controls that allows us to develop graphic rich applications.

Some important WPF controls that we can use in our WPF applications in Visual Studio 2012 are as follows:

Ribbon Control:

- The Ribbon control enables us to place some controls on a horizontal bar called ribbon which is present at top of the edge of an application window.

- Some Properties of Ribbon Class are AllowDrop, Cursor, Background ,FlowDirection.

Using Grid Control:

- A Grid control is automatically added to a WPF application .The Grid control is a child element of the Window element and Page element because it is present in both the application.
- Some Properties of the Grid Class are ColumnDefinitions,RowDefinitions,ShowGridLines,Columns,Row

Using the Button Control:

- It is same as the button control in Windows forms applications. It allows us to perform an action when a user clicks it.
- Some of the properties of button class are IsCancel,Content,IsDefault,IsDefaultedat

Using the PasswordBox Control:

- There may be situations when we want users to enter some confidential information, such as passwords. Textbox control allows users to enter their passwords.
- The PasswordBox control is a special type of textbox, which is particularly aimed at entering passwords such that they remain

undisclosed to other users.We need to set the PasswrodChar property by a special character that masked the input text entered by the user.

- Some property of the PasswordBox Class are MaxLength, Password,,PasswordChar

Using the TextBlock Control:

- The TextBox control in WPF allows us to Work with the text in a flexible manner as compared to the traditional Label control.
- This implies that it has several properties that allow us to modify the appearance of the text entered in the TextBlock control. It is an instance of the TextBlock class.
- Some of the properties of the TextBlock Classs are FontFamily,FontSize,Text,TextAlignment.

Using the border Control:

- The Border control in WPF allows us to add a border, background, or both to other WPF controls. It can have only one child control, that is we cannot use the Border control with mu,tiple child controls. The Border control is an instance of the Border class.
- Some of the properties of the border Class are Background,BorderBrush,BorderThickness,CornerRadius

Using the GridSplitter Control:

- The GridSplitter control in WPF offers a unique facility by allowing us to redistribute the space between the cells of a Grid control.
- With the GridSpiltter control, the height and width of the Grid control do not change ; only the space between the cells of the grid changes. The Grid Splitter control is an instance of the GridSplitter class.
- Some of the properties of the GridSplitter Class are PreviewStyle, ResizeBehaviour, ResizeDirection, ShowPreview.

Using the Canvas Control:

- If we want to place a control at an exact location rather than moving it on the window or specifying the margins for the control to place it appropriately we use the Canvas control, which allows us to place other controls at exact or absolute positions inside it.
- The child controls are positioned by using coordinates with respect to the size of the Canvas control. It is an instance of the Canvas class, which is derived from the Panel class.
- Some of the properties of the Canvas Class are Bottom, Left, Right, Top

Using the StackPanel Control:

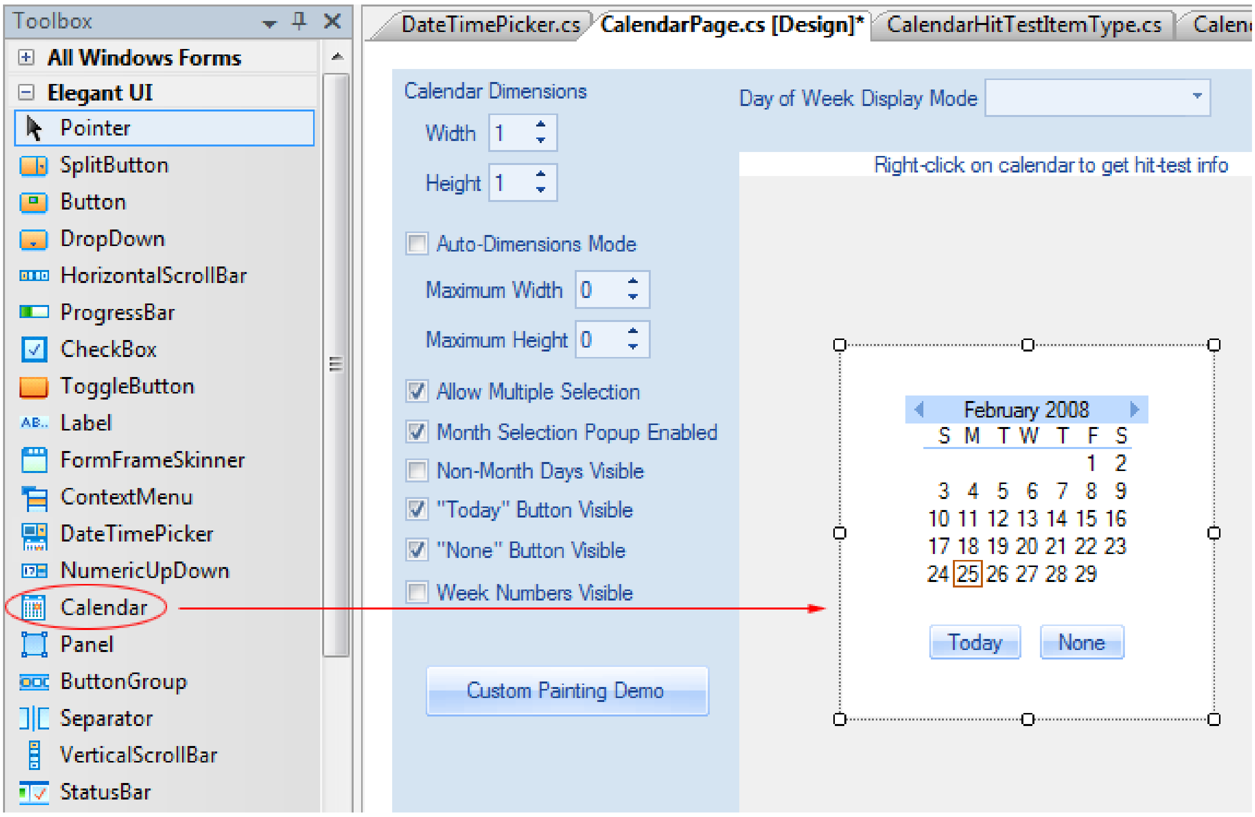
- If we want to place multiple controls on a WPF window such that one control appears on the top of another control, forming a vertical stack of controls.
- Instead of using Canvas control and placing the controls at exact locations we can use a StackPanel control. The StackPanel Class is a container control that allows us to position its child controls in a vertical or horizontal .
- By default stack.controls of the StackPanel controls of the StackPanel control are stacked vertically.
- The child controls horizontally. The horizontal stack starts from the left edge of the StackPanel control and ends at its right edge.
- The StackPanel control is an instance of the StackPanel class which is derived from the Panel class.

Using the DataGrid Control

- The DataGrid control is a new WPF control introduced in WPF 4.5. Similar to DataGrid control in ASP NET web forms and the Data/Grid View control in Windows Forms.
- The DataGrid control to display the data from a database or from ay collection, which implements IEnumerable collection.
- Some of the Properties of the DataGrid class are Columns, Items,CellStyle.

Using the Calendar Control:

- The Calendar control is another control in WPF 4.5. User nselect a particular date.
- Whenever a date is selected in a Calendar control, an event is fired for the selected date. It is an instance of Calendar class which is placed inside the System.Windows.Controls namespace.
- Some of the properties of the Calendar class are BlackOutDates, DisplayDate, DisplayDateStart



Calendar Dimensions

Width 1

Height 1

□ Auto-Dimensions Mode

Maximum Width 0

Maximum Height 0

☑ Allow Multiple Selection

☑ Month Selection Popup Enabled

□ Non-Month Days Visible

☑ "Today" Button Visible

☑ "None" Button Visible

□ Week Numbers Visible

Day of Week Display Mode

Right-click on calendar to get hit-test info

Using the DatePicker Control:

- ✓ This control allows us to select a date. The difference between calendar and DatePicker control is that while the Calendar control of only a calendar from which we can select a date, the DatePicker Control is a combination of Combobox and calendar.
- ✓ It is an instance of the System.Windows.Controls namespace.
  - ✓ Some of the properties of the DatePicker class are BlackOutDates, DisplayDate, DisplayDateStart .

Working with Resources and Styles:

- ✓ If a WPF stand alone application with multiple windows and we want to apply a resource to all the windows, then we need to define the resource in the **Application.xaml** file of the application.

Using the Static Resources :

- The resources that are referred by using the StaticResource markup extension are known as the static resources. We can use the references throughout the runtime , the value of the resource does not change once it is referenced.

- When a static resource is used as a property value, the key of the static resources is searched in the available resource at the time of loading the application. If a resource with the given key is found then the value of a static resource is assigned to the property.

Using the Dynamic Resources:

- It can change the value of the resource at runtime through user or system settings.
- With dynamic resources, an expression is created for the requested resource. This expression is not evaluated until runtime. With the dynamic resource is used to assign a property value at runtime, the key of the dynamic resource is searched in the available resources.
- If the dynamic resource with the given key is found, then the expression is evaluated and assigned as the property value.

Using Styles as Resources:

- Apply a given style on several controls or elements uniformly. We can use the Style element to define a style as a resource.
- Each Style element specific an element to which we want to apply the Style. A Style element also has one or more Setter elements that specify the properties of the element.

.