What is Microsoft .NET?

- Microsoft .NET (pronounced "dot net") is a software component that runs on the Windows operating system. .NET provides tools and libraries that enable developers to create Windows software much faster and easier. .NET benefits end-users by providing applications of higher capability, quality and security. The .NET Framework must be installed on a user's PC to run .NET applications.
- .NET technology provides the ability to quickly build, deploy, manage, and use connected, security-enhanced solutions with Web services

What is VB.Net?

- Visual Basic .NET (VB.NET), is an object-oriented computer programming language that can be viewed as an evolution of the classic Visual Basic (VB), which is implemented on the .NET Framework
- It is the next generation of the visual Basic language.
- It supports OOP concepts such as abstraction, inheritance, polymorphism, and aggregation.

.NET Framework Architecture :-

A programming infrastructure created by Microsoft for building, deploying, and running applications and services that use .NET technologies, such as desktop applications and Web services

1) It is a platform for application developers.
2) It is tiered, modular, and hierarchal.
3) It is a service or platform for building, deploying and running applications.
4) It consists of 2 main parts: Common language runtime and class libraries.

- The common language runtime is the bottom tier, the least abstracted.
- The .NET Framework is partitioned into modules, each with its own distinct responsibility.
- The architectural layout of the .NET Framework is illustrated in following figure:

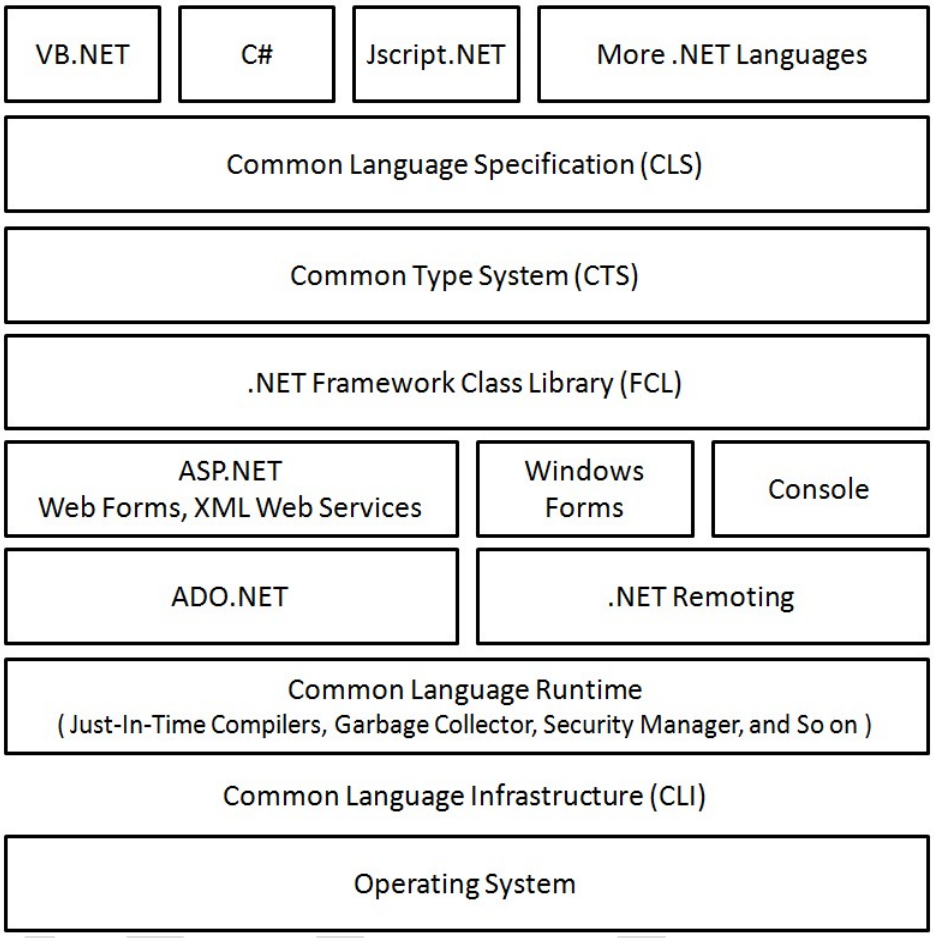


Figure 1 An overview of the .NET architecture.

Here we examine the following key components of the .NET Framework:

1) **Common Language Infrastructure (CLI):** The purpose of the Common Language Infrastructure (CLI) is to provide a language-neutral platform for application development and execution, including functions for Exception handling, Garbage Collection, security, and interoperability.
2) **Common Language Runtime (CLR):** The .NET Framework provides a runtime environment called the Common Language Runtime or CLR (similar to the Java Virtual Machine or JVM in Java), which handles the execution of code and provides useful services for the implementation of the program.

The CLR is the execution engine for .NET applications and serves as the interface between .NET applications and the operating system. The CLR provides many services such as:

- Loads and executes code
- Converts intermediate language to native machine code
- Manages memory and objects
- Enforces code and access security
- Handles exceptions
- Interfaces between managed code, COM objects, and DLLs
- Provides type-checking
- Provides code meta data (Reflection)
- Provides profiling, debugging, etc.
- Separates processes and memory

3) **Framework Class Library (FCL):** It is also known as a base class library. The FCL is a collection of over 7000 reusable classes, interfaces, and value types that enable .NET applications to :
    a) read and write files,
    b) access databases,
    c) process XML,
    d) display a graphical user interface,
    e) draw graphics,
    f) use Web services, etc.

- The .Net Framework class library (FCL) organized in a hierarchical tree structure and it is divided into Namespaces. Namespaces is a logical grouping of types for the purpose of identification. Framework class library (FCL) provides the consistent base types that are used across all .NET enabled languages. The Classes are accessed by namespaces, which reside within Assemblies.
- Other name of FCL is BCL – Base Class Library

4) Common Type System (CTS)

1) CTS allows written in different programming to easily share to information.
2) A class written in C# should be equivalent to a class written in VB.NET.
3) Languages must agree on the meanings of these concepts before they can integrate with one and other.
4) CLS forms a subset of Common type system this implies that all the rules that for apply to Common type system apply to common language specification.
5) It defines rules that a programming language must follow to ensure that objects written in different programming languages can interact which each other.
6) Common type system provide cross language integration.

- The common type system supports two general categories of types:
    a) Value Type
    b) Reference Type

a) **Value Type:** Stores directly data on stack. In built data type. For ex. Dim a as integer.
b) **Reference Type:** Store a reference to the value's memory address, and are allocated on the heap. For ex: dim obj as new oledbconnection.

- The Common Language Runtime (CLR) can load and execute the source code written in any .Net language, only if the type is described in the Common Type System (CTS)

7) Common Language Specification (CLS)

The CLS is a common platform that integrates code and components from multiple .NET programming languages. In other words, a .NET application can be written in multiple programming languages with no extra work by the developer (though converting code between languages can be tricky).

.NET includes new object-oriented programming languages such as C#, Visual Basic .NET, J# (a Java clone) and Managed C++. These languages, plus other experimental languages like F#, all compile to the Common Language Specification and can work together in the same application.

- CLS includes basic Language features needed by almost all the application
- It serves as a guide for Library Writers and Compiler Writer.
- The Common Language Specification is a subset of the common type system (CTS).
- The common language specification is also important to application to who are writing codes that will be used by other developers.
- CLS a series of basic rules that are required for language integration.

8) **Microsoft Intermediate Language:**

- When you compile your Visual Basic .NET source code, it is changed to an intermediate language (IL) that the CLR and all other .NET development environments understand.
- All .NET languages compile code to this IL, which is known as Microsoft Intermediate Language, MSIL, or IL.
- MSIL is a common language in the sense that the same programming tasks written with different .NET languages produce the same IL code.
- At the IL level, all .NET code is the same regardless of whether it came from C++ or Visual Basic.
- When a compiler produces Microsoft Intermediate Language (MSIL), it also produces Metadata.
- The Microsoft Intermediate Language (MSIL) and Metadata are contained in a portable executable (PE) file.
- Microsoft Intermediate Language (MSIL) includes instructions for loading, storing, initializing, and calling methods on objects, as well as instructions for arithmetic and logical operations, control flow, direct memory access, exception handling, and other operations

Advantages :

- It offers cross− language integration, including cross− language inheritance, which allows you to create a new class by deriving it from a base class written in another language.
- It facilitates automatic memory management, known as garbage collection.
- compilation is much quicker
- It allows you to compile code once and then run it on any CPU and operating system that supports the runtime.

Disadvantages:

- IL is not compiled to machine, so it can more easily be reverse engineered. Defense mechanisms for handling this are likely to follow shortly after the .NET Framework is officially released.
- While IL is further compiled to machine code, a tiny percentage of algorithms will require a direct unwrapped access to system resources and hardware.

- Figure: 2 shows what happens to your code from its inception in Visual Studio to execution.

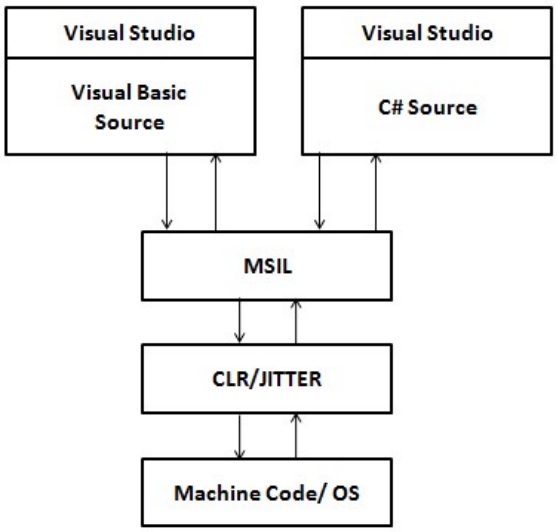


Figure 2 : Following the IL

The Just-In-Time Compiler

- Your code does not stay IL for long, however. It is the PE file, containing the IL that can be distributed and placed with the CLR running on the .NET Framework on any operating system for which the .NET Framework exists, because the IL is platform independent. When you run the IL, however, it is compiled to native code for that platform. Therefore, you are still running native code. The compilation to native code occurs via another tool of the .NET Framework: the Just-In-Time (JIT) compiler.
- With the code compiled, it can run within the Framework and take advantage of low level features such as memory management and security. The compiled code is native code for the CPU on which the .NET Framework is running. A JIT compiler will be available for each platform on which the .NET Framework runs, so you should always be getting native code on any platform running the .NET Framework.
- **Just-in-time compilation** (**JIT**), also known as **dynamic translation**, is a method to improve the runtime performance of computer programs. Historically, computer programs had two modes of runtime operation, either interpreted or static (ahead-of-time) compilation. Interpreted code is translated from a high-level language to a machine code continuously during every execution, whereas statically compiled code is translated into machine code before execution, and only requires this translation once.
- JIT compilers represent a hybrid approach, with translation occurring continuously, as with interpreters, but with caching of translated code to minimize performance degradation. It also offers other advantages over statically compiled code at development time, such as handling of late-bound data types and the ability to enforce security guarantees.
- The Common Language Runtime (CLR) provides various Just In Time compilers (JIT) and each works on a different architecture depending on Operating System. That is why the same Microsoft Intermediate Language (MSIL) can be executed on different Operating Systems without rewrite the source code.

- Just In Time (JIT) compilation preserves memory and save time during initialization of application.
- Just In Time (JIT) compilation is used to run at high speed, after an initial phase of slow interpretation.
- Just In Time Compiler (JIT) code generally offers far better performance than interpreters.
- **There are three types of JIT:**
    1) Pre JIT
    2) Econo JIT
    3) Normal JIT

1) **Pre JIT**: It converts all the code in executable code in a single cycle and it is slow.
2) **Econo JIT:** It will convert the called executable code only. But it will convert code every time when a code is called again.
3) **Normal JIT:** It will only convert the called code and will store in cache so that it will not require converting code again. Normal JIT is fast.

.NET Languages

- .Net languages are CLI computer programming languages that may also optionally use the .NET Framework Base Class Library and which produce programs that execute within the Microsoft .NET Framework. Microsoft provides several such languages, including C#, F#, Visual Basic .NET, and Managed C++.
- Generally .NET languages call into two main categories, TypeSafe Languages (such as C#) and Dynamic Languages (Such as Python). Type Safe Languages are built on the .NET Common Language Runtime and Dynamic Languages are built on top of the .NET Dynamic Language Runtime. The .NET Framework is unique in its ability to provide this flexibility.
- Regardless of which .NET language is used, the output of the language compiler is a representation of the same logic in an intermediate language named Common Intermediate Language (CIL).
- As the program is being executed by the CLR, the CLI code is compiled and cached, just in time, to the machine code appropriate for the architecture on which the program is running. This last compilation step is usually performed by the Common Language Runtime component of the framework "just in time" (JIT) at the moment the program is first invoked, though it can be manually performed at an earlier stage.

Microsoft Intermediate Language (MSIL)

- MSIL or IL(Intermediate Language) is machine independent code generated by .NET framework after the compilation of program written in any language by user.
- MSIL or IL is now known as CIL(Common Intermediate Language).
- One of the more interesting aspects of .NET is that when you compile your code, you do not compile to native code. But the compilation process translates your code into something called Microsoft intermediate language, which is also called MSIL or just IL.
- The compiler also creates the necessary metadata and compiles it into the component. This IL is CPU independent. After the IL and metadata are in a file, this compiled file is called the PE, which stands for either *portable executable* or *physical executable*. Because the PE contains your IL and metadata, it is therefore self-describing, eliminating the need for a type library or interfaces specified with the Interface.

.NET Assembly

- Whatever .NET language you create applications with, compilers generate an *assembly*, which is a file containing .NET executable code and is composed essentially by two kinds of elements: MSIL code and metadata.
- The .NET assembly is the standard for components developed with the Microsoft.NET. Dot NET assemblies may or may not be executable, i.e., they might exist as the executable (.exe) file or dynamic link library (DLL) file.
- All the .NET assemblies contain the definition of types, versioning information for the type, meta-data, and manifest. The designers of .NET have worked a lot on the component (assembly) resolution.

The structure of an assembly: Assemblies contain code that is executed by the Common Language Runtime.

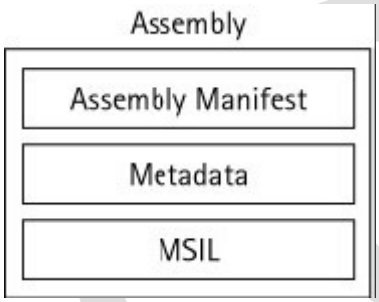


Figure 3 : A diagram of assembly

- Assemblies are made up of the following parts:
  a) The assembly manifest
  b) Type metadata
  c) Microsoft Intermediate Language (MSIL) code
- The assembly manifest is where the details of the assembly are stored. The assembly is stored within the DLL or EXE itself. Assemblies can either be single or multiple file assemblies and, therefore, assembly manifests can either be stored in the assembly or as a separate file. The assembly manifest also stores the version number of the assembly to ensure that the application always uses the correct version.
- The metadata contains information on the types that are exposed by the assembly such as security permission information, class and interface information, and other assembly information.

Contents of an Assembly:

a) Assembly Manifest
b) Assembly Name
c) Version Information
d) Types
e) Cryptographic Hash
f) Security Permissions

An assembly does the following functions:

- It contains the code that the runtime executes.
- It forms a security boundary. An assembly is the unit at which permissions are requested and granted.
- It forms a type boundary. Every type's identity includes the name of the assembly at which it resides.

- It forms a reference scope boundary. The assembly's manifest contains assembly metadata that is used for resolving types and satisfying resource requests.
- It forms a version boundary. The assembly is the smallest version able unit in the common language runtime; all types and resources in the same assembly are versioned as a unit.
- It forms a deployment unit. When an application starts, only the assemblies the application initially calls must be present. Other assemblies, such as localization resources or assemblies containing utility classes, can be retrieved on demand. This allows applications to be kept simple and thin when first downloaded.
- It is a unit where side-by-side execution is supported.

There are two kinds of assemblies in .NET

a) Private
b) Shared

a) Private assemblies is the assembly which is used by application only, normally it resides in your application folder directory.

b) Shared assemblies - It resides in GAC, so that anyone can use this assembly. Public assemblies are always share the common functionalities with other applications.

- An assembly can be a single file or it may consist of the multiple files. In case of multi-file, there is one master module containing the manifest while other assemblies exist as non-manifest modules. A module in .NET is a sub part of a multi-file .NET assembly. Assembly is one of the most interesting and extremely useful areas of .NET architecture along with reflections and attributes, but unfortunately very few people take interest in learning such theoretical looking topics.

The .NET Framework Namespaces

- . Net framework class library is a collection of namespaces.
- Namespace is a logical naming scheme for types that have related functionality.
- Namespace means nothing but a logical container or partition.
- For example: My computer contains C:, D:, E: and F: Each drive contains 1.txt file. The file 1.txt is available in all the drive so it is require to specify the drive name to locate the actual required file.
- At the top of the hierarchy is the System namespace.
- A namespace is just a grouping of related classes. It's a method of putting classes inside a container so that they can be clearly distinguished from other classes with the same name.
- A namespace is a logical grouping rather than a physical grouping. The physical grouping is accomplished by an assembly
- The .NET CLR consists of multiple namespaces, which are spread across many assemblies. For example, ADO.NET is the set of classes located in the System.Data namespace, and ASP.NET is the set of classes located in the System.Web namespace. In the CLR, the classes and structures contained in each of the namespaces represent a common theme of development responsibility.
- .NET Framework class library is collection of namespaces.
- Following table shows Common Namespaces supported by .NET

System	Contains fundamental classes and base classes.
System.IO	Contains classes for reading and writing data in file.
System.XML	Contains classes work with XML.
System.Windows.Forms	Contains classes for windows-based applications.
System.Data	Contains classes for the database connection.

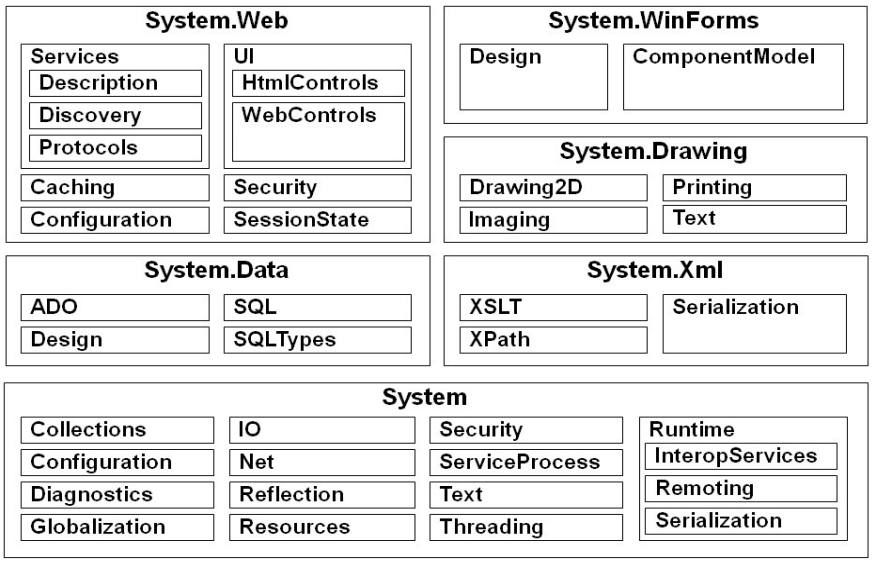


Figure 4: .Net namespaces

VB.NET - Introduction

- Microsoft .NET is a software component that runs on the Windows operating system.
- .NET provides tools and libraries that enable developers to create Windows software much faster and easier. .NET benefits end-users by providing applications of higher capability, quality and security.
- This is how Microsoft describes it: ".NET is the Microsoft Web services strategy to connect information, people, systems, and devices through software. Integrated across the Microsoft platform, .NET technology provides the ability to quickly build, deploy, manage, and use connected, security-enhanced solutions with Web services.
- VB .NET is an object-oriented computer programming language that can be viewed as an evolution of the classic Visual Basic (VB), which is implemented on the .NET Framework.
- Visual Basic 2008 version 9.0 was released together with the Microsoft .NET Framework 3.5

Why .NET?

- Interoperability between language and execution environment.
- Uniformity in schema or formats for Data exchange using XML, XSL (Extensible Style Sheet Language)
- Extend or use existing code that is valid.
- Programming complexity of environment is reduced
- Multiplatform applications, automatic resource management simplification of application deployment.
- It provides security like – code authenticity check, resources access authorizations, declarative and imperative security and cryptographic security methods for embedding into user's application.
- The .Net platform is on integral component a new and simplified model for programming and deploying application on the windows platform.
- .Net development framework provides a new and simplified model for programming and deploying applications on the windows platform.

Compilation and Execution

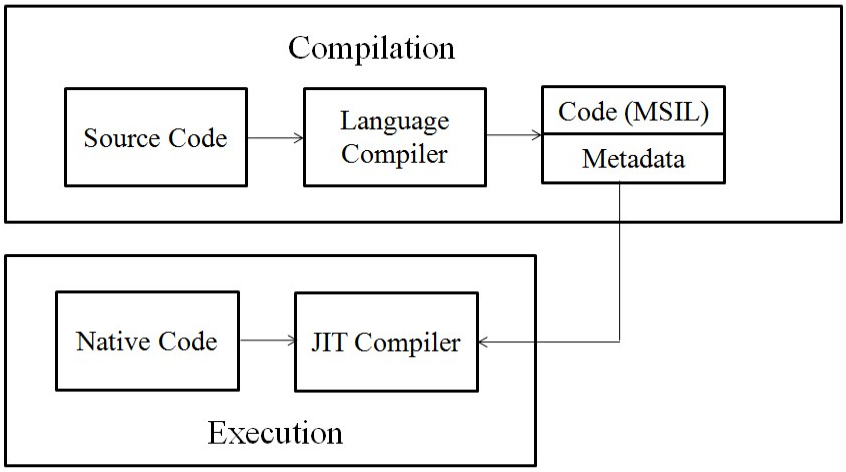


Figure 5: Compilation and Execution Process

Solutions and Projects

- In VB.Net project groups are known as solutions.
- By default, when you create a new project in VB.Net, then visual basic creates a new solution first and then adds a project to that solution.

File Extensions used in VB.Net

- When you save a solution, the file extension is ".sln" and all projects in the solution are saved with extension ".vbproj".
- Most popular file extension is ".vb"

Types of projects:

The following list provides a comparison of Visual Basic 6.0 and Visual Basic .NET project types.

Visual Basic 6.0	Visual Basic .NET
Standard EXE	Windows Application
ActiveX DLL	Class Library
ActiveX EXE	Class Library
ActiveX Control	Windows Control Library
ActiveX Document	No equivalent. Visual Basic .NET can interoperate with ActiveX Documents.
DHTML Application	No equivalent. Use ASP.NET Web Application.
IIS Application (Web Class)	No equivalent. Use ASP.NET Web Application.

The Visual Basic projects you can create are as follows:

- **Windows Application** Windows standard thick client applications based on forms (EXE)
- **Class Library** For individual classes or collections of classes (DLL)
- **Windows Control Library** Controls and components for Windows Forms (classic)
- **. ASP.NET Web Application** ASP.NET−based application composed of static or dynamic HTML pages

- **ASP.NET Web Service** For Web services to be used by clients communicating over the HTTP protocol
- **Web Control Library** Web−based controls for ASP.NET applications
- **Console Application** Your standard Console application
- **Windows Service** Create Windows services
- **Empty Project** Empty Windows application project
- **Empty Web Project** Empty Web server−based application

Visual Basic Integrated Development Environment:

Start Page

- User can use the start page to select from recent projects
- By default 'Get Started' item is selected in the start page.
- User can create new project or open existing project from recent project item.

Toolbars

- This feature is another handy aspect of the IDE.
- These appear near the top of the IDE.
- IDE displays tool tips, it becomes easy to know which button performs which operation.
- Toolbars provides a quick way to select menu item.

Graphical Designer

- VB.Net can display those elements which will look like at run time.
- Different types of graphical designers including.
    a) windows form designers
    b) web form designers
    c) compact designers
    d) XML designers
- From tools menu → select options → options dialog box will open from that select "Window Form Designer" folder display possible options.

The Object Explorer

- This tool lets you look at all the members of an object at once
- The Object Explorer helps open up any mysterious objects that Visual Basic has added to your code so you can see what's going on inside.
- To open the Object Explorer, select View →Other Windows →Object Explorer
- The Object Explorer shows all the objects in your program and gives you access to what's going on in all of them.

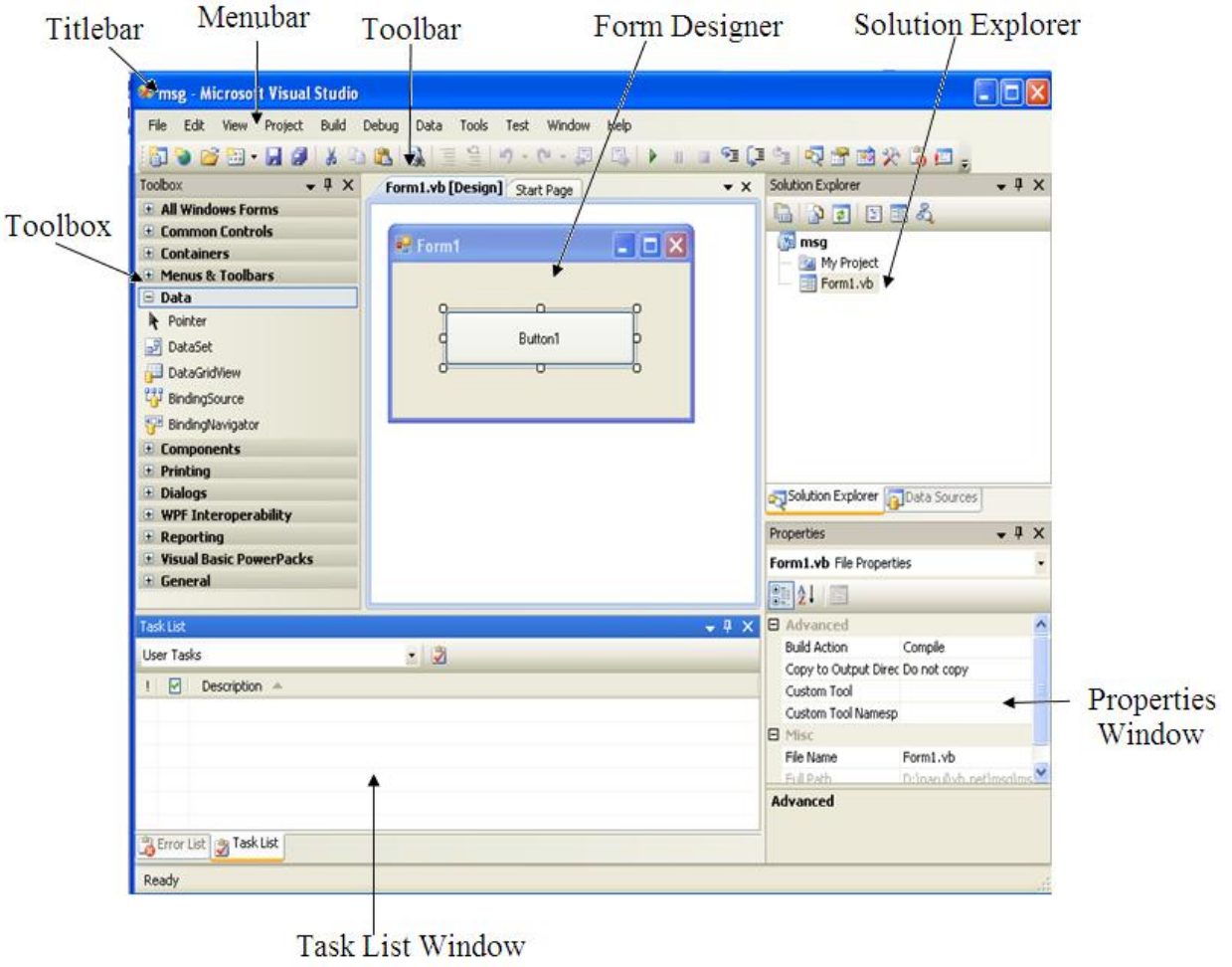


Figure 6: VB .NET IDE

The Menu

- Visual Studio .NET's menu is dynamic, meaning that items will be added or removed depending on what you are trying to do. The menu bar consist of the following options:

File: With this menu you can create a new project, open existing one, save the current project ,exit form the vb.net etc

Edit: The Edit menu provides access to the items you would expect: Undo, Redo, Cut, Copy, Paste, and Delete.

View: The View menu provides quick access to the windows that make up the IDE, such as the Solution Explorer, Properties window, Output window, Toolbox, etc.

Project: The Project menu allows you to add various extra files to your application.

Build: The Build menu becomes important when you have completed your application and want to be able to run it without the use of the Visual Basic .NET environment.

Debug: The Debug menu allows you to start and stop running your application within the Visual Basic .NET IDE. It also gives you access to the Visual Studio .NET **debugger**.

Data: The Data menu helps you use information that comes from a database. It only appears when you are working with the visual part of your application ,not when you are writing code.

Format: The Format menu also only appears when you are working with the visual part of your application. Items on the Format menu allow you to manipulate how the windows you create will appear to the users of your application.

Tools : The Tools menu has commands to configure the Visual Studio .NET IDE, as well as links to other external tools that may have been installed.
Window: The commands on this menu allow you to change the physical layout of the windows in the IDE.
Help: The Help menu provides access to the Visual Studio .NET documentation.

Code Designers

- You can use the tabs at the top center of the IDE to switch between graphical designer and code designer
- From view menu, you can also switch between by using code (F7) and Designer (Shift+F7) items.
- From solution Explorer, from the left side you can use top two buttons.
- At the top of code designer two drop down list boxes are available. The two drop-down list boxes at the top of the code designer; the one on the left lets you select what object's code you're working with, and the one on the right lets you select the part of the code that you want to work on, letting you select between the declarations area, functions, Sub procedures, and methods.

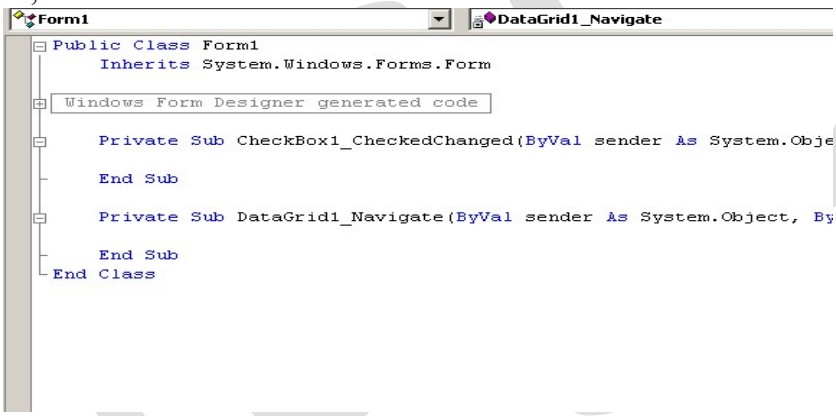


Figure 7: Code Designer Window

IntelliSense

- One useful feature of VB .NET code designers is Microsoft's *IntelliSense*. IntelliSense are those boxes that open as you write your code, listing all the possible options and even completing your typing for user.
- IntelliSense is made up of a number of options, including:
- *List Members*-Lists the members of an object.
- *Parameter Info*-Lists the arguments of procedure calls.
- *Quick Info*-Displays information in tool tips as the mouse rests on elements in your code.
- *Complete Word*-Completes typed words.
- *Automatic Brace Matching*-Adds parentheses or braces as needed.
- you can turn various parts of IntelliSense off if you want; just select the Tools → Options menu item, then select the Text Editor folder, then the Basic subfolder, and finally the General item in the Basic subfolder.

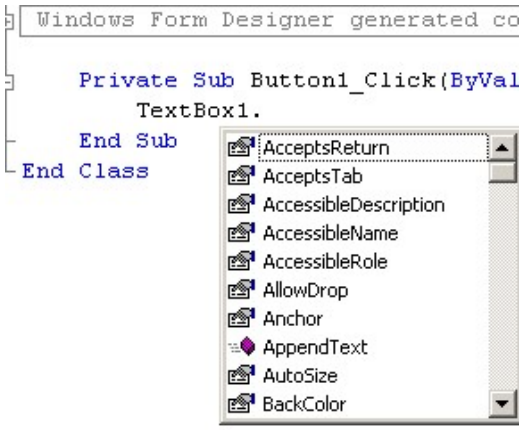


Figure 8 : Intellisense

The Toolbox

- It is available on left side of IDE.
- It uses tabs to divide its contents into categories like marked Data, Components, Windows Forms, and General.
- The Data, Components, Windows Forms, and General tabs appear when you're working with a Windows form in a Windows form designer, but when you switch to a code designer in the same project, all you'll see are General and Clipboard Ring in the toolbox. When you're working on a Web form, you'll see Data, Web Forms, Components, Components, HTML, Clipboard Ring, and General, and so on.
- The Data tab displays tools for creating datasets and making data connections.
- The Windows Forms tab displays tools for adding controls to Windows forms and so on.
- The General tab is empty by default, and is a place to store general components, controls, and fragments of code in.

The Solution Explorer

- It is available on right top corner of IDE
- This tool displays a hierarchy-with the solution at the top of the hierarchy, the projects one step down in the hierarchy, and the items in each project as the next step down.
- You can set the properties of various items in a project by selecting them in the Solution Explorer and then setting their properties in the properties window. And you can set properties of solutions and projects by right-clicking them and selecting the Properties item in the menu that appears, or you can select an item and click the properties button, which is the right-most button at the top of the Solutions Explorer.
- User can switch between graphical and code designers by using the buttons that appear at top left in the Solution Explorer
- You can right-click a solution and add a new project to it by selecting the Add→New Project menu item in the popup menu that appears. And you can specify which of multiple projects runs first-that is, is the startup project or projects-by right-clicking the project and selecting the Set As Startup Object item, or by right-clicking the solution and selecting the Set Startup Projects item.
- With this tool user can also add new item
- By clicking on see all button, solution explorer will shows all files available with current project.
- Refresh button is also available.

The Class View Window

- If you click the Class View tab under the Solution Explorer, you'll see the Class View window, This view presents solutions and projects in terms of the classes they contain, and the members of these classes.
- Using the Class View window gives you an easy way of jumping to a member of class that you want to access quickly-just find it in the Class View window, and double-click it to bring it up in a code designer.

The Properties Window

- The Properties window is divided into two columns of text, with the properties on the left, and their settings on the right.
- From drop-down list box at top of properties window, user can select any object which is available in current form.
- When you select a property, Visual Basic will give you an explanation of the property in the panel at the bottom of the Properties window. And you can display the properties alphabetically by clicking the second button from the left at the top of the Properties window, or in categories by clicking the left-most button.

The Dynamic Help Window

- The window that shares the Properties window's space, however, is quite new-the Dynamic Help window. Visual Basic .NET includes the usual Help menu with Contents, Index, and Search items, of course, but it also now supports dynamic help, which looks things up for you automatically. You can see the Dynamic Help window by clicking the Dynamic Help tab under the Properties window.
- VB .NET looks up all kinds of help topics on the element you've selected automatically; for example, I've selected a button on a Windows form, and dynamic help has responded by displaying all kinds of helpful links to information on buttons.

The Server Explorer

- It is used to explore what's going on in a server
- Using this tool you can drag and drop whole items onto windows forms from server explorer. Ex. Database.

The Output Window

- At the bottom of the IDE, two tabs are available one is Output and other is Breakpoints windows.
- From View menu → Other Window → Select Output Window.
- This window displays results of building and running programs.
- Using system Diagnostic.Debug.Write method user can send output to output window.
- Ex. System.Diagnostics.Debug.Write("Hello")

The Task List

- It is display from View → Show Tasks → All
- The Task List displays tasks that VB .NET assumes you still have to take care of, and when you click a task, the corresponding location in a code designer appears.

The Command Window

- It is display from View → Other Windows → Command Window
- It opens the Command window
- This window is a little like the Immediate window in VB6, because you can enter commands like **File.AddNewProject** here and VB .NET will display the Add New Project dialog box. However, this window is not exactly like the Immediate window, because you can't enter Visual Basic code and have it executed.

Object Explorer Window

- The object explorer window allows us to view all the members of an object at once. It lists all the objects in our code and gives us access to them. The image below displays an object explorer window. You can view the object explorer window by selecting View->Other Windows-> Object Browser from the main menu.

Data Types:

- The following are the data type supported by VB.Net .
- Numeric Data Type(Short, Integer, Long, Single, Double, Decimal)
- Character Data Type (Char, String)
- Miscellaneous Data Type(Boolean , Byte, Date, Object)

Data Type	Sample value	Range of values
String	"hello there"	Alphanumeric characters, digits and other characters
Char	'a'C	Single characters the C stands for character
Integer	-5	Larger numbers ranging from -2147483648 to 2147483647
Long	18459038	Very Large whole numbers
Boolean	True	**True** or **False**
Byte	10	small numbers range 0 to 255
Short	1234	medium numbers ranging from -32,768 to 32,767
Single	5.678	Single-precision floating point numbers with six digits of accuracy.
Double	-2.4585E30	Double-precision floating point numbers with 14 digits of accuracy
Decimal	10.50	Decimal fractions, such as dollars and cents.
Date	#5/23/2002#	month, day, year, various date formats available
Object	frm as Form	Objects represent a group of many values

Following table shows storage size in memory for the data type.

Type	Storage Size
String	2 bytes
Char	2 bytes
Integer	4 bytes
Long	8 bytes
Boolean	2 bytes
Byte	1 byte
Short	2 bytes
Single	4 bytes
Double	8 bytes
Decimal	16 bytes
Date	8 bytes
Object	4 bytes

Variables:

- A variable is something that is used in a program to store data in memory.
- A variable has a name and a data type which determine the kind of data the variable can store.

Variable declaration: Dim statement is used to declare a variable.
Syntax: Dim variablename [([subscript])] [As [New] datatype

Variablename : It is required. It specifies the name of variable which user wants to create.
Subscript : It is optional. Subscript is used to specify the size of array when user declares an array.

New: New keyboard enables creation of new object. If you use new when declaring the object variable, a new instance of the object is created.
Type : The type specify the data type of the variables.
Ex: Dim a as integer, s1 as string

Variable name should follow the following rules:

- Begin with a letter or _.
- Must contain atleast one numeric digit or alphabetic character.
- Maximun 1023 characters are allowed.
- It must be unique in its scope.

- In VB.NET each variable contains default value depends on its data type.
- Default value for Numeric and Byte data type is 0(zero).
- Default value for Char data type is Binary 0(zero).
- Default value for all reference types like object, string, and arrays is Nothing.
- Default value for Boolean data type is False.
- Default value for Date data type is 12:00 AM of 1,1,0001.

Constant Declaration:

Declares and defines one or more constants.

Syntax: Const constantlist

Each *constant* has the following syntax and parts:

constantname[As *datatype*] = *initializer*

***constantlist* :** Required. List of constants being declared in this statement. *Constant* [, *constant* ...]
Constantname: It is required. It specifies the name of the constant.

Datatype: It specifies the type of constant.

Initialize: it is required. The value which is assigned to the constant. Once you initialize the constant variable with a value it can never change.

Ex: Const pi as double = 3.24

Operators

- Visual Basic comes with many built-in operators that allow us to manipulate data. An operator performs a function on one or more operands. For example, we add two variables with the "+" addition operator and store the result in a third variable with the "=" assignment operator like this: int x + int y = int z. The two variables (x ,y) are called operands. There are different types of operators in Visual Basic and they are described below in the order of their precedence.
- Operators may me Unary or Binary
- Unary used with a single operand for example Ans= -10
- Binary used with two operands for example Ans= 10 / 5
- Operators also categorized in following categories:

Arithmetic Operators : Arithmetic operators are used to perform arithmetic operations that involve calculation of numeric values. The table below summarizes them:

Operator	Use
^	Exponentiation
-	Negation (used to reverse the sign of the given value, exp -intValue)
*	Multiplication
/	Division for ex a=11/5 then answer is 5.5
\	Integer Division for ex a=11\5 then answer is 5
Mod	Modulus Arithmetic
+	Addition
-	Subtraction

Concatenation Operators : Concatenation operators join multiple strings into a single string. There are two concatenation operators, + and & as summarized below:

Operator	Use
+	**String Concatenation**
&	**String Concatenation**

Comparison Operators : A comparison operator compares operands and returns a logical value based on whether the comparison is true or not. The table below summarizes them:

Operator	Use
=	Equality
<>	Inequality
<	Less than
>	Greater than
>=	Greater than or equal to
<=	Less than or equal to

Logical / Bitwise Operators : The logical operators compare Boolean expressions and return a Boolean result. In short, logical operators are expressions which return a true or false result over a conditional expression. The table below summarizes them:

Operator	Use
Not	Negation

And	Conjunction
AndAlso	Conjunction
Or	Disjunction
OrElse	Disjunction
Xor	Disjunction

Type Conversion functions

- Type conversion is used for convert one data type to another
- There are two type of conversion : **Implicit and Explicit.**
- An **Implicit Conversion** does not require any special syntax in the source code.
- An **Explicit Conversion** requires function.
- For Example : Implicit Conversion

```
Dim a As Integer
Dim b As Double
a = 4499
b = a
```

- An explicit conversion requires function.

Function Name	Convert Into
CBool	Boolean
CByte	Byte
CChar	Char
CDate	Date
CDbl or Val	Double
CDec	Decimal
CInt	Integer
CLng	Long
CObj	Object
CShort	Short
CSng	Single
CStr	String

- Following is common syntax for each type conversion function:

Syntax : Function_Name(argument)

For Example:

```
Dim str As String
Dim no As Integer
str = "5"
no = Cint(str)
```

CTYPE function

- It uses to convert one type to another type.
- Instead of remember all conversion functions , we can use CTYPE function
- Execution is faster .

Syntax : Ctype(expression,Type name)

For Example :

```
Dim no1 As Integer
Dim no2 As Double
no2 = 66.77
no1 = Ctype(no2,Integer)
```

Boxing and Unboxing:

- Boxing and unboxing act like bridges between value type and reference types. When we convert
  value type to a reference type it's termed as boxing. Unboxing is just vice-versa.
- Boxing: The conversion of a value type instance to an object.
- Unboxing : The conversion of an object instance to a value type.
- Example: Dim no As Integer= 10
  Dim obj As Object = no ---- Boxing
  Dim ans As Integer = CInt(obj) ---- Unboxing

Boxing conversions.

A boxing conversion permits any value-type to be implicitly converted to the type object or to any interface-type implemented by the value-type.
Boxing a value of a value-type consists of allocating an object instance and copying the value-type value into that instance.
For example any value-type G, the boxing class would be declared as follows:

```
Class vBox
Private value As G
Sub New(ByVal g As G)
value = g
End Sub 'New
End Class
```

BoxBoxing of a value v of type G now consists of executing the expression new G_Box(v), and returning the resulting instance as a value of type object.
Thus, the statements

```
Dim i As Integer = 12
Dim box As Object = i
```

conceptually correspond to

```
Dim i As Integer = 12
Dim box = New int_Box(i)
```

Boxing classes like G_Box and int_Box above don't actually exist and the dynamic type of a boxed value isn't actually a class type. Instead, a boxed value of type G has the dynamic type G, and a dynamic type check using these operator can simply reference type G. For example,

```
Dim i As Integer = 12
Dim box As Object = i
If TypeOf box Is Integer Then
    Console.Write("Box contains an int")
End If
```

will output the string "Box contains an integer" on the console.

Unboxing conversions.

An unboxing conversion permits an explicit conversion from type object to any value-type or from any interface-type to any value-type that implements the interface-type. An unboxing operation consists of first checking that the object instance is a boxed value of the given value-type, and then copying the value out of the instance.
Unboxing conversion of an object box to a value-type G consists of executing the expression ((G_Box)box).value.
Thus, the statements

```
Dim box As Object = 12
Dim i As Integer = CInt(box)
```

conceptually correspond to

```
Dim box = New int_Box(12)
Dim i As Integer = CType(box, int_Box).value.
```

For an unboxing conversion to a given value-type to succeed at run-time, the value of the source argument must be a reference to an object that was previously created by boxing a value of that value-type. If the source argument is null or a reference to an incompatible object, an InvalidCastException is thrown.

Array:

- An ordered collection of same type of data having single variable name is known as array. Each element of the array can be referenced by a numerical subscript.
- In VB.Net two types of arrays are:
  a) Standard Array
  b) Dynamic Array

Declaration Of Standard Array:
Syntax : Dim varname [(subscripts)] [As type]

WithEvents: This keyword is valid only in class modules. This keyword specifies that varname is an object variable used to respond to events triggered by an ActiveX object.
VarName : The Varname specify the name of variable which you want to create.
Subscript : Subscript is used when you declare an array.
Type : The type specify the data type of the array variables. User can also include "To" keyword in array declaration.
Ex : Dim n(5) As Integer.
Dim s(3) As String.
Dim n(1 to 4) As Integer.

Dynamic Array:

- In any C or C++ programming language user can not modify the size of the array. Once we declared the size of array then it becomes fixed.
- In VB .NET we can increase the size of array.
- In some cases we may not know exactly the size of array at declaration time. We may need to change the size of the array at runtime. So we can resize the array at any time by using Redim statement.
- But with dynamic array we cannot change the dimension of the array.

Redim Statement: It reallocates storage space for array variables.
Syntax: ReDim [**Preserve**] *name***(**boundlist**) [,** *name***(**boundlist**) …]**
Preserve: It is optional. It is used to preserve the data in an existing array when user changes the size of the last dimension.

Name: The name of the array variable.
Boundlist: It is required. It is dimensions of an array variable.
Example:

```
Dim a() As Integer
Private Sub cmdInput_Click()
   ReDim a(5) As Integer
   For i = LBound(a) To UBound(a)
      a(i) = InputBox("Enter elements:")
   Next
End Sub
```

```
Private Sub cmdPrint_Click()
   ReDim Preserve a(3) As Integer
   'MsgBox LBound(a) & UBound(a)
    For i = LBound(a) To UBound(a)
      msgbox a(i)
   Next
End Sub
```

- ## String Functions:

1) Len: This function returns an integer containing the number of characters in a given string.

Syntax : Len(string)

String: Any valid string expression. If *string* contains Null, Null is returned.

Example:

```
S1=”hello”
Msgbox len(s1)
```

2) Mid: It returns a **Variant** (**String**) containing a specified number of characters from a string.

Syntax: **Mid**(***string***, ***start***[, ***length***])

The **Mid** function syntax has these named arguments:

Part	Description
String	Required. String expression from which characters are returned. If ***string*** contains Null, **Null** is returned.
Start	Required; Long. Character position in ***string*** at which the part to be taken begins. If ***start*** is greater than the number of characters in ***string***, **Mid** returns a zero-length string ("").
Length	Optional; **Variant** (**Long**). Number of characters to return. If omitted or if there are fewer than ***length*** characters in the text (including the character at ***start***), all characters from the ***start*** position to the end of the string are returned.

Examples:

```
Dim MyString, FirstWord, LastWord
MyString = "Mid Function Demo"   ' Create text string.
FirstWord = Mid(MyString, 1, 3)   ' Returns "Mid".
LastWord = Mid(MyString, 14, 4)   ' Returns "Demo".
```

3) Trim , Rtrim, Ltrim: It Returns a string that contains a copy of a specified string without leading spaces (**LTrim**), without trailing spaces (**RTrim**), or without leading or trailing spaces (**Trim**).

Syntax: **Trim / Rtrim / Ltrim (string)**

String: It is requires any valid **String** expression. If *string* equals **Nothing**, the function returns an empty string.

Example:

```
S1= “   This   is test   “
Msgbox Trim(s1)
Msgbox Rtrim(s1)
Msgbox  Ltrim(s1)
```

4) **Instr:** It returns a **Variant** (**Long**) specifying the position of the first occurrence of one string within another.

Syntax: **InStr**([*start*,]*string1*, *string2*[, *compare*])

The **InStr** function syntax has these arguments:

Part	Description
Start	Optional. Numeric expression that sets the starting position for each search. If omitted, search begins at the first character position. If ***start*** contains Null, an error occurs. The ***start*** argument is required if ***compare*** is specified.
string1	Required. String expression being searched.
string2	Required. String expression sought.
Compare	Optional. Specifies the type of string comparison. If ***compare*** is Null, an error occurs. If ***compare*** is omitted, the **Option Compare** setting determines the type of comparison.

Settings: The *Compare* argument settings are:

Constant	Value	Description
Binary	0	Performs a binary comparison
Text	1	Performs a text comparison

Return Value

If	InStr returns
String1 is zero length or **Nothing**	0
String2 is zero length or **Nothing**	*start*
String2 is not found	0
String2 is found within *String1*	Position where match begins

Examples:

```
Dim SearchString, SearchChar, MyPos
SearchString ="XXpXXpXXPXXP"   ' String to search in.
SearchChar = "P"   ' Search for "P".
' A textual comparison starting at position 4. Returns 6.
MyPos = Instr(4, SearchString, SearchChar, 1)
```

' A binary comparison starting at position 1. Returns 9.
MyPos = **Instr(**1**,** SearchString**,** SearchChar**,** 0**)**
' Comparison is binary by default (last argument is omitted).
MyPos = **Instr(**SearchString**,** SearchChar**)** ' Returns 9.

5) Lcase: It returns a String that has been converted to lowercase.

Syntax: **LCase**(*string*)

The required *string* argument is any valid string expression. If *string* contains Null, Null is returned. Only uppercase letters are converted to lowercase; all lowercase letters and nonletter characters remain unchanged.
Example:
Dim UpperCase, LowerCase
Uppercase = "Hello World 1234" ' String to convert.
Msgbox **Lcase(**UpperCase**)** ' Returns "hello world 1234"

6) Ucase: It returns a **Variant** (**String**) containing the specified string, converted to uppercase.

Syntax: **UCase**(*string*)

The required *string* argument is any valid string expression. If *string* contains Null, **Null** is returned.
Only lowercase letters are converted to uppercase; all uppercase letters and nonletter characters remain unchanged.
Example:
Dim LowerCase, UpperCase
LowerCase = "Hello World 1234" ' String to convert.
Msgbox **UCase(**LowerCase**)** ' Returns "HELLO WORLD 1234".

7) Asc: It returns an Integer representing the character code corresponding to the first letter in a string.

Syntax: **Asc(***string***)**

The required *string* argument is any valid string expression. If *String* is a **String** expression, only the first character of the string is used for input. If *String* is **Nothing** or contains no characters, an error occurs.
Example: msgbox **Asc(**"A"**)**

8) Chr: It returns a String containing the character associated with the specified character code.

Syntax: **Chr(***charcode***)**
An **Integer** expression representing the *code point*, or character code, for the character. If *CharCode* is outside the valid range, an error occur. The valid range for **Chr** is 0 through 255.
Examples:
Dim MyChar
MyChar = **Chr(**65**)** ' Returns A.

9) Space: It returns a string consisting of the specified number of spaces.

Syntax: **Space**(number)

Number : It is required Integer expression. The number of spaces you want in the string.
Example: Msgbox " Hi" & space(5) & " How r u?"

10) Format: This function returns a string formatted according to instructions contained in a format String expression.

Syntax: **Format** (Expression, style)

Expression : it is any valid expression.
Style: it is applied on specified expression
Example:
D1= #02/14/1989#
Msgbox format(d1,"DD-MM-YY")

11) Strcomp: It returns -1, 0, or 1, based on the result of a string comparison.
Syntax: **Strcomp**(string1,string2[,compare])
String1 :Required. Any valid **String** expression.
String2 :Required. Any valid **String** expression.
Compare :Optional. Specifies the type of string comparison. If *Compare* is omitted, the **Option Compare** setting determines the type of comparison.

The *Compare* argument settings are:

Return Value:The **StrComp** function has the following return values.

If	StrComp returns
String1 sorts ahead of *String2*	-1
String1 is equal to *String2*	0
String1 sorts after *String2*	1

Example:

```
Dim TestStr1 As String = "ABCD"
Dim TestStr2 As String = "abcd"
Dim TestComp As Integer
' The two strings sort equally. Returns 0.
TestComp = StrComp(TestStr1, TestStr2, CompareMethod.Text)
' TestStr1 sorts after TestStr2. Returns -1.
TestComp = StrComp(TestStr1, TestStr2, CompareMethod.Binary)
' TestStr2 sorts before TestStr1. Returns 1.
TestComp = StrComp(TestStr2, TestStr1)
```

12) Left: It returns a **Variant** (**String**) containing a specified number of characters from the left side of a string.

Syntax: **Left**(***str***, ***length***)

str :it is required **String** expression from which the rightmost characters are returned.
Length*:* It is required Numeric expression indicating how many characters to return. If 0, a zero-length string ("") is returned. If greater than or equal to the number of characters in *str*, the entire string is returned.

Examples:
Dim AnyString, MyStr
AnyString = "Hello World" ' Define string.
MyStr = Microsoft.VisualBasic.**Left(**AnyString**,** 1**)**
Msgbox Mystr ' Returns "H".

13) Right: It returns a string containing a specified number of characters from the right side of a string.

Syntax: **Right**(str,length)

str *:*it is required **String** expression from which the rightmost characters are returned.
Length: It is required Numeric expression indicating how many characters to return. If 0, a zero-length string ("") is returned. If greater than or equal to the number of characters in *str*, the entire string is returned.

Example:
S1="this is test"
Msgbox Micosoft.VisualBasic.Right(s1,3)

14) Replace: It returns a string in which a specified substring has been replaced with another substring a specified number of times.

Syntax: **Replace (Expression, Find, Replacement)**
Expression :Required. String expression containing substring to replace.
Find :Required. Substring being searched for.
Replacement :Required. Replacement substring.

Example: S1= "Shopping List"

Msgbox Replace(s1, "o","I")

Tostring with its Methods:

1) ToString.Concat: This method is used Concatenates three specified instances of String.

Syntax : **System.String.Concat(str1,str2)**

str1 : Parameter String

str2 : Parameter String

Example:

```
Public Class Form1
   Private Sub Button1_Click(ByVal sender As System.Object, _
           ByVal e As System.EventArgs) Handles Button1.Click
      Dim str1 As String
      Dim str2 As String

      str1 = "Concat() "
      str2 = "Test"
      MsgBox(String.Concat(str1, str2))
   End Sub
End Class
```

2) String.copy : This method creates a new instance of String with the same value as a specified String.

Syntac: System.String.Copy(str)

str : The argument String for Copy method

Example:

```
Public Class Form1
   Private Sub Button1_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles Button1.Click
      Dim str1 As String
      Dim str2 As String

      str1 = "VB.NET Copy() test"
      str2 = String.Copy(str1)
      MsgBox(str2)
   End Sub
End Class
```

3) String.Indexof: It returns the index of the first occurrence of the specified substring.

Syntax: System.String.IndexOf(str)

str - The parameter string to check its occurrences

If the parameter String occurred as a substring in the specified String then it returns position of the first character of the substring. If it does not occur as a substring, -1 is returned.

Example:

```
Public Class Form1
   Private Sub Button1_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles Button1.Click
      Dim str As String
      str = "VB.NET TOP 10 BOOKS"
      MsgBox(str.IndexOf("BOOKS"))
```

```
    End Sub
End Class
```

4) **String.substring:** It returns a new string that is a substring of this string. The substring begins at the specified given index and extended up to the given length.

Syntax: Substring(startIndex,length)

startIndex: The index of the start of the substring.

length: The number of characters in the substring.

Example:

```
Public Class Form1
    Private Sub Button1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button1.Click

        Dim str As String
        Dim retString As String
        str = "This is substring test"
        retString = str.Substring(8, 9)
        MsgBox(retString)

    End Sub
End Class
```

5) **String.format:** VB.NET **String Format** method replace the argument Object into a text equivalent System.Striing.

Syntax: System.Format(format, arg0)

String format : The format String

The format String Syntax is like {indexNumber:formatCharacter}

Object arg0 : The object to be formatted.

Examples:

Currency : ***String.Format("{0:c}", 10)*** will return $10.00

The currency symbol ($) displayed depends on the global locale settings.

Date : ***String.Format("Today's date is {0:D}", DateTime.Now)***

You will get Today's date like : 01 January 2005

Time : ***String.Format("The current time is {0:T}", DateTime.Now)***

You will get Current Time Like : 10:10:12

```
Public Class Form1
   Private Sub Button1_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles Button1.Click
      Dim dNum As Double
      dNum = 32.123456789
      MsgBox("Formated String " & String.Format("{0:n4}", dNum))
   End Sub
End Class
```

6) **String.ToUpper:** This method uses the casing rules of the current culture to convert each character in the current instance to its uppercase equivalent. If a character does not have an uppercase equivalent, it is included unchanged in the returned string.

The ToUpper method is often used to convert a string to uppercase so that it can be used in a case-insensitive comparison.

Syntax: string.ToUpper()

Example:
S1= “This is Test”
Msgbox s1.Toupper()

7) **String.ToLower**: This method does not modify the value of the current instance. Instead, it returns a new string in which all characters in the current instance are converted to lowercase.

Syntax: string.tolower()

Example:
S1= “This is Test”
Msgbox s1.tolower()

8) **String.Remove:** Deletes all the characters from this string beginning at a specified position and continuing through the last position.

Syntax: String.Remove(startIndex)

startIndex : The position to begin deleting characters.

Example:

S1=”abc---def”
Msgbox s1.remove(3)

If…Else Statements

It is used to execute one or more statement conditionally. You can use single line syntax or multi line syntax.

Syntax : 1) If <condition> then <statement>

Syntax : 2) If <condition> then
<statement>
End if

Syntax : 3) If *condition* Then
[*statements*]
[ElseIf *condition-n* Then
[*elseifstatements*] …]
[Else
[*elsestatements*]]
End If

If condition is true, VB .NET executes all the statements following the then keyword.

- You can use either single line or multiple-line syntax to execute just 1 statement.
- You can use comparison operators in the condition to generate a logical result that's true or false.
- If *condition* is **True**, the statements immediately following the **Then** keyword in the body of the **If** statement will be executed, and the **If** statement will terminate before the code in any **ElseIf** or **Else** statement is executed.
- If *condition* is **False**, the following **ElseIf** statements are evaluated, if there are any; this statement lets you test additional conditions, and if any are **True**, the corresponding code (*elseifstatements* above) is executed and the **If** statement terminates. If there are no **ElseIf** statements, or if none of their conditions are **True**, the code in the **Else** statement (*elsestatements* above), if there is one, is executed automatically.

Example Dim a, b, c As Integer

```
a = InputBox("Enter A: ")
b = InputBox("Enter B: ")
c = InputBox("Enter C: ")

If a>b And a>c Then
   MsgBox("A is Maximum")
ElseIf b>a And b>c Then
   MsgBox("B is Maximum")
Else
   MsgBox("C is Maximum")
End If
```

Select Case statement:

Executes one of several groups of statements depending on the value of an expression.

Syntax:

```
Select Case testexpression
    [Case expressionlist-n
        [statements-n]]…
    [Case Else
        [elsestatements]]
End Select
```

- **Testexpression:** Required It may be any numeric or string expression.
- **Expression list_n:** it may be
  - expression or
  - expression To expression
  - Is expression operator expression
- **statements-n :** one or more statements executed if test expression doesn't match any of the case clause.

- If test expression matches any case expression list expression then statement following that clause is executed upto the next case clause or for the last clause or up to End select.
- After performing select……..case, control executes the next statement after the End select.
- If test expression matches more than 1 expression then only the statements which follow the 1st match are executed.
- The Case Else clause is used to indicated the else statements to be executed if no match is found between test expression and expression list.
- Select case statement can be nested. Each nested select case statement must have a matching End select statement.

Example: Dim no1, no2 as integer

```
No2 =20
No1 = val ( Inputbox ("Enter the value of no1:")
Select case no1
        Case 1:
            Msgbox("The value of no1 is:") & no1
        Case 2:
            Msgbox("The value of no1 is:") & no1
        Case 3 To 15:
            Msgbox("The value of no1 is between 3 to 5")
        Case  Is > no2:
            Msgbox  no1 & "is grater than" & no2
        Case else:
            Msgbox ("nothing");
    End select.
```

***With* Statement**

The **With** statement is not a loop, but it can be as useful as a loop. User can use the **With** statement to execute statements using a particular object.

Syntax: With *object*

[*statements*]

End With

Example:

```
With TextBox1
   .Height = 1000
   .Width = 3000
   .Text = "Welcome to Visual Basic"
End With
```

Do Loop:

The Do loop keeps executing its enclosed statements while or until *condition* is true. You can also terminate a Do loop at any time with an Exit Do statement. The Do loop has two versions; you can either evaluate a condition at the beginning:

Syntax:
```
Do [{While | Until} condition ]
   [statements]
   [Exit Do]
   [statements]
Loop
```
OR

Syntax:
```
Do
   [statements]
   [Exit Do]
   [statements]
Loop [{While | Until} condition]
```

- A loop can be executed either while the condition is true or until the condition becomes true.
- When VB executes the loops, if first evaluates the condition if condition is false, the Do…While or Do…Until loop is skipped & execute the statement followed by the loop.
- The Do…Loop can execute any number of times as long as condition is true.
- If the condition is initially false, the statements may never execute.
- Exit Do statement terminates the loop.

Example:

```
i=5                          i=1
Do While i>0                 Do until i<=5
```

```
        Msgbox  i                    me.point i
        i=i-1                        i=i+1
    loop                         loop
```

For Loop

The **For** loop is probably the most popular of all Visual Basic loops. The Do loop doesn't need a *loop index*, but the **For** loop does; a loop index counts the number of loop iterations as the loop executes.

Syntax: For *index = start* To *end* [Step *step*]

[*statements*]

[Exit For]

[*statements*]

Next [*index*]

- The keywords in square brackets are optional.
- The keyword counter, start, end, increment all one of numeric type.
- **Index:** Required Numeric variable. The variable can't be a Boolean or an array element.
- **End:** Required find value of counter.
- **Step:** Optional Counter is changed each time through the loop. If not specified then step defaults to one.
- **Statement:** Optional one or more statement between for and Next that the executed the specified number of times.
- **Exit For:** to terminate the loop without executing the statements after exit for keyword.
- The loop is executed as many times as required for the counter to reach the end value.
- The increment argument can be either positive or negative. If start is greater than end, the value of increment must be negative. If not the loop's body can't be executed.

Example:
```
for i=0 To 10
    msgbox i
Next
```
```
for i=10 To 1 step -1
    msgbox i
Next.
```

If I were to use a **step** size of 2:

```
For i = 0 To n Step 2
   MsgBox(i)
Next
```

For Each…Next Loop

You use the **For Each…Next** loop to loop over elements in an array or a Visual Basic collection. This loop automatically loops over all the elements in the array or collection.

Syntax:
```
For Each element In group
    [statements]
    [Exit For]
    [statements]
Next [element]
```

Example:
```
Dim intIDArray(3), intArrayItem As Integer
intIDArray(0) = 0
intIDArray(1) = 1
intIDArray(2) = 2
intIDArray(3) = 3
For Each intArrayItem In intIDArray
   System.Console.WriteLine(intArrayItem)
Next intArrayItem
```

While Loop

While loops keep looping while the condition they test remains true, so you use a **While** loop if you have a condition that will become false when you want to stop looping.

Syntax:
```
While condition
    [statements]
End While
```

- If condition is true, all the statements are executed & when the wend statement is reached, control is returned to the While statement which evaluate condition again.
- If condition is still true, the process is repeated.
- If condition is false, the program executes the statement following Wend statement.

Example:
```
i=5
While i>=0
    total=total+i
    i=i-1
End while
```

➔ Working with Procedure :--

- Dividing your code into procedure allows you to break it up into more modular units. As your program become longer that's invaluable as it stops everything from becoming too cultured

In visual basic , all executable code must be in procedure

There are two typed of procedure
1) sub procedure (sub routine)
2) Functions

Procedure:

- A procedure is a set of one or more program statements that can be run or call by referring to the procedure name.
- We can divide the big program into small procedures.
- Due to procedure the application is easier to debug and easier to find error.

- The main advantage of procedure is its reusability.
- Procedures are categorized into two categories:
  1) In-built procedure
  2) User-defined procedure
- There are 4 types of procedure in VB.Net.
  1) Sub procedure: It does not return value.
  2) Event handling procedure: These are Sub procedures that execute in response to an event triggered by user action.
  3) Function procedures: they return a value.
  4) Property procedures: used in object oriented concept.

Sub Procedure: Also known as Sub Routine.

- Sub procedure may or may not have arguments. Arguments are not compulsory.
- A sub procedure does not return the value.
- By Val is by default argument type.
- To declare procedure Sub keyword is used.

Sub Statement: Declares the name, parameters, and code that define a **Sub** procedure.

Syntax: [**Private** | **Public**] **Sub** name [**(**parameterlist**)**]
[statements]
[**Exit Sub**]
[statements]
End Sub

- **name :** It specifies name of the procedure.
- **parameterlist :** It is optional. It specifies the list of local variable names representing the parameters of this procedure. Specifies the parameters a procedure expects when it is called.
- Multiple parameters are separated by commas. The following is the syntax for one parameter.
  [optional] [**ByVal** | **ByRef**] varname[**( )**] [**As** type] [**=** defaultvalue]
- **Optional:** Optional. Specifies that this parameter is not required when the procedure is called.
- **ByVal** : It is Optional. Specifies that the procedure cannot replace or reassign the variable element underlying the corresponding argument in the calling code.
- **ByRef** : It is Optional. Specifies that the procedure can modify the underlying variable element in the calling code the same way the calling code itself can.
- **parametername :** It is Required. Name of the local variable representing the parameter.

- **parametertype :** Required if **Option Strict** is **On**. Data type of the local variable representing the parameter.
- **defaultvalue :** Required for **Optional** parameters. Any constant or constant expression that evaluates to the data type of the parameter. If the type is **Object**, or a class, interface, array, or structure, the default value can only be **Nothing**.

Example:

```
Dim ans As Double
Private Sub sMsg()
   MsgBox("Hello")
End Sub
Private Sub sum1(ByRef n1 As Integer)
   MsgBox(n1 + 20)
End Sub
Private Sub butOk_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butOk.Click
    Call sMsg()
    Call sum1(Val(Me.txtN1.Text))
End Sub
```

Working with Modules:-

- Visual Basic uses the term module level to apply equally to modules,classes and structures you can declare elements at this level by placing the declaration statement out side of any procedure or block within the module class or structure .
- When you make a declaration at the module level the accessibility you choose determines the scope. The namespace that contains the Module , class or Structure also affects the scope
- Elements for which you declare private accessibility are available for reference to every procedure in that module . but , not to any code in a different module

The Dim statement at Module level defaults to Private accessibility . o , it is equivalent to using the private statement. However , you can make the scope and accessibility more obvious by using private.

Class :--

- A class means collection of methods/functions. Method/function accepts parameters, process set of codes which you have written in the module/function and returns the output to the caller. Collection of class is called Class Library. When you complie the Class Library it becomes a DLL.

- A class is simply an abstract model used to define new data types. A class may contain any combination of encapsulated data (fields or member variables), operations that can be performed on the data (methods) and

accessors to data (properties). For example, there is a class String in the System namespace of .Net Framework Class Library (FCL). This class contains an array of characters (data) and provide different operations (methods) that can be applied to its data like *ToLowerCase()*, *Trim()*, *Substring()*, etc. It also has some properties like *Length* (used to find the length of the string).

- A class in VB.Net is declared using the keyword Class and its members are enclosed with the *End Class* marker

❖ **NameSpace:** If you declared an element at module level using the friend or public statement it becomes available to all procedures throughout the entire namespace in which it is declared.
Not that , an element accessible in a namespace is also accessible from inside any namespace nested inside that namespace .
A namespace is used in .NET Framework to define the scope of a set of related object. The namespace scope lets you organize code and gives you a way to create globally unique types. Whether or not you explicitly declare a namespace in a source file, the compiler adds a default namespace. This unnamed namespace, sometimes referred to as the global namespace, is present in every file.

Namespaces organize the objects defined in an assembly. Assemblies can contain multiple namespaces, which can in turn contain other namespaces. Namespaces prevent ambiguity and simplify references when using large groups of objects such as class libraries.

For example, the .NET Framework defines the ListBox class in the System.Windows.Forms namespace. The following code fragment shows how to declare a variable using the fully qualified name for this class:

Ex: Dim LBox As System.Windows.Forms.ListBox

Introduction of Windows Forms :--

Windows Forms is the new platform for Microsoft Windows application development, based on the .NET Framework. This framework provides a clear, object-oriented, extensible set of classes that enable you to develop rich Windows applications. Additionally, Windows Forms can act as the local user interface in a multi-tier distributed solution.

A form is a bit of screen real estate, usually rectangular, that you can use to present information to the user and to accept input from the user. Forms can be standard windows, multiple document interface (MDI) windows, dialog boxes, or display surfaces for graphical routines. The easiest way to define the user interface for a form is to place controls on its surface. Forms are objects that expose properties which define their appearance, methods which define their

behavior, and events which define their interaction with the user. By setting the properties of the form and writing code to respond to its events, you customize the object to meet the requirements of your application.

As with all objects in the .NET Framework, forms are instances of classes. The form you create with the Windows Forms Designer is a class, and when you display an instance of the form at run time, this class is the template used to create the form. The framework also allows you to inherit from existing forms to add functionality or modify existing behavior. When you add a form to your project, you can choose whether it inherits from the **Form** class provided by the framework, or from a form you have previously created.

Form Life Cycle

- **Move:** This event occurs when the form is moved. Although by default, when a form is instantiated and launched, the user does not move it, yet this event is triggered before the Load event occurs.
- **Load:** This event occurs before a form is displayed for the first time.
- **VisibleChanged:** This event occurs when the Visible property value changes.
- **Activated:** This event occurs when the form is activated in code or by the user.
- **Shown:** This event occurs whenever the form is first displayed.
- **Paint:** This event occurs when the control is redrawn.
- **Deactivate:** This event occurs when the form loses focus and is not the active form.
- **Closing:** This event occurs when the form is closing.
- **Closed:** This event occurs when the form is being closed.

MsgBox Function

Syntax: MsgBox(*Prompt* [, *Buttons* As MsgBoxStyle = MsgBoxStyle.OKOnly [, *Title*])

- ***Prompt***—A string expression displayed as the message in the dialog box. The maximum length is about 1,024 characters (depending on the width of the characters used).
- ***Buttons***—The sum of values specifying the number and type of buttons to display, the icon style to use, the identity of the default button, and the modality of the message box. If you omit *Buttons*, the default value is zero. See below.
- ***Title***—String expression displayed in the title bar of the dialog box. Note that if you omit *Title*, the application name is placed in the title bar.

You can find the possible constants to use for the *Buttons* argument in Table

MsgBox constants.

Constant	Value	Description
OKOnly	0	Shows OK button only.
OKCancel	1	Shows OK and Cancel buttons.
AbortRetryIgnore	2	Shows Abort, Retry, and Ignore buttons.
YesNoCancel	3	Shows Yes, No, and Cancel buttons.
YesNo	4	Shows Yes and No buttons.
RetryCancel	5	Shows Retry and Cancel buttons.
Critical	16	Shows Critical Message icon.
Question	32	Shows Warning Query icon.
Exclamation	48	Shows Warning Message icon.
Information	64	Shows Information Message icon.
DefaultButton1	0	First button is default.
DefaultButton2	256	Second button is default.
DefaultButton3	512	Third button is default.
ApplicationModal	0	Application modal, which means the user must respond to the message box before continuing work in the current application.
SystemModal	4096	System modal, which means all applications are unavailable until the user dismisses the message box.
MsgBoxSetForeground	65536	Specifies the message box window as the foreground window.
MsgBoxRight	524288	Text will be right-aligned.

MsgBox constants.

Constant	Value	Description
MsgBoxRtlReading	1048576	Specifies text should appear as right-to-left on RTL systems such as Hebrew and Arabic.

Example:

```
Private Sub Button1_Click
   Dim Result As Integer
   Result = MsgBox("This is a message box!", MsgBoxStyle.OKCancel +
      MsgBoxStyle.Information + MsgBoxStyle.SystemModal, "Message Box")
   End If
End Sub
```

InputBox Function

User can use the **InputBox** function to get a string of text from the user.

Syntax : InputBox(Prompt [, Title [, DefaultResponseg[,XPos [, YPos]]]])

- **Prompt—** A string expression displayed as the message in the dialog box. The maximum length is about 1,024 characters (depending on the width of the characters used).
- **Title—** String expression displayed in the title bar of the dialog box. Note that if you omit **Title**, the application name is placed in the title bar.
- **DefaultResponse—** A string expression displayed in the text box as the default response if no other input is provided. Note that if you omit **DefaultResponse**, the displayed text box is empty.
- **XPos—** The distance in pixels of the left edge of the dialog box from the left edge of the screen. Note that if you omit **XPos**, the dialog box is centered horizontally.

- **YPos—** The distance in pixels of the upper edge of the dialog box from the top of the screen. Note that if you omit **YPos**, the dialog box is positioned vertically about one-third of the way down the screen.
- Input boxes let you display a prompt and read a line of text typed by the user, and the InputBox function returns the string result.

Example:

```
Private Sub Button3_Click
   Dim Result As String
   Result = InputBox("Enter your text!")
   TextBox1.Text = Result
End Sub
```

Function Statement:

Declares the name, parameters, and code that define a **Function** procedure.

Syntax:

[**Public** | **Private**] **Function** name [**(**arglist**)**] [**As** returntype]
    [statements]
    [name **=** expression]
    [**Exit Function**]
    [statements]
    [name **=** expression]

End Function

- **name :** Required. Name of the procedure.
- **Parameterlist:** Optional. List of local variable names representing the parameters of this procedure.

- **returntype :** Required if **Option Strict** is **On**. Data type of the value returned by this procedure.
- Specifies the parameters a procedure expects when it is called. Multiple parameters are separated by commas. The following is the syntax for one parameter.

[Optional] [{ ByVal | ByRef }] *parametername*[( )] [As *parametertype*] [= *defaultvalue*]

- **Optional** : Optional. Specifies that this parameter is not required when the procedure is called.
- **ByVal** : Optional. Specifies that the procedure cannot replace or reassign the variable element underlying the corresponding argument in the calling code.
- **ByRef** : Optional. Specifies that the procedure can modify the underlying variable element in the calling code the same way the calling code itself can.
- ***parametername* :** Required. Name of the local variable representing the parameter.
- ***parametertype* :** Required if **Option Strict** is **On**. Data type of the local variable representing the parameter.
- ***defaultvalue* :** Required for **Optional** parameters. Any constant or constant expression that evaluates to the data type of the parameter. If the type is **Object**, or a class, interface, array, or structure, the default value can only be **Nothing**.

Example:

```
Private Function res1(ByVal n1 As Double) As Double
    Return (n1 * 3.14)
End Function
Private Sub butOk_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butOk.Click
```

```
        ans = res1(Val(Me.txtN1.Text))
        MsgBox("Result is:" & ans)
    End Sub
```

About SDI

- Most applications in Windows 95 or later use a Single Document Interface. Each window of the application holds a single document, so if the user wants to open more documents with that application, he must open a new window. also the default mode when building an application with VB.Net. An example of an SDI application is Windows Notepad.

Advantages of SDI

- An SDI interface works very well with multiple monitors and multiple virtual desktops. It also allows users to switch between multiple open documents using the native Windows taskbar and task manager, rather than through special code that must be written into your application.

About MDI

- Multiple Document Interfaces were more popular in versions of Windows prior to Windows 95. With an MDI, each window within an application holds multiple documents, usually in sub-windows. Each time the user wants to open a new document, rather than opening a new window, the document opens within the existing window and shares it with all other open documents. An example of an MDI application is a tabbed Web browser like Firefox, where users have an option to open documents in multiple tabs within the same window.

Advantages of MDI

- MDI applications can often handle multiple documents more readily than SDI programs. For example, many MDI text editors allow the user to open multiple text files side by side in the same window, making it easy to compare and look up information from a second document while working on the first.

Answer the following questions:

1) Differentiate between: Message box – Input box

	Message box		Input box
1	A message box is non-interactive.	1	An Input box is interactive.
2	It just simply displays an output message for the user to read	2	it usually will ask the user to type in some value such as either a number/or, text.
3	When you click button [OK]...the message box immediately closes down/disappears. MsgBox("Hello, world!")	3	whenever the user clicks on the button [OK]; the input dialog box disappears; however, the number/text value which the user went and typed in will, normally, get stored inside of a program memory box variable name; so that the program can get to do something with this value, later on. num=InputBox ("Enter a number: ","PROGRAM: Find square of number") MsgBox("The square of your number is: " & num*num)
4	The MsgBox displays a dialog box with a message displayed on a label and command button to choose one of the few possible actions.	4	in an InputBox, it displays a dialog box with a message displayed on a label, a textbox to enter data and command button to accept or ignore the data entered.

2. Write a difference between Procedure and Function.

FUNCTION IN VB	PROCEDURE IN VB
A procedure that enclosed by the Function and End Function statements	A block of Visual Basic statements enclosed by a declaration statement and a matching End declaration
Used to perform a contain task	Helps to make the code readable, easy to modify and debug
A function is a specific type of procedure	A procedure is a generalized type of function

3. Write a difference between MDI and SDI.

SDI	MDI
SDI applications, require the user to close the currently open document before opening another.	MDI applications permit the user to have two or more documents open for editing at once.
SDI does not support multiple documents at all.	MDI applications some times support multiple document types.
SDI applications generally feature just one menu.	MDI applications have at least two
SDI applications use just one frame window—the top-level frame window that serves as the application's main window and frames views of open documents.	MDI applications use two: a top-level frame window and *child frames or document frames that float within the top-level frame window and frame views of open documents*

4. Define Property, method and event.

Events

- Events are the message sent by an object to the main program loop, these messages have some information about some specific occurrence that takes place.
- For example, when we click a button a click event occurs and we can handle that event with the help of Event handler.

Properties

- They are retrieved and set like variables.
- **Property Get** and **Property Set** procedures which provide more control on how values are set use to implement the properties.

Methods

- Methods represent the object's built-in procedures.
- For example, a class named First have a method named Display. We can define a method as given blow:

```
Public Class First
   'creating a class named First
   Sub Display ()
      'creating a method named Display in the class
   End Sub
End Class
```

We use New keyword to create the Object for this class and is like

```
Dim obj As New First ().
```

5. List any five properties supported by Windows Form and explain any one of them.

S. N	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 specifies 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.

6. Explain Show method of Windows Form.

Show method displays the control to the user.

Example: Form1.show()

7. List any five events supported by Windows Form and explain any one of them.

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.

➢ What is Control?

A control is an object that can be drown on to the from making control are visible object. The control are to enable or and hence user interaction with the Application. All the control has properties methods and events.

Control class is the base class of all the windows controls. We can work with controls in two ways at Design time and at Run time.

Some of common properties of the controls are given the below.

1. **Back color** - Background Color.
2. **Background Image** -Background Image.
3. **Bottom** -The distance between the Bottom of the control and the top of the it's contains.
4. **Context menu** – It gets or sets the shortcut menu for the control.
5. **Cursor property** – It the cursor.
6. **Unable property** - It gets or sets a value indicting if they control is enable.
7. **Font** – To gets or sets the font for the control.
8. **Fore color** – It gets or sets the font color of the control.
9. **Height** – It gets or sets the height of control.
10. **Left** – It gets or sets the X co-ordinate of the control left page in pixels.
11. **Location** – It gets or sets the co-ordinate of the upper left corner of the control.
12. **Locked** – It gets or sets the size or move the control at design time.
13. **Name property** – It gets or sets name of the control.
14. **Right** – It retunes the distance between the right edge of the control and the left edge of its container.
15. **Size** – It gets or sets size of the control in pixels.
16. **Tab index** – It gets or sets tab order of this control in its container.
17. **Tab Stop** – It gets or sets a value specify if the user can Tab or Shift + Tab to this control with the Tab key.
18. **Text** – It gets or sets text for this control.
19. **Top** – It gets or sets the top co-ordinate of the control.
20. **Visible** – It gets or sets a value for visibility of the control.
21. **Width** – Its gets or sets width of the control.

Some common event of the control are as given below :

1. **Click** – It occurs then the control it click
2. **Got Focus** – It occurs when the control received focus.
3. **Key down** – It occurs when the key is pressed the control has focus.
4. **Key pressed –** It occurs when a key is pressed one control has focus.
5. **Key up –** It occurs when a key is released by the control has focus.
6. **Lost focus –** It occurs when the control user losses focus.
7. **Mouse click –** It occurs when the control is click by the mouse.
8. **Mouse down** – It occurs when the mouse pointer is over the control and the mouse button is pressed.

9. **Mouse enter** – It occurs when the mouse pointer enter the controls the mouse over it occurs when the mouse pointer rests on the controls.
10. **Mouse leave** – It occurs when the mouse pointer lives the control.
11. **Mouse move** - It occurs when the mouse pointer is moved over the control.
12. **Mouse up-** It occurs when the mouse pointer is over the control and a mouse button is released.
13. **Mouse while** - It occurs when the mouse while moves when the control has focus.

Button

One of the most popular controls in Visual Basic is the Button Control (previously Command Control). They are the controls which we click and release to perform some action. Buttons are used mostly for handling events in code, say, for sending data entered in the form to the database and so on. The default event of the Button is the Click event and the Button class is based on the ButtonBase class which is based on the Control class.

Properties:

Properties	Description
Visible	Property used to make the control visible or invisible
Enabled	This property is used to enable the control.
Width	This property is used to specify the width of the control.
Font	Property used to set the font properties like bold, Italic, Name and so on.
Height	Property used to specify the height.
Width	Property is used to set the width of the control.
Left	Property is set the X coordinate of the control.
BackColor	Property is used set the background color of the control.
Image	Property is used to set a background picture for the control.

DialogResult	Property is used to set or get the value returned to the parent from when the button is clicked.
Image	Property is used to set a background picture for the control.
ImageAlign	Property is used to get or set alignment of the image.
ImageList	Property is used to set or get the Image List that contains the images displayed.

Methods:

Method	Description
PerformClick	Method used to generate a click event for the radio button.

Events:

Events	Description
Enter	Triggered when Command Button Control gets focus.
Click	Triggered when the control is clicked.
Leave	Triggered when control loses focus.

Label

Label Control is usually used to display text that cannot be edited by the user. But using the properties or code that is displayed can be changed.
Labels are those controls that are used to display text in other parts of the application. They are based on the Control class.
Notable property of the label control is the text property which is used to set the text for the label.

Properties:

Properties	Description
AutoSize	This property is used to set or get a value specifying if the control should be automatically resized.
BorderStyle	This property is used to get or set the border style for the control.

FlatStyle	This property is used to get or set the flat style appearance of label control.
Image	Property is used to set or get the image that is displayed on a label.
ImageAlign	Property is used to set or get the alignment of an image that is displayed in the control.
PreferredHeight	Property is used to get the preferred height of the control.
PreferredWidth	Property is used get the preferred width of the control.
TextAlign	Property is used set or get the alignment of text in the control.
UseMnemonic	Property is used set or get a value specifying whether to consider ampersand as an access key character.

TextBox

- This controls looks like a box and accepts input from the user.
- Typically a textbox control is used to display or accept as input a single line of text.
- Text can display multi line you can do that by setting the textbox multi line property is true.
- Using the scrollbar property's there are four ways to add scrollbar to a textbox value of scrollbar is 0 as nun, 1 as Horizontal 2 as vertical, 3 as both.
- You can use text line properties you can set it to lest justified, right justified and centered.
- You can use text line properties to make a textbox read only setting this properties to true means that the user cannot enter text into the textbox.
- You can disable a textbox by setting its enabled property to false.

By default a textbox holds up to 32767 char. But when displaying multi line a textbox holds up to 2GB of text.
To convert a slandered textbox into a password box you just assign some char to the textboxes password char. Property.
The text property contains in the control to set or get text.

Properties:

Properties	Description
TextAlign	Text alignment is set using this property
Multiline	This property is used to set more than one-line text
ScrollBars	This property is used to specify vertical and horizontal scroll bars.
MaxLength	Property used to specify the maximum number of characters accepted by a TextBox Control.
Enabled	Property used to enable a textbox control.
Index	Property used to specify the index of a control array.
Readonly	Property is set to true to use the control, false the control cannot be used.
SelectionLength	Property is used set or get the number of characters selected in the text box.
SelectionStart	Property is used set or get the starting point of a text box.
SelectedText	Property is used for indicating the currently selected text box.

Methods:

Method	Description
Drag	Method used for the drag drop functionality.
SetBounds	Methods used to set the bound for the control at the specifed location and size.
Focus	Method to set focus to a TextBox Control
Clear	Method used to clear all text from the text box.
SelectAll	Methods used to select all the text in the control.

Select	Method to selects the text in the text box.
Copy	Method used to copy selected text.
Cut	Methods used to cut the selected text.
Paste	Method to paste the text in the control to the clipboard.

Events:

Events	Description
AutoSizeChanged	Triggered when the AutoSize property is changed.
ReadOnlyChanged	Triggered when ReadOnly property value changes.
Click	Triggered when the control is clicked .

Radio Button

Properties:

Properties	Description
Text	Property used to display the text center aligned on the control.
Size	This property is used to specify the width, height of the control.
Font	Property used to set the font properties like bold, Italic, Name and so on.
Appearance	Property is used to set or get the value that determines the appearance.
Autocheck	Property is used to set or get whether the checked value, appearance change when clicked on the radio button.
Checked	Property is used to set or get the value indicating whether the radio button is checked.
FlatStyle	Property is used to set or get flatStyle

	appearance of the radio button.
Image	Property is used to set or get the image displayed on the radio button.
ImageAlign	Property is used to set or get the alignment of the image on the control.
ImageList	Property is used to set or get the images displayed on the control.

Methods:

Method	Description
PerformClick	Method used to generate a click event for the radio button.

Events:

Events	Description
CheckedChanged	Triggered when the control is checked or changed.
AppearanceChanged	Triggered when the appearance property changes.

Combobox

Properties:

Sr.No.	Property & Description
1	**AllowSelection** Gets a value indicating whether the list enables selection of list items.
2	**AutoCompleteCustomSource** Gets or sets a custom System.Collections .Specialized.StringCollection to use when the AutoCompleteSourceproperty is set to CustomSource.
3	**AutoCompleteMode** Gets or sets an option that controls how automatic completion works for the ComboBox.
4	**AutoCompleteSource** Gets or sets a value specifying the source of complete strings used for

	automatic completion.
5	**DataBindings** Gets the data bindings for the control.
6	**DataManager** Gets the CurrencyManager associated with this control.
7	**DataSource** Gets or sets the data source for this ComboBox.
8	**DropDownHeight** Gets or sets the height in pixels of the drop-down portion of the ComboBox.
9	**DropDownStyle** Gets or sets a value specifying the style of the combo box.
10	**DropDownWidth** Gets or sets the width of the of the drop-down portion of a combo box.
11	**DroppedDown** Gets or sets a value indicating whether the combo box is displaying its drop-down portion.
12	**FlatStyle** Gets or sets the appearance of the ComboBox.
13	**ItemHeight** Gets or sets the height of an item in the combo box.
14	**Items** Gets an object representing the collection of the items contained in this ComboBox.
15	**MaxDropDownItems** Gets or sets the maximum number of items to be displayed in the drop-down part of the combo box.
16	**MaxLength** Gets or sets the maximum number of characters a user can enter in

	the editable area of the combo box.
17	**SelectedIndex** Gets or sets the index specifying the currently selected item.
18	**SelectedItem** Gets or sets currently selected item in the ComboBox.
19	**SelectedText** Gets or sets the text that is selected in the editable portion of a ComboBox.
20	**SelectedValue** Gets or sets the value of the member property specified by the ValueMember property.
21	**SelectionLength** Gets or sets the number of characters selected in the editable portion of the combo box.
22	**SelectionStart** Gets or sets the starting index of text selected in the combo box.
23	**Sorted** Gets or sets a value indicating whether the items in the combo box are sorted.
24	**Text** Gets or sets the text associated with this control.

Methods:

Sr.No.	Method Name & Description
1	**BeginUpdate** Prevents the control from drawing until the EndUpdate method is called, while items are added to the combo box one at a time.
2	**EndUpdate** Resumes drawing of a combo box, after it was turned off by the BeginUpdate method.

3	**FindString** Finds the first item in the combo box that starts with the string specified as an argument.
4	**FindStringExact** Finds the first item in the combo box that exactly matches the specified string.
5	**SelectAll** Selects all the text in the editable area of the combo box.

Events:

Sr.No.	Event & Description
1	**DropDown** Occurs when the drop-down portion of a combo box is displayed.
2	**DropDownClosed** Occurs when the drop-down portion of a combo box is no longer visible.
3	**DropDownStyleChanged** Occurs when the DropDownStyle property of the ComboBox has changed.
4	**SelectedIndexChanged** Occurs when the SelectedIndex property of a ComboBox control has changed.
5	**SelectionChangeCommitted** Occurs when the selected item has changed and the change appears in the combo box.

Checkbox

Sr.No.	Property & Description
1	**Appearance** Gets or sets a value determining the appearance of the

	check box.
2	**AutoCheck** Gets or sets a value indicating whether the Checked or CheckedState value and the appearance of the control automatically change when the check box is selected.
3	**CheckAlign** Gets or sets the horizontal and vertical alignment of the check mark on the check box.
4	**Checked** Gets or sets a value indicating whether the check box is selected.
5	**CheckState** Gets or sets the state of a check box.
6	**Text** Gets or sets the caption of a check box.
7	**ThreeState** Gets or sets a value indicating whether or not a check box should allow three check states rather than two.

Methods:

Sr.No.	Method Name & Description
1	**OnCheckedChanged** Raises the CheckedChanged event.
2	**OnCheckStateChanged** Raises the CheckStateChanged event.
3	**OnClick** Raises the OnClick event.

Events:

Sr.No.	Event & Description
1	**AppearanceChanged** Occurs when the value of the Appearance property of the check box is changed.
2	**CheckedChanged** Occurs when the value of the Checked property of the CheckBox control is changed.
3	**CheckStateChanged** Occurs when the value of the CheckState property of the CheckBox control is changed.

Listbox

Properties

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 box.
8	**MultiColumn** Gets or sets a value indicating whether the list box supports multiple columns.
9	**ScrollAlwaysVisible** Gets or sets a value indicating whether the vertical scroll bar is shown at all times.
10	**SelectedIndex** Gets or sets the zero-based index of the currently selected item in a list box.
11	**SelectedIndices** Gets a collection that contains the zero-based indexes of all currently selected items in the list box.
12	**SelectedItem** Gets or sets the currently selected item in the list box.
13	**SelectedItems** Gets a collection containing the currently selected items in the list box.
14	**SelectedValue** Gets or sets the value of the member property specified by the ValueMember property.
15	**SelectionMode** Gets or sets the method in which items are selected in the list box. This property has values − • None • One • MultiSimple • MultiExtended

16	**Sorted** Gets or sets a value indicating whether the items in the list box are sorted alphabetically.
17	**Text** Gets or searches for the text of the currently selected item in the list box.
18	**TopIndex** Gets or sets the index of the first visible item of a list box.

Methods

Sr.No.	Method Name & Description
1	**BeginUpdate** Prevents the control from drawing until the EndUpdate method is called, while items are added to the ListBox one at a time.
2	**ClearSelected** Unselects all items in the ListBox.
3	**EndUpdate** Resumes drawing of a list box after it was turned off by the BeginUpdate method.
4	**FindString** Finds the first item in the ListBox that starts with the string specified as an argument.
5	**FindStringExact** Finds the first item in the ListBox that exactly matches the specified string.
6	**GetSelected** Returns a value indicating whether the specified item is selected.
7	**SetSelected** Selects or clears the selection for the specified item in a ListBox.

8	**OnSelectedIndexChanged** Raises the SelectedIndexChanged event.
8	**OnSelectedValueChanged** Raises the SelectedValueChanged event.

Events

Sr.No.	Event & Description
1	**Click** Occurs when a list box is selected.
2	**SelectedIndexChanged** Occurs when the SelectedIndex property of a list box is changed.

Timer

Properties:

Property Name	Description
Name	It is used to specify name of the Timer Control.
Enabled	It is used to determine whether Timer Control will be enabled or not. It has Boolean value true or false. Default value is false.
Interval	It is used to specify interval in millisecond. Tick event of Timer Control generates after the time which is specified in Interval Property.

Methods:

Method Name	Description
Start	This method is used to start the Timer Control.
Stop	This method is used to stop the Timer Control.

Events:

Event Name	Description

Tick	Tick event of the Timer Control fires continuously after the time which is specified in the Inteval property of Timer Control.

PictureBox

Properties:

Properties	Description
BackColor	Property gets or sets the bsckground color for the control.
SizeMode	Property used to Get or sets size options for the control.
BorderStyle	Property used to specify the border style for the control.
Font	Property to use to set or get the font style for the control.
Image	Property used to specify the image to be loaded either from a resource file or from a local location.
Visible	Property to use to specify whether to make the control visible.
WaitOnLoad	Property to use to wait till a big image gets loaded.
Enabled	Property to use to enable or disable the control.

Events:

Events	Description
Resize	Triggered when the picture box is resized.
SizeModeChanged	Triggered when the SizeMode changes.

HScrollbar and VScrollbar

Properties

Sr.No.	Property & Description
1	**AutoSize** Gets or sets a value indicating whether the ScrollBar is automatically resized to fit its contents.
2	**BackColor** Gets or sets the background color for the control.
3	**ForeColor** Gets or sets the foreground color of the scroll bar control.
4	**ImeMode** Gets or sets the Input Method Editor (IME) mode supported by this control.
5	**LargeChange** Gets or sets a value to be added to or subtracted from the Value property when the scroll box is moved a large distance.
6	**Maximum** Gets or sets the upper limit of values of the scrollable range.
7	**Minimum** Gets or sets the lower limit of values of the scrollable range.
8	**SmallChange** Gets or sets the value to be added to or subtracted from the Value property when the scroll box is moved a small distance.
9	**Value** Gets or sets a numeric value that represents the current position of the scroll box on the scroll bar control.

Methods

Sr.No.	Method Name & Description

1	**OnClick** Generates the Click event.
2	**Select** Activates the control.

Events

Sr.No.	Event & Description
1	**Click** Occurs when the control is clicked.
2	**DoubleClick** Occurs when the user double-clicks the control.
3	**Scroll** Occurs when the control is moved.
4	**ValueChanged** Occurs when the Value property changes, either by handling the Scroll event or programmatically.

CheckedListBox

Property	Purpose
CheckOnClick	It is used to specify weather CheckBox should be toggled (change state) or not when an item is selected in the CheckedListBox. It has Boolean value. Default value is False.
MultiColumn	It is used to specify whether CheckedListBox supports multiple columns or not. It has Boolean value. Default value is false.
ColumnWidth	It is used to specify width of each column in MultiColumn CheckedListBox.

Items	It represents collection of items contained in CheckedListBox control.
Sorted	It is used to specify weather items of CheckedListBox are sorted in alphabetical order or not. It has Boolean value. Default value is false.
SelectionMode	It is used to get or set SelectionMode of CheckedListBox. It determines how user can select the Items of CheckedListBox. It can have one of the following four options: **(1)None:** No Selection is allowed **(2)One:** User can select only one item at a time. **(3)MultiSimple:** User can select or deselect item just by mouse click or pressing spacebar. **(4) MultiExtended:** User can select or deselect items by holding Ctrl key and mouse click. User can also select or deselect items by pressing Shift key and mouse click or arrow key. Default value is One.
ScrollAlwaysVisible	It is used to specify weather Scroll Bars is always associated with CheckedListBox or Not regardless of number of items present in CheckedListBox. It has Boolean value. Default value is false.
HorizontalExtent	It is used to get or set width in pixel, by which a CheckedListBox can scrolled horizontally. It works only when Horizontal Scrollbar Property is set to true.
HorizontalScrollbar	It is used to specify whether CheckedListBox can have Horizontal Scroll Bar or Not, If Number of Items in CheckedListBox are not accommodate in specified width. It has Boolean value. Default value is False.
ThreeDCheckBoxes	It is used to get or set value which determines CheckBox has Flat or Normal Button State. It has Boolean value.

	Default value is false. When it is true check box has flat button state.
SelectedIndex	It is used to get or set zero based index of the item currently selected in CheckedListBox.
SelectedItem	It is used to get or set item currently selected in CheckedListBox.
SelectedItems	It is used to get collection of multiple items currently selected in CheckedListBox.
CheckedItems	It is used to get collection of multiple items currently checked in CheckedListBox.
SelectedIndices	It is used to get collection of zero-based indexes of all items currently selected in CheckedListBox.
CheckedIndices	It is used to get collection of zero-based indexes of all items currently checked in CheckedListBox.
Enable	It is used to specify whether CheckedListBox Control is enabled or not at runtime. It has Boolean value. Default value is true.
Visible	It is used to specify whether CheckedListBox Control is visible or not at runtime. It has Boolean value. Default value is true.
TabStop	It is used to specify weather user can use TAB key to set focus on CheckedListBox Control or not. It has Boolean value. Default value is true.

Methods

Method	Purpose
ClearSelected	It is used to unselect all the items that are currently

	selected in ListBox.
FindString	It is used to find first occurrences of an item in the ListBox that partially match with string specified as an argument. If an item is found than it returns zero based index of that item, otherwise it returns -1. The search performed by this method is case insensitive.
FindStringExact	It is used to find first occurrences of an item in the ListBox that exactly match with string specified as an argument. If an item is found than it returns zero based index of that item, otherwise it returns -1. The search performed by this method is case insensitive.
GetSelected	It is used to determine whether an item whose index is passed as an argument is selected or not. It returns Boolean value.
SetSelected	It is used to select or deselect an item whose index is passed as an argument. **Example:** ListBox1.SetSelected (1, true) will select second item of ListBox.
ClearSelected	It is used to unselect all items in CheckedListBox.
GetItemChecked	It is used to check weather an item whose index is passed as an argument is checked or not. It returns Boolean value.
GetItemCheckState	It is used to get check state of an item whose index is passed as an argument. It returns 1 if item is checked otherwise false.
SetItemCheckState	It is used to set the check state of an item whose index is passed as an argument. **Example:**

	CheckedListBox1.SetItemChecked (1, CheckState.Checked) will check the second item of CheckedListBox.

Events

Event	Purpose
SelectedIndexChanged	It is the default event of ListBox Control. It fires each time a selected Item in the ListBox is changed.
ItemCheck	It fires each time an item is checked or unchecked.

GroupBox

Properties

Property	Purpose
BackColor	It is used to get or set background color of the GroupBox.
BackgroundImage	It is used to get or set background Image of the GroupBox.
BackgroundImageLayout	It is used to get or set background Image layout of the GroupBox. It has one of the following values: None, Tile, Centre, Stretch, Zoom
Font	It is used to get or set font Style, Font Size, Font Face of the text contained in GroupBox Control.
ForeColor	It is used to get or set color of the text contained in GroupBox Control.
Enabled	It is used to specify whether GroupBox Control is enabled or not. It has Boolean value. Default

	value is true.
Visible	It is used to specify whether GroupBox Control is visible or not at run time. It has Boolean value. Default value is true.
Text	It is used to get or set Title or Header Text of the GroupBox Control.

Methods

Method	Purpose
Show	It is used to show GroupBox Control.
Hide	It is used to Hide GroupBox Control at run time.
Focus	It is used to set cursor or focus on GroupBox Control.

DateTimePicker

Properties:

Properties	Description
CalendarFont	Property to set the font for calendar.
CalendarForeColor	Property to set or get the foreground color for the calendar.
CalendarMonthBackGround	Property to set or get the background color for the calendar month.
CalendarTitleBacKColor	Property to get or set background color for the calendar title.
CalendarTitleForeColor	Property to get or set foreground color for the calendar title.
BackColor	Property to set the background color for the control.
BindingContext	Property used to set or get the binding context for the control.

Checked	Property used to get or set a value indicating whether the Value property has been set a valid date/time value that can be updated.
Format	Property used to set or get the format of the date and time displayed.
MaxDate	Property used to set or get the maximum date that can be selected using the control.
MinDate	Property used to set or get the minimum date that can be selected using the control.
ShowCheckBox	Property used to set or get a value to decide whether to display a check box to the left of the selected date.
Value	Property used to set or get the date, time value assigned to the control.
Width	Property set width of the control.
RightToLeft	Property specifies a value to know whether the text appears from right to left.
Locked	Prevents the control being moved at design time.

Methods:

Method	Description
DoDragDrop	Method used to begin Drag Drop Operation.
Equals	Method used to check if two instances of an object are equal.
FindForm	Method used to retreive the form the control is on.
Focus	Method used to set the input focus on the control.
Hide	Method used to hide the control.

PointToScreen	Method used to calculate the location of the specified client point into screen coordinates.
ToString	Method used to convert this object into its equivalent String value.
Select	Method used to select the control.

Events:

Events	Description
DragDown	Triggered when the dropdown claendar appears.
CloseUp	Triggered when the dropdown calendar disappears.
FormatChanged	Triggered when the format property is changed.
ValueChanged	Triggered when the value property is chnaged.

LinkLabel

Properties

Property	Purpose
LinkColor	It is used to get or set Fore color of the Hyperlink in its default state.
ActiveLinkColor	It is used to get or set Fore color of the Hyperlink when user clicks it.
DisabledLinkColor	It is used to get or set Fore color of the Hyperlink when LinkLabel is disabled.
VisitedLinkColor	It is used to get or set Fore color of the Hyperlink when LinkVisited property of LinkLabel is set to true.
LinkVisited	It is used to specify weather Hyperlink is already visited or

	not. It has Boolean value. Default value is false.
Text	It is used to get or set text associated with LinkLabel Control.
TextAlign	It is used to get or set alignment of the text associated with LinkLabel Control.
ForeColor	It is used to get or set Fore Color of the text associated with LinkLabel Control.
BackColor	It is used to get or set Background color of the LinkLabel Control.
Enabled	It is used to specify weather LinkLabel control is enabled or not at runtime. It has Boolean value true or false. Default value is true.
Visible	It is used to specify weather LinkLabel control is visible or not at runtime. It has Boolean value true or false. Default value is true.

Methods:

Method	Purpose
Show	It is used to Show LinkLabel Control at runtime.
Hide	It is used to Hide LinkLabel Control at runtime.
Focus	It is used to set input focus on LinkLabel Control.

Events:

Event	Purpose
Link Clicked	It is the default event of LinkLabel Control. It fires each time a user clicks on a hyperlink of LinkLabel Control.

TreeView

Properties:

Sr.No.	Property & Description
1	**BackColor** Gets or sets the background color for the control.
2	**BackgroundImage** Gets or set the background image for the TreeView control.
3	**BackgroundImageLayout** Gets or sets the layout of the background image for the TreeView control.
4	**BorderStyle** Gets or sets the border style of the tree view control.
5	**CheckBoxes** Gets or sets a value indicating whether check boxes are displayed next to the tree nodes in the tree view control.
6	**DataBindings** Gets the data bindings for the control.
7	**Font** Gets or sets the font of the text displayed by the control.
8	**FontHeight** Gets or sets the height of the font of the control.
9	**ForeColor** The current foreground color for this control, which is the color the control uses to draw its text.
10	**ItemHeight** Gets or sets the height of each tree node in the tree view control.
11	**Nodes** Gets the collection of tree nodes that are assigned to the tree

	view control.
12	**PathSeparator** Gets or sets the delimiter string that the tree node path uses.
13	**RightToLeftLayout** Gets or sets a value that indicates whether the TreeView should be laid out from right-to-left.
14	**Scrollable** Gets or sets a value indicating whether the tree view control displays scroll bars when they are needed.
15	**SelectedImageIndex** Gets or sets the image list index value of the image that is displayed when a tree node is selected.
16	**SelectedImageKey** Gets or sets the key of the default image shown when a TreeNode is in a selected state.
17	**SelectedNode** Gets or sets the tree node that is currently selected in the tree view control.
18	**ShowLines** Gets or sets a value indicating whether lines are drawn between tree nodes in the tree view control.
19	**ShowNodeToolTips** Gets or sets a value indicating ToolTips are shown when the mouse pointer hovers over a TreeNode.
20	**ShowPlusMinus** Gets or sets a value indicating whether plus-sign (+) and minus-sign (-) buttons are displayed next to tree nodes that contain child tree nodes.
21	**ShowRootLines** Gets or sets a value indicating whether lines are drawn between the tree nodes that are at the root of the tree view.

22	**Sorted** Gets or sets a value indicating whether the tree nodes in the tree view are sorted.
23	**StateImageList** Gets or sets the image list that is used to indicate the state of the TreeView and its nodes.
24	**Text** Gets or sets the text of the TreeView.
25	**TopNode** Gets or sets the first fully-visible tree node in the tree view control.
26	**TreeViewNodeSorter** Gets or sets the implementation of IComparer to perform a custom sort of the TreeView nodes.
27	**VisibleCount** Gets the number of tree nodes that can be fully visible in the tree view control.

Methods

Sr.No.	Method Name & Description
1	**CollapseAll** Collapses all the nodes including all child nodes in the tree view control.
2	**ExpandAll** Expands all the nodes.
3	**GetNodeAt** Gets the node at the specified location.
4	**GetNodeCount** Gets the number of tree nodes.

5	**Sort** Sorts all the items in the tree view control.
6	**ToString** Returns a string containing the name of the control.

Events:

Sr.No.	Event & Description
1	**AfterCheck** Occurs after the tree node check box is checked.
2	**AfterCollapse** Occurs after the tree node is collapsed.
3	**AfterExpand** Occurs after the tree node is expanded.
4	**AfterSelect** Occurs after the tree node is selected.
5	**BeforeCheck** Occurs before the tree node check box is checked.
6	**BeforeCollapse** Occurs before the tree node is collapsed.
7	**BeforeExpand** Occurs before the tree node is expanded.
8	**BeforeLabelEdit** Occurs before the tree node label text is edited.
9	**BeforeSelect** Occurs before the tree node is selected.
10	**ItemDrag** Occurs when the user begins dragging a node.

11	**NodeMouseClick** Occurs when the user clicks a TreeNode with the mouse.
12	**NodeMouseDoubleClick** Occurs when the user double-clicks a TreeNode with the mouse.
13	**NodeMouseHover** Occurs when the mouse hovers over a TreeNode.
14	**PaddingChanged** Occurs when the value of the Padding property changes.
15	**Paint** Occurs when the TreeView is drawn.
16	**RightToLeftLayoutChanged** Occurs when the value of the RightToLeftLayout property changes.
17	**TextChanged** Occurs when the Text property changes.

RichTextBox

Properties:

Properties	Description
AutoSize	Property gets or sets the value specifying to change the rich text box automatically as the font changes.
BackColor	Property used to Get or set background color for the control.
AutoWordSelection	Property used to set or get a value specifying automatic word selection.
CanRedo	Property to use to indicate that any actions can be reapplied.
CanUndo	Property to use to undo any previous

	actions.
HideSelection	Property used to set or get value specify a text should stay highlighted when control loses focus.
MaxLength	Property used to set or get the maximum number of a line a user can type into a rich text box.
Multiline	Property used to set or get a value specifying multiline input for the control.
ScrollBars	Property used to set or get the kind of scroll bar to be used.
SelectedText	Property used to set or get the selected text within the control.
SelectionColor	Property used to set or get color for the selected text.
SelectionLength	Property used to set or get the number of characters selected in the control.
SelectionFont	Property used to set or get the font for the selected text.
TextLength	Property used to set or get the length of text in the control.
Text	Property used to set or get the current text in the control.

Methods:

Method	Description
Appends	Method used to append text to current text of the control.
CanPaste	Method determines if the infromation can be pasted from a clipboard.
Clear	Method used to clear text from the control.
Find	Method used to search a text inside

	the control.
GetLineFromCharIndex	Method used to get the line number from the specified character position.
GetPositionFromCharIndex	Method used to get the location within the control at the specified character index.
LoadFile	Methods used to load the contents of a file into the control.
Redo	Method to reapply the last operation.
Select	Methods used to select the text within the control.
Undo	Method to undo the last edit operation.

Events:

Events	Description
Click	Triggered when the control is clicked.
LinkClicked	Triggered when the user clicks on the link within the text.
ModifiedChanged	Triggered when the value of the Modified property is changed.
ReadOnlyChanged	Triggered when the value of the ReadOnly property is changed.
SelectionChanged	Triggered when the value of the Selection property is changed.
VScroll	Triggered when the vertical scroll bars are clicked.

ColorDialogBox

Properties	Description
AllowFullOpen	Property gets or sets whether the user can use the dialog box to define custom colors.
AnyColor	Property gets or sets whether a dialog box displays all available colors in the

	set of basic colors.
Color	Property gets or sets the color selected by the user.
FullOpen	Property used to get or set whether the controls used to create custom colors are visible.
ShowColorOnly	Property used to set or get whether the dialog box will restrict users to select solid colors only.
ShowHelp	Property used to set or get whether the dialog box displays Help button.

Methods:

Method	Description
Reset	Method used to reset all dialog options to default values.
ShowDialog	Method shows the dialog box.

Events:

Events	Description
HelpRequest	Triggered when the user clicks the Help Button.

FontDialogbox

Properties:

Properties	Description
AllowSimulations	Property gets or sets if the dialog box allows graphics device interface font simulations.
AllowVectorFonts	Property gets or sets whether a dialog box allows vector font selections.
AllowVerticalFonts	Property gets or sets whether a dialog box displays both vertical and horizontal font or any one.
Color	Property used to get or set the selected font color.
FixedPitchOnly	Property used to get or set whether the dialog box allows only the selection of fixed pitch fonts.
Font	Property used to get or set the selected font.

FontMustExists	Property to use to get or set whether the dialog box specifies an error condition if the user attempts to select a font or style that doesn't exists.
MaxSize	Property to use to set or get the maximum point size a user can select.
MinSize	Property used to set or get the minimum point size a user can select.
ShowApply	Property used to set or get whether dialog box contans an Apply button.
ShowColor	Property used to set or get whether the dialog box display the color choice.
ShowEffects	Property used to set or get whether the dialog box contains controls that allow user to specify strikethrough underline, and text color options.
ShowHelp	Property used to set or get whether the dialog box displays Help button.

Methods:

Method	Description
Reset	Method used to reset all dialog options to default values.
ShowDialog	Method shows the dialog box.

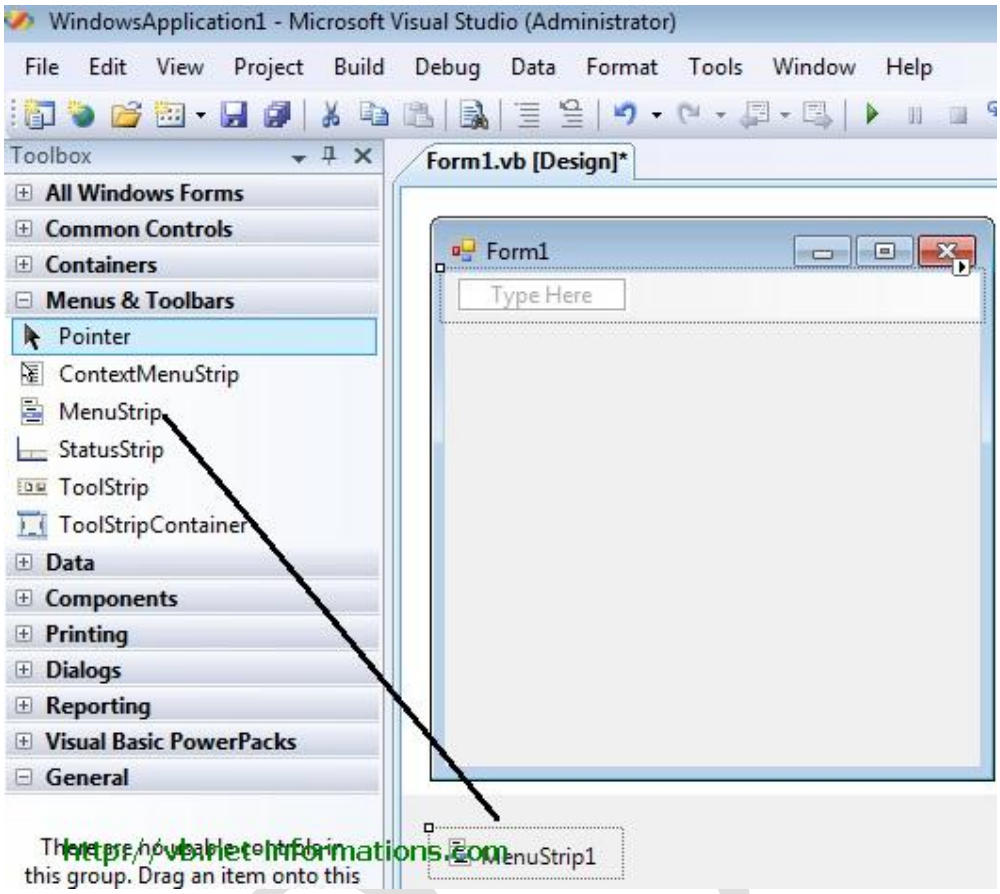
Events:

Events	Description
Apply	Triggers when the user clicks the Apply button.
HelpRequest	Triggered when the user clicks the Help Button.

Menu Control

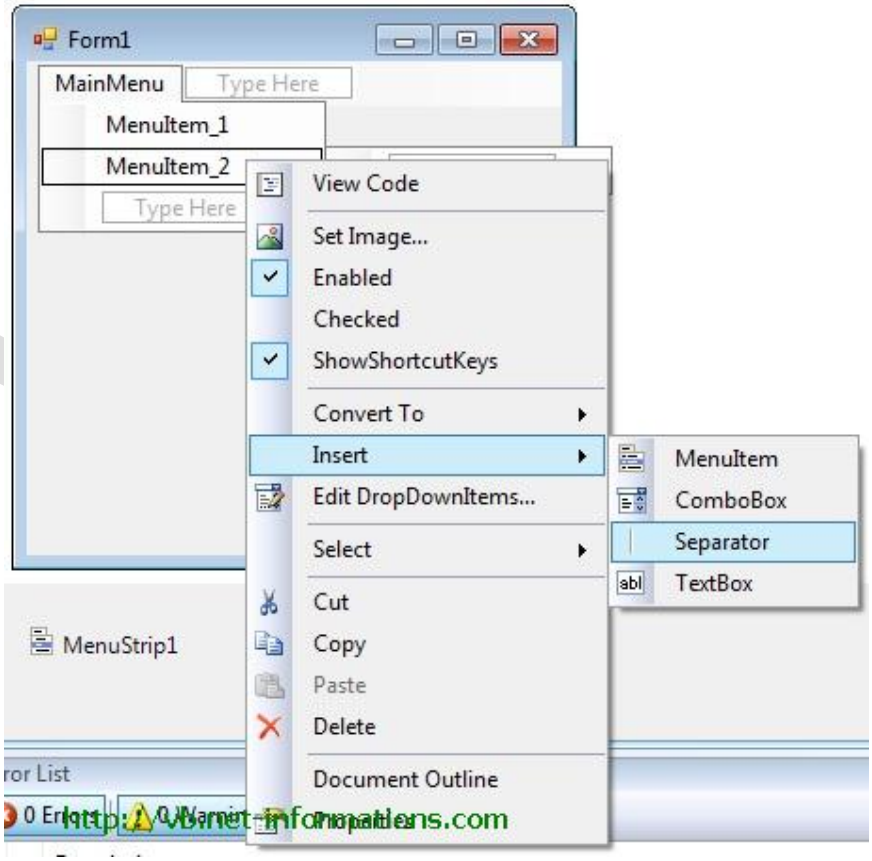
A menu is located on the menu bar and contains a list of related commands. MainMenu is the container for the Menu structure of the form and menus are made of MenuItem objects that represent individual parts of a menu.

You can create a main menu object on your form using the MainMenu control. The following picture shows how to drag the Menustrip Object to the Form.



After drag the Menustrip on your form you can directly create the menu items by type a value into the "Type Here" box on the menubar part of your form. From the following picture you can understand how to create each menu items on mainmenu Object.

If you need a separator bar , right click on your menu then go to insert->Separator.



After creating the Menu on the form , you have to double click on each menu item and write the programs there depends on your requirements. The following Vb.Net program shows how to show a messagebox when clicking a menu item.

Error Handling

The error handling construct in Visual Studio .NET is known as structured exception handling. The constructs used may be new to Visual Basic users, but should be familiar to users of C++ or Java.

Structured exception handling is straightforward to implement, and the same concepts are applicable to either VB.NET or C#. Throughout this section, example code will be shown in both languages.

VB .NET allows backward compatibility by also providing unstructured exception handling, via the familiar On Error GoTo statement and Err object, although this model is not discussed in this section.

Exceptions

Exceptions are used to handle error conditions in Visual Studio .NET. They provide information about the error condition.

An exception is an instance of a class which inherits from the System.Exception base class. Many different types of exception class are provided by the .NET Framework, and it is also possible to create your own exception classes. Each type extends the basic functionality of the System.Exception class by allowing further access to information about the specific type of error that has occurred.

An instance of an Exception class is created and thrown when the .NET Framework encounters an error condition. You can deal with exceptions by using the Try, Catch Finally construct.

Try, Catch, Finally

This construct allows you to catch errors that are thrown within your code. An example of this construct is shown below. An attempt is made to rotate an envelope, which throws an error.

```
Try
  Dim env As IEnvelope = New EnvelopeClass()
  env.PutCoords(0D, 0D, 10D, 10D)
  Dim trans As ITransform2D = env
  trans.Rotate(env.LowerLeft, 1D)
Catch ex As System.Exception
  MessageBox.Show("Error: " + ex.Message)
Finally
```

```
    ' Perform any tidy up code.
End Try
```

The Try block is placed around the code which may fail. If an error is thrown within the Try block, the point of execution will switch to the first Catch block.

The Catch block handles a thrown error. A Catch block is executed when the Type of a thrown error matches the Type of error specified by the Catch block. You can have more than one Catch block to handle different kinds of errors. The code shown below checks first if the exception thrown is a DivideByZeroException.

Try, Catch and Finally keywords.

The Try, Catch and Finally keywords are usually referred to as Error Handling. Handling exceptions with these methods will stop your application from receiving runtime errors (errors encountered during running the application) caused by exceptions.

When you handle these errors, VB.NET gives you the ability to get a wealth of information about the exception and ways to let your application correspond according to them. Before we get into getting exception information, let's talk the complete basics.

- Try - The Try keyword will tell the compiler that we want to handle an area for errors. The Try keyword must have a Catch method, and must be encapsulated in an End Try statement.
- Catch - The Catch keyword tells the application what to do if an error occurs. It will stop the application from crashing, and let the application do the rest.
- Finally - The finally keyword will occur regardless if the Try method worked successfully or a Catch method was used. The finally keyword is optional.

Here is an example of a basic Try...Catch method:

```
01 Try
02     'Create a new integer variable.
03     Dim anInteger As Integer = 0
04     ' Divide zero by zero to produce a DivideByZeroException exception.
05     anInteger \= 0
06 Catch ex As Exception
07     ' Show an error telling the user an error occured.
08     MessageBox.Show("An error occured")
09 Finally
10     ' Tell the user that it has got to the Finally statement.
11     MessageBox.Show("This has finally happened. :)")
12 End Try
```

Unit : 4

Database Programming With ADO.NET

- ADO.NET – introduction and applications
- ADO.NET – architecture (connected and disconnected)
- Database connectivity using ADO.NET
- Use of Data sources, Server Explorer and working with DataSet
- Populating data in a DataGridView
- Working with report

ADO.NET – introduction and applications

- ADO.NET is a part of the Microsoft .Net Framework.
- The full form of ADO.Net is ActiveX® Data Objects.
- ADO.Net has the ability to separate data access mechanisms, data manipulation mechanisms and data connectivity mechanisms.
- ADO.Net is a set of classes that allow application to read and write information in databases.
- ADO.Net can be used by any .Net Language.
- It's concept. It's not a programming language.
- ADO.Net introduces the concept of disconnected architecture.
- We need to add System.Data namespace for work with ADO.Net
- It's a next version of ActiveX Data Objects (ADO) technology which was used in VB6.0.

ADO.NET provides consistent access to data sources such as SQL Server and XML, and to data sources exposed through OLE DB and ODBC. Data-sharing consumer applications can use ADO.NET to connect to these data sources and retrieve, handle, and update the data that they contain.

ADO.NET separates data access from data manipulation into discrete components that can be used separately. ADO.NET includes .NET Framework data providers for connecting to a database, executing commands, and retrieving results. Those results are either processed directly, placed in an ADO.NET DataSet object in order to be exposed to the user in an ad hoc manner, combined with data from multiple sources, or passed between tiers. The **DataSet** object can also be used independently of a .NET Framework data provider to manage data local to the application or sourced from XML.

The ADO.NET classes are found in System.Data.dll, and are integrated with the XML classes found in System.Xml.dll. For sample code that connects to a database, retrieves data from it, and then displays that data in a console window.

ADO.NET provides functionality to developers who write managed code similar to the functionality provided to native component object model (COM) developers by ActiveX Data Objects (ADO). We recommend that you use ADO.NET, not ADO, for accessing data in your .NET applications

Comparison between ADO and ADO.NET

	ADO	ADO.NET
Data access	ADO used connected data usage. (Connection-Oriented Models)	ADO.NET used disconnected data environment. (Disconnected Models)
XML Support	In ADO XML Support is limited.	In ADO.NET XML robust Support.
Format of data transferring	ADO used technology to access data and is COM – Based.	ADO.NET uses xml for transmitting data to and from your database and web application.
Data provider	In ADO disconnected data provide by Record Set.	In ADO.NET disconnected data provide by DataSet and DataAdpter.
Tables	In ADO, Record Set, is like a single table or query result.	In ADO.NET DataSet, can contain multiple tables.
Client connection	In ADO client connection model is very poor. Client application needs to be connected to data-sever while working on the data.	In ADO.NET client disconnected as soon as the data is fetched or processed. DataSet is always disconnected.

Advantages of ADO.NET

Interoperability

ADO.NET applications can take advantage of the flexibility and broad acceptance of XML. Because XML is the format for transmitting datasets across the network, any component that can read the XML format can process data. In fact, the receiving component need not be an ADO.NET component at all: The transmitting component can simply transmit the dataset to its destination without regard to how the receiving component is implemented. The destination component might be a Visual Studio application or any other application implemented with any tool whatsoever. The only requirement is that the receiving component be able to read XML. As an industry standard, XML was designed with exactly this kind of interoperability in mind.

Maintainability

In the life of a deployed system, modest changes are possible, but substantial, architectural changes are rarely attempted because they are so difficult. That is unfortunate, because in a natural course of events, such substantial changes can become necessary. For example, as a deployed application becomes popular with users, the increased performance load might require architectural changes. As the performance load on a deployed application server grows, system resources can become scarce and response time or throughput can suffer. Faced with this problem, software architects can choose to divide the server's business-logic processing and user-

interface processing onto separate tiers on separate machines. In effect, the application server tier is replaced with two tiers, alleviating the shortage of system resources.

The problem is not designing a three-tiered application. Rather, it is increasing the number of tiers after an application is deployed. If the original application is implemented in ADO.NET using datasets, this transformation is made easier. Remember, when you replace a single tier with two tiers, you arrange for those two tiers to trade information. Because the tiers can transmit data through XML-formatted datasets, the communication is relatively easy.

Programmability

ADO.NET data components in Visual Studio encapsulate data access functionality in various ways that help you program more quickly and with fewer mistakes. For example, data commands abstract the task of building and executing SQL statements or stored procedures.

Similarly, ADO.NET data classes generated by the designer tools result in typed datasets. This in turn allows you to access data through typed programming. The code for the typed dataset is easier to read. It is also easier to write, because statement completion is provided.

Performance

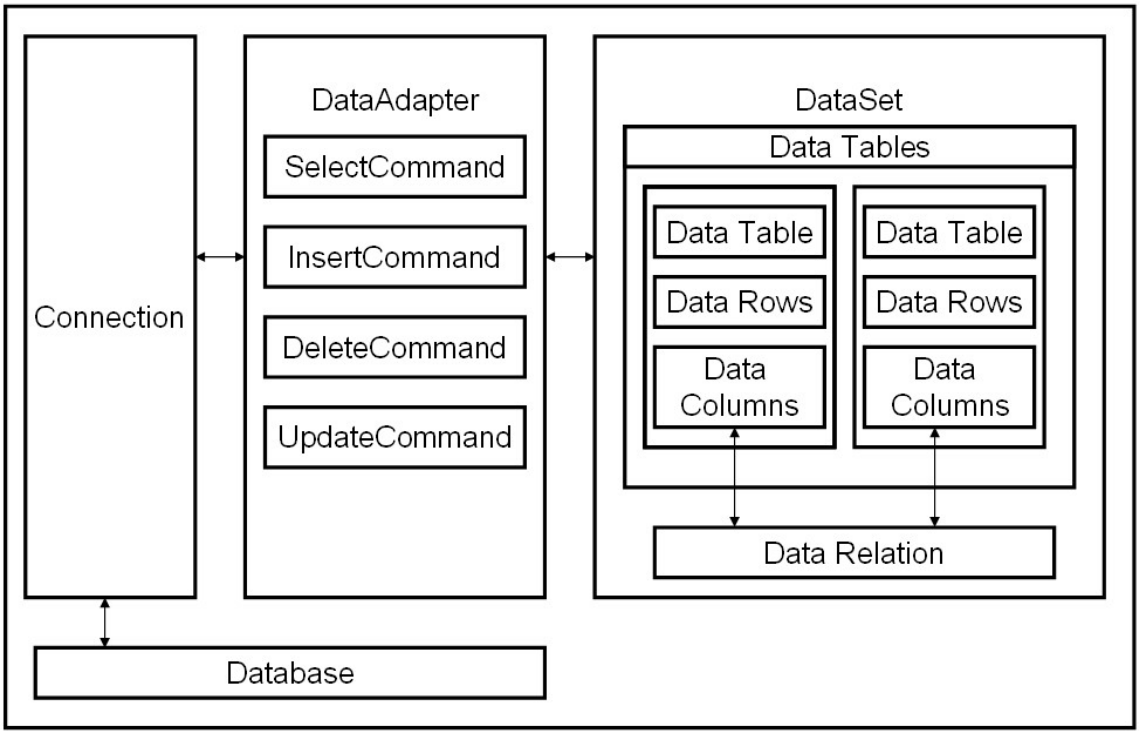
For disconnected applications, ADO.NET datasets offer performance advantages over ADO disconnected recordsets. When using COM marshalling to transmit a disconnected recordset among tiers, a significant processing cost can result from converting the values in the recordset to data types recognized by COM. In ADO.NET, such data-type conversion is not necessary.

Scalability

Because the Web can vastly increase the demands on your data, scalability has become critical. Internet applications have a limitless supply of potential users. Although an application might serve a dozen users well, it might not serve hundreds —or hundreds of thousands — equally well. An application that consumes resources such as database locks and database connections will not serve high numbers of users well, because the user demand for those limited resources will eventually exceed their supply.

ADO.NET accommodates scalability by encouraging programmers to conserve limited resources. Because any ADO.NET application employs disconnected access to data, it does not retain database locks or active database connections for long durations.

ADO.NET Architecture



Data processing has traditionally relied primarily on a connection-based, two-tier model. As data processing increasingly uses multi-tier architectures, programmers are switching to a disconnected approach to provide better scalability for their applications.

The two main components of ADO.NET 3.0 for accessing and manipulating data are the .NET Framework data providers and the DataSet.

.NET Framework Data Providers

The .NET Framework Data Providers are components that have been explicitly designed for data manipulation and fast, forward-only, read-only access to data. The **Connection** object provides connectivity to a data source. The **Command** object enables access to database commands to return data, modify data, run stored procedures, and send or retrieve parameter information. The **DataReader** provides a high-performance stream of data from the data source. Finally, the **DataAdapter** provides the bridge between the **DataSet** object and the data source. The **DataAdapter** uses **Command** objects to execute SQL commands at the data source to both load the **DataSet** with data and reconcile changes that were made to the data in the **DataSet** back to the data source.

The DataSet

The ADO.NET **DataSet** is explicitly designed for data access independent of any data source. As a result, it can be used with multiple and differing data sources, used with XML data, or used to manage data local to the application. The **DataSet** contains a collection of one or more DataTable objects consisting of rows and columns of data, and also primary key, foreign key,

constraint, and relation information about the data in the **DataTable** objects. The following diagram illustrates the relationship between a .NET Framework data provider and a **DataSet**.

Selecting a DataReader or a DataSet

When you decide whether your application should use a **DataReader** or a **DataSet**, consider the type of functionality that your application requires. Use a **DataSet** to do the following:

- Cache data locally in your application so that you can manipulate it. If you only need to read the results of a query, the **DataReader** is the better choice.
- Remote data between tiers or from an XML Web service.
- Interact with data dynamically such as binding to a Windows Forms control or combining and relating data from multiple sources.
- Perform extensive processing on data without requiring an open connection to the data source, which frees the connection to be used by other clients.

If you do not require the functionality provided by the **DataSet**, you can improve the performance of your application by using the **DataReader** to return your data in a forward-only, read-only manner. Although the **DataAdapter** uses the **DataReader** to fill the contents of a **DataSet**, by using the **DataReader**, you can boost performance because you will save memory that would be consumed by the **DataSet**, and avoid the processing that is required to create and fill the contents of the **DataSet**.

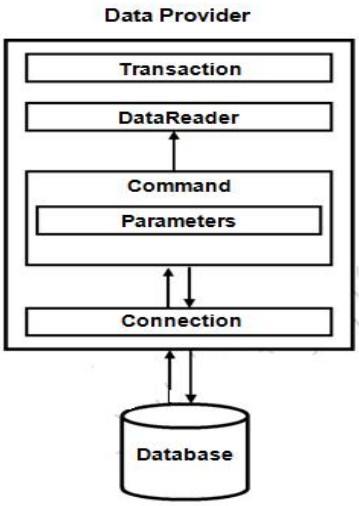
Connected Architecture of ADO.NET

The architecture of ADO.net, in which connection must be opened to access the data retrieved from database is called as connected architecture. Connected architecture was built on the classes connection, command, datareader and transaction.

Connection : in connected architecture also the purpose of connection is to just establish aconnection to database and it self will not transfer any data.

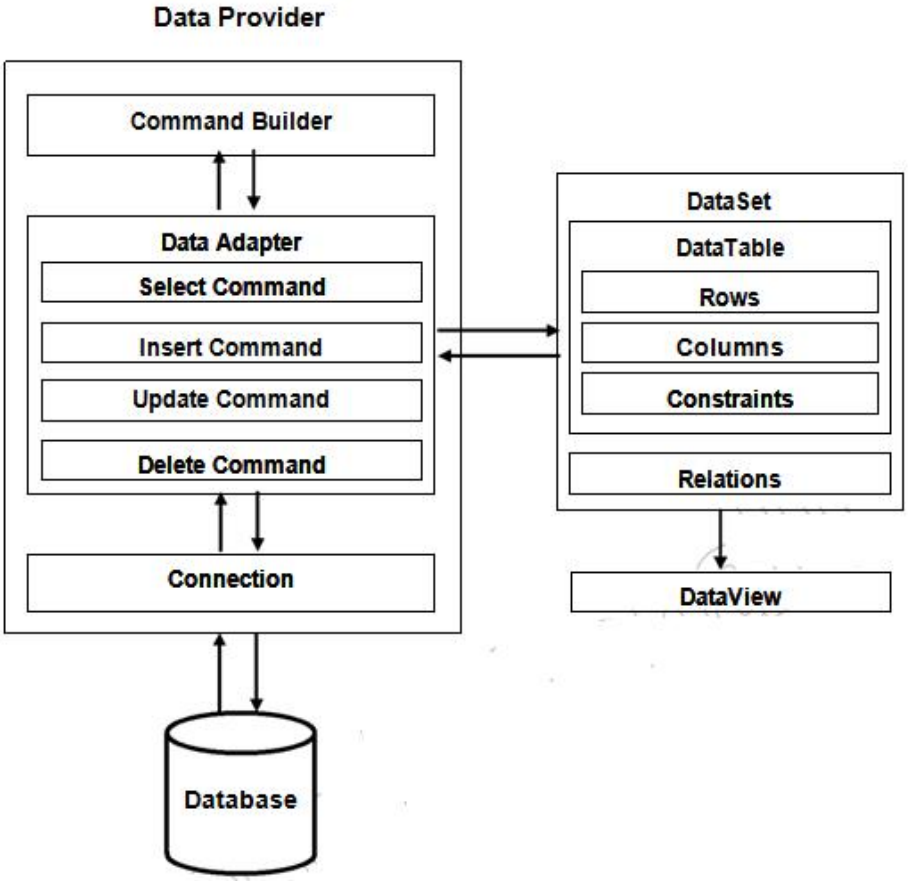
DataReader : DataReader is used to store the data retrieved by command object and make it available for .net application. Data in DataReader is read only and within the DataReader you can navigate only in forward direction and it also only one record at a time.

To access one by one record from the DataReader, call **Read()** method of the DataReader whose return type is **bool.** When the next record was successfully read, the Read() method will return true and otherwise returns false.



Disconnected Architecture in ADO.NET

The architecture of ADO.net in which data retrieved from database can be accessed even when connection to database was closed is called as disconnected architecture. Disconnected architecture of ADO.net was built on classes connection, dataadapter, commandbuilder and dataset and dataview.



Connection : Connection object is used to establish a connection to database and connectionit self will not transfer any data.

DataAdapter : DataAdapter is used to transfer the data between database and dataset. It has commands like select, insert, update and delete. Select command is used to retrieve data from database and insert, update and delete commands are used to send changes to the data in dataset to database. It needs a connection to transfer the data.

CommandBuilder : by default dataadapter contains only the select command and it doesn'tcontain insert, update and delete commands. To create insert, update and delete commands for the dataadapter, commandbuilder is used. It is used only to create these commands for the dataadapter and has no other purpose.

DataSet : Dataset is used to store the data retrieved from database by dataadapter and make it available for .net application.

To fill data in to dataset **fill()** method of dataadapter is used and has the following syntax.

Da.Fill(Ds,"TableName");

When fill method was called, dataadapter will open a connection to database, executes select command, stores the data retrieved by select command in to dataset and immediately closes the connection.

As connection to database was closed, any changes to the data in dataset will not be directly sent to the database and will be made only in the dataset. To send changes made to data in dataset to the database, **Update()** method of the dataadapter is used that has the following syntax.

Da.Update(Ds,"Tablename");

When Update method was called, dataadapter will again open the connection to database, executes insert, update and delete commands to send changes in dataset to database and immediately closes the connection. As connection is opened only when it is required and will be automatically closed when it was not required, this architecture is called disconnected architecture.

A dataset can contain data in multiple tables.

DataView : DataView is a view of table available in DataSet. It is used to find a record, sort the records and filter the records. By using dataview, you can also perform insert, update and delete as in case of a DataSet.

DataReader is **Connected Architecture** since it keeps the connection open until all rows are fetched one by one

DataSet is **DisConnected Architecture** since all the records are brought at once and there is no need to keep the connection alive

Difference between Connected and disconnected architecture

Connected (DataReader)	Disconnected (DataSet)
It is connection oriented.	It is disconnection oriented.
Connected methods gives faster performance	Disconnected get low in speed and performance.
connected can hold the data of single table	disconnected can hold multiple tables of data
connected you need to use a read only forward only data reader	disconnected you cannot
Data Reader can't persist the data	Data Set can persist the data
It is Read only, we can't update the data.	We can update data

ADO.NET Components

There are two major components of ADO.NET:

1. DataSet
2. DataProvider

DataSet : represents either an entire database or a subset of database. It can contain tables and relationships between those tables.

Data Provider is a collection of components like Connection, Command, DataReader, DataAdapter objects and handles communication with a physical data store and the dataset.

Data Provider

- The data provider is responsible for providing and maintaining the connection to the database.
- It is a set of classes that can be used for communicating with database, and holding/manipulating data
- The DataProvider connects to the data source on behalf of ADO.NET.
- The data source can be Microsoft SQL Server or Oracle database and OLEDB data provider.
- The data provider components are specific to data source.

The following lists the .NET Framework data providers that are included in the .NET Framework.

For SQL Server

- Imports System.Data.SqlClient namespace.
- It provides data access for Microsoft SQL Server.

For OLEDB

- Imports System.Data.OleDb namespace.
- It provides data sources exposed using OLEDB.
- We can use OLEDB for connect Microsoft Access

For ODBC

- Imports System.Data.Odbc namespace.
- It provides data sources exposed using ODBC

For Oracle

- Imports System.Data.OracleClient namespace.
- It provides data access for oracle.

Data Provider's common set of classes for all DataSource.

1. Connection
2. Command
3. DataAdapter
4. DataReader

Conncetion

- It establishes or connects a connection to the data source.
- In SQL Server the connection can establish using SqlConnection object.

Command

- Fires SQL commands or perform some action on the data source, such as insert, update, delete.
- In SQL Server command can fires using SqlCommand object.

DataAdapter

- It's a bride between Data source and DataSet object for transferring data.
- In SQL Server the Data Adapter can create using SqlDataAdapter object

DataReader

- Used when large list of results one record at a time.
- It reads records in a read-only, forward-only mode.
- In SQL Server the datareader can create using SqlDataReader object

Use of Server Explorer

Server Explorer/Database Explorer is the server management console for Visual Studio. Use this window to open data connections and to log on to servers and explore their system services.

Use **Server Explorer/Database Explorer** to view and retrieve information from all of the databases you are connected to. You can do the following:

- List database tables, views, stored procedures, and functions
- Expand individual tables to list their columns and triggers
- Right-click a table to perform actions, such as showing the table's data or viewing the table's definition, from its shortcut menu.

To access **Server Explorer/Database Explorer**, choose **Server Explorer** or **Database Explorer** on the **View** menu. To make the **Server Explorer**/**Database Explorer** window close automatically when not in use, choose **Auto Hide** on the **Window** menu.

Namespaces

System.Data	The System.Data namespace provides access to classes that represent the ADO.NET architecture. ADO.NET lets you build components that efficiently manage data from multiple data sources. Consists of the classes that constitute the ADO.NET architecture, which is the primary data access method for managed applications. The ADO.NET architecture enables you to build components that efficiently manage data from multiple data sources. ADO.NET also provides the tools to request, update, and reconcile data in distributed applications.
System.Data.OleDb	The System.Data.OleDb namespace is the.NET Framework Data Provider for OLE DB. Classes that make up the .NET Framework Data Provider for OLE DB-compatible data sources. These classes allow you to connect to an OLE DB data source, execute commands against the source, and read the results.
System.Data.SqlClient	The System.Data.SqlClient namespace is the.NET Framework Data Provider for SQL Server. Classes that make up the .NET Framework Data Provider for SQL Server, which allows you to connect to SQL Server 7.0, execute commands, and read results. The **System.Data.SqlClient** namespace is similar to the **System.Data.OleDb** namespace, but is optimized for access to SQL Server 7.0 and later.
System.Data.SqlTypes	The System.Data.SqlTypes namespace provides classes for native data types in SQL Server. These classes provide a safer, faster alternative to the data types provided by the .NET Framework common language

	runtime (CLR). Using the classes in this namespace helps prevent type conversion errors caused by loss of precision. Because other data types are converted to and from **SqlTypes** behind the scenes, explicitly creating and using objects within this namespace also yields faster code.

Connection Object

A primary function of any database application is connecting to a data source and retrieving the data that it contains. The .NET Framework data providers of ADO.NET serve as a bridge between an application and a data source, allowing you to execute commands as well as to retrieve data by using a **DataReader** or a **DataAdapter**. A key function of any database application is the ability to update the data that is stored in the database

In ADO.NET you use a **Connection** object to connect to a specific data source by supplying necessary information in a connection string. The **Connection** object you use depends on the type of data source.

Each .NET Framework data provider included with the .NET Framework has a **Connection** object: the .NET Framework Data Provider for OLE DB includes an OleDbConnection object, the .NET Framework Data Provider for SQL Server includes a SqlConnection object, the .NET Framework Data Provider for ODBC includes an OdbcConnection object, and the .NET Framework Data Provider for Oracle includes an OracleConnection object.

Steps to connect database:

1. Create Database.
2. Start writing connection string in VB.Net.
3. Set the provider.
4. Specify the data source.

Connection object have ConnectionString property. Depends on the parameter specified in the Connection String, ADO.Net Connection Object connect to the specified Database.

Properties :

ConnectionString : Gets/sets the connection string to open a database.

Database : Gets the name of the database to open

DataSource : Gets the name of the SQL Server to use.

State : Gets the connection's current state

Methods :

Close : Closes the connection to the data provider.

Open : Opens a database connection.

Command Object :

- It’s depended on Connection Object.

- Command objects are used to execute commands to a database across a data connection.
- The Command object in ADO.NET executes SQL statements and stored procedures against the data source specified in the connection object.
- The Command objects has a property called Command Text, which contains a String (Query) value that represents the command that will be executed in the Data Source.

There are many ways to initialize Command object:

For Example

```
Dim con As New SqlConnection
Dim cmd As SqlCommand
Dim str1 As String

Str1 = "Insert into stud values('"+ TextBox1.Text +"', '"+ TextBox1.Text +"')
con.Open()
cmd = New SqlCommand(str1, con)
cmd.ExecuteNonQuery()
con.close()
```

OR

```
Dim con As New SqlConnection
Dim cmd As SqlCommand

con.Open()
cmd.Connection = con
cmd.CommandText = "Insert into stud values('"+ TextBox1.Text +"', '"+ TextBox1.Text +"')"
cmd.ExecuteNonQuery()
con.close()
```

Properties

Property	Meaning
CommandText	Gets/sets the SQL statement (or stored procedure) for this command to execute.
CommandType	Gets/sets the type of the CommandText property (typically set to text for SQL).
Connection	Gets/sets the SqlConnection to use.
Parameters	Gets the command parameters.

Methods

Methods	Meaning
ExecuteNonQuery	Executes a non-row returning SQL statement, returning the number of affected rows.
ExecuteReader	Creates a data reader using the command
ExecuteScalar	Executes the command and returns the value in the first column in the first row of the result.

DataAdapter Object :

The **SqlDataAdapter**, serves as a bridge between a DataSet and SQL Server for retrieving and saving data. The **SqlDataAdapter** provides this bridge by mapping Fill, which changes the data in the DataSet to match the data in the data source, and Update, which changes the data in the data source to match the data in the DataSet, using the appropriate Transact-SQL statements against the data source. The update is performed on a by-row basis. For every inserted, modified, and deleted row, the Update method determines the type of change that has been performed on it (**Insert**, **Update**, or **Delete**). Depending on the type of change, the **Insert**, **Update**, or **Delete** command template executes to propagate the modified row to the data source.

When the **SqlDataAdapter** fills a DataSet, it creates the necessary tables and columns for the returned data if they do not already exist. **SqlDataAdapter** is used in conjunction with SqlConnection and SqlCommand to increase performance when connecting to a SQL Server database.

The **SqlDataAdapter** also includes the SelectCommand, InsertCommand, DeleteCommand, UpdateCommand properties to facilitate the loading and updating of data.

When an instance of **SqlDataAdapter** is created, the read/write properties are set to initial values. For a list of these values, see the **SqlDataAdapter** constructor.

The InsertCommand, DeleteCommand, and UpdateCommand are generic templates that are automatically filled with individual values from every modified row through the parameters mechanism.

For every column that you propagate to the data source on Update, a parameter should be added to the **InsertCommand**, **UpdateCommand**, or **DeleteCommand**. The SourceColumn property of the DbParameter object should be set to the name of the column. This setting indicates that the value of the parameter is not set manually, but is taken from the particular column in the currently processed row.

Properties

Name	Meaning
DeleteCommand	Gets or sets a Transact-SQL statement or stored procedure to delete records from the data set.
InsertCommand	Gets or sets a Transact-SQL statement or stored procedure to insert new records into the data source.
SelectCommand	Gets or sets a Transact-SQL statement or stored procedure used

	to select records in the data source.
TableMappings	Gets a collection that provides the master mapping between a source table and a DataTable. (Inherited from DataAdapter.)
UpdateCommand	Gets or sets a Transact-SQL statement or stored procedure used to update records in the data source.

Methods	**Meaning**
Fill	Adds or updates rows in a data set to match those in the data source. Creates a table named "Table" by default
Update	Updates the data store by calling the INSERT, UPDATE, or DELETE statements for each inserted, updated, or deleted row in the given dataset.

For Example :

```
Dim da As SqlDataAdapter
Dim ds As New DataSet
str1 = "select * from stud"
da = New SqlDataAdapter(str1, con)
da.Fill(ds)
```

DataSet Object

- ADO.NET caches data locally on the client and store that data into DataSet.
- The dataset is a disconnected, I-memory representation of data.
- It's not exact copy the database.
- It can be considered as a local copy of the some portions of the database.
- The DataSet contains a collection of one or more DataTable objects made up of rows and columns of data.
- Tables can be identified in DataSet using DataSet's Tables property.
- It also contains primary key, foreign key, constraint and relation information about the data in the DataTable objects.
- DataSet are also fully XML-featured.
- Whaterer operations are made by the user it is stored temporary in the DataSet, when the use of this DataSet is finished, changes can be made back to the central database for updating.
- DataSet doesn't "know" where the data it contains came from and if fact it can contain data from multiple sources.
- The DataSet is populated DataAdapter's Fill method.

The **DataSet** is a major component of the ADO.NET architecture. The **DataSet** consists of a collection of DataTable objects that you can relate to each other with Data Relation objects. You can also enforce data integrity in the **DataSet** by using the UniqueConstraint and ForeignKeyConstraint objects. For further details about working with **DataSet** objects.

Whereas DataTable objects contain the data, the DataRelationCollection allows you to navigate though the table hierarchy. The tables are contained in a DataTableCollection accessed through the Tables property. When accessing DataTable objects, note that they are conditionally case sensitive.

For example, if one DataTable is named "mydatatable" and another is named "Mydatatable", a string used to search for one of the tables is regarded as case sensitive. However, if "mydatatable" exists and "Mydatatable" does not, the search string is regarded as case insensitive. For more information about working with DataTable objects.

A **DataSet** can read and write data and schema as XML documents. The data and schema can then be transported across HTTP and used by any application, on any platform that is XML-enabled. You can save the schema as an XML schema with the WriteXmlSchema method, and both schema and data can be saved using the WriteXml method. To read an XML document that includes both schema and data, use the ReadXml method.

In a typical multiple-tier implementation, the steps for creating and refreshing a **DataSet**, and in turn, updating the original data are to:

1. Build and fill each DataTable in a **DataSet** with data from a data source using a DataAdapter.
2. Change the data in individual DataTable objects by adding, updating, or deleting DataRow objects.
3. Invoke the GetChanges method to create a second **DataSet** that features only the changes to the data.
4. Call the Update method of the DataAdapter, passing the second **DataSet** as an argument.
5. Invoke the Merge method to merge the changes from the second **DataSet** into the first.
6. Invoke the AcceptChanges on the **DataSet**. Alternatively, invoke RejectChanges to cancel the changes.

Properties

Name	Description
Relations	Get the collection of relations that link tables and allow navigation from parent tables to child tables.
Tables	Gets the collection of tables contained in the DataSet.

For Example :

```
Dim da As SqlDataAdapter
Dim ds As New DataSet
str1 = "select * from stud"
da = New SqlDataAdapter(str1, con)
da.Fill(ds)
DataGridView1.DataSource = ds.Tables(0)
```

DataReader Object

Provides a way of reading a forward-only stream of rows from a SQL Server database

To create a **SqlDataReader**, you must call the ExecuteReader method of the SqlCommand object, instead of directly using a constructor.

While the **SqlDataReader** is being used, the associated SqlConnection is busy serving the **SqlDataReader**, and no other operations can be performed on the SqlConnection other than closing it. This is the case until the Close method of the **SqlDataReader** is called. For example, you cannot retrieve output parameters until after you call Close.

Changes made to a result set by another process or thread while data is being read may be visible to the user of the **SqlDataReader**. However, the precise behavior is timing dependent.

IsClosed and RecordsAffected are the only properties that you can call after the **SqlDataReader** is closed. Although the RecordsAffected property may be accessed while the **SqlDataReader** exists, always call Close before returning the value of RecordsAffected to guarantee an accurate return value.

Properties

Name	Description
Connection	Gets the SqlConnection associated with the SqlDataReader.
IsClosed	Retrieves a Boolean value that indicates whether the specified SqlDataReader instance has been closed. (Overrides DbDataReader.IsClosed.)
Item	Overloaded. Gets the value of a column in its native format.

Methods

Name	Description
Close	Closes the SqlDataReader object. (Overrides DbDataReader.Close().)
Read	Advances the SqlDataReader to the next record. (Overrides DbDataReader.Read().)

```
Dim reader As SqlDataReader
sql = " Select * from stud"
Try
        con.Open()
        cmd = New SqlCommand(sql, con)
        reader = cmd.ExecuteReader()
        While reader.Read()
                MsgBox(reader.Item(0) & "  -  " & reader.Item(1))
        End While
        reader.Close()
        cmd.Dispose()
```

```
        con.Close()
Catch ex As Exception
        MsgBox("Can not open connection ! ")
End Try
```

DataGridView Control

The **DataGridView** control provides a powerful and flexible way to display data in a tabular format. You can use the **DataGridView** control to show read-only views of a small amount of data, or you can scale it to show editable views of very large sets of data.

You can extend the **DataGridView** control in a number of ways to build custom behaviors into your applications. For example, you can programmatically specify your own sorting algorithms, and you can create your own types of cells. You can easily customize the appearance of the **DataGridView** control by choosing among several properties. Many types of data stores can be used as a data source, or the **DataGridView** control can operate with no data source bound to it.

The **DataGridView** control provides a customizable table for displaying data. The **DataGridView** class allows customization of cells, rows, columns, and borders through the use of properties such as DefaultCellStyle, ColumnHeadersDefaultCellStyle, CellBorderStyle, and GridColor.

You can use a **DataGridView** control to display data with or without an underlying data source. Without specifying a data source, you can create columns and rows that contain data and add them directly to the **DataGridView** using the Rows and Columns properties. You can also use the Rows collection to access DataGridViewRow objects and the DataGridViewRow.Cells property to read or write cell values directly. The Item indexer also provides direct access to cells.

As an alternative to populating the control manually, you can set the DataSource and DataMember properties to bind the **DataGridView** to a data source and automatically populate it with data. For more information, see Displaying Data in the Windows Forms DataGridView Control.

When working with very large amounts of data, you can set the VirtualMode property to **true** to display a subset of the available data. Virtual mode requires the implementation of a data cache from which the **DataGridView** control is populated. For more information, see Data Display Modes in the Windows Forms DataGridView Control.

For additional information about the features available in the **DataGridView** control, see DataGridView Control (Windows Forms). The following table provides direct links to common tasks.

Explain the steps to bind DataGridView.

Step : 1	Take New VB.NET Project
Step : 2	Add database in your project (Named : dbemp.mdf) Add New Table in database file (Named : emp) from the Server Explorer Window Add some records in that table
Step : 3	Now add emp table to the DBEmpDataSet.xsd file

Step : 4	Add DataGridView control on the form
Step : 5	Select the Choose Data Source In that click on Other Data Source Project Data Source DBEmpDataSet Emp ← Table Name
Step : 6	At the end, Run the project

Binding Data Grids

As we've already seen, you can use data grids to display entire data tables. To bind a data grid to a table, you can set the data grid's **DataSource** property (usually to a dataset, such as **dsDataSet**) and **DataMember** property (usually to text naming a table like "authors"). At run time, you can set both of these properties at once with the built-in data grid method **SetDataBinding** (data grids are the only controls that have this method):

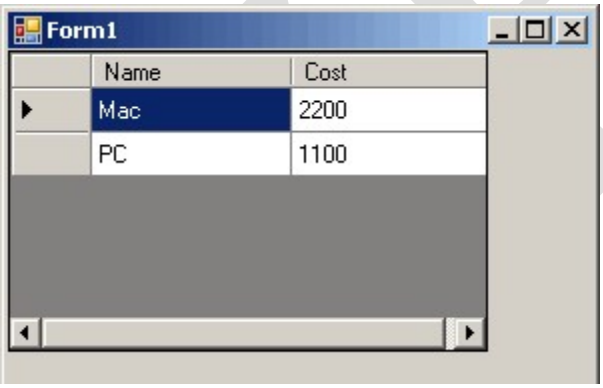
```
DataGrid1.SetDataBinding(dsDataSet, "authors")
```

You can use these data sources with the data grid's **DataSource** property:

- **DataTable** objects
- **DataView** objects
- **DataSet** objects
- **DataViewManager** objects
- single dimension arrays

To determine which cell was selected by the user, use the **CurrentCell** property. You can change the value of any cell using the **Item** property, which can take either the row or column indexes of the cell. And you can use the **CurrentCell Changed** event to determine when the user selects another cell.

Example



First, you should add a DataGridView collection to your Windows Forms application by double-clicking on the control name in the Visual Studio designer panel. After you add the control, you can add the Load event on the form, which you can create from the Form's event pane.

```
Public Class Form1
   Private Sub Form1_Load(ByVal sender As System.Object, _
                          ByVal e As System.EventArgs) Handles MyBase.Load
```

```
        DataGridView1.DataSource = GetDataTable()
    End Sub

    Private Function GetDataTable() As DataTable
        Return New DataTable()
    End Function
End Class
```

This event handler is executed when the program starts up and when the DataGridView control is displayed. The Form1_Load autogenerated subroutine calls into the GetDataTable function, which would return a DataTable from your database in SQL Server.

Assigning the DataSource property on DataGridView does not copy any data, but instead allows the DataGridView to read in the DataTable and display all its contents on the screen in grid form. This is often the most efficient way to populate DataGridView.

Explain the steps to bind the application with the Database in ADO .net using Binding Navigator control

Step : 1	Take New VB.NET Project
Step : 2	Add two labels and two textbox on the form1
Step : 3	Add database in your project (Named : dbemp.mdf) Add New Table in database file (Named : emp) from the Server Explorer Window Add some records in that table
Step : 4	Now add emp table to the DBEmpDataSet.xsd file
Step : 5	After creating table Add BindingNavigator1 and BindingSource1 Control to the form
Step : 6	Now set the property of BindingSource1 control DataSource = DBEmpDataSet DateMember = emp
Step : 7	Now Set the property of BindingNavigator1 control Right click on BindingNavigator1 control and select Edit Items… Set BindingSource = BindingSource1 Then ok
Step : 8	Now Set the property of each textbox control Select TextBox1 and set the DataBinding property Click on Advanced property. The Formatting and Advanced Binding Window is appeared. In this window set Binding = BindingSource1 – empno Similarly, set all the textbox control
Step : 9	At the end, Run the project

Properties of BindingSource Control:

1) Datasource: Gets or sets the data source that the connector binds to.

   Syntax: instance.DataSource = *value*

2) Datamember: Gets or sets the specific list in the data source to which the connector currently binds to.

   Syntax: instance.DataMember = *value*

Properties of BindingNavigator Control:

1) BindingSource: Gets or sets the System.Windows.Forms.BindingSource component that is the source of data.

   Syntax: instance.BindingSource = *value*

Explain use of "ExecuteScaler" , "ExecuteNonQuery" and "ExecuteReader" method in detail.

ExecuteNonQuery

ExecuteNonQuery() is one of the most frequently used method in SqlCommand Object and is used for executing statements that do not return result set. ExecuteNonQuery() performs Data Definition tasks as well as Data Manipulation tasks also. The Data Definition tasks like creating Stored Procedures and Views perform by ExecuteNonQuery() . Also Data Manipulation tasks like Insert , Update and Delete perform by ExecuteNonQuery().

The following example shows how to use the method ExecuteNonQuery() through SqlCommand Object.

```
sql = "Insert into stud values('"+ TextBox1.Text +"', '"+ TextBox1.Text +"')
Try
        con.Open()
        cmd = New SqlCommand(Sql, con)
        cmd.ExecuteNonQuery()
        cmd.Dispose()
        con.Close()
        MsgBox(" ExecuteNonQuery in SqlCommand executed !!")
Catch ex As Exception
        MsgBox("Can not open connection ! ")
End Try
```

ExecuteReader

ExecuteReader() in SqlCommand Object send the SQL statements to Connection Object and populate a SqlDataReader Object based on the SQL statement. When the ExecuteReader method in SqlCommand Object execute, it instantiate a SqlClient.SqlDataReader Object.

The SqlDataReader Object is a stream-based, forward-only, read-only retrieval of query results from the Data Source, which do not update the data. The SqlDataReader cannot be created directly from code, they created only by calling the ExecuteReader method of a Command Object.

```
Dim reader As SqlDataReader
sql = " Select * from stud"
Try
        con.Open()
        cmd = New SqlCommand(sql, con)
        reader = cmd.ExecuteReader()
```

```
        While reader.Read()
            MsgBox(reader.Item(0) & "  -  " & reader.Item(1))
        End While
        reader.Close()
        cmd.Dispose()
        con.Close()
Catch ex As Exception
        MsgBox("Can not open connection ! ")
End Try
```

ExecuteScaler

ExecuteScalar() in SqlCommand Object is used for get a single value from Database after its execution. It executes SQL statements or Stored Procedure and returned a scalar value on first column of first row in the Result Set. If the Result Set contains more than one columns or rows , it takes only the first column of first row, all other values will ignore. If the Result Set is empty it will return a Null reference.

It is very useful to use with aggregate functions like Count(*) or Sum() etc. When compare to ExecuteReader() , ExecuteScalar() uses fewer System resources.

```
sql = " Select Count(*) from stud"
Try
        con.Open()
        cmd = New SqlCommand(sql, con)
        Dim count As Int32 = Convert.ToInt32(cmd.ExecuteScalar())
        cmd.Dispose()
        con.Close()
        MsgBox(" No. of Rows " & count)
Catch ex As Exception
        MsgBox("Can not open connection ! ")
End Try
```

What are the four common SQL commands used to retrieve and modify data in a SQL Database? Also explain each of them.

Four Common SQL Commands :

1. Select
2. Insert
3. Update
4. Delete

Select

Retrives data from one or more tables or views.
Systax :

```
        Select * from TableName
```

Select Columnname1, Columnname2 from TableName
Select * from TableName Where <Condition>
Select * from TableName Order by Columnnane

Example :

Select * from stud
Select Sno,Sname from stud
Select * from stud where city = "Anand"
Select * from stud order by total

Insert

Create a new row and inserts values into specified columns

Syntax:

Insert into TableName values(Value1,Value2,…)

Example :

Insert into stud values (1,'ABC')

Update

Changes the values of individual columns in one or more existing rows in a table.

Syntax :

Update TableName set Columnname = value, ….
Update TableName set Columnname = value where <condition>

Example :

Update stud set sname = 'xyz' where sno = 1

Delete

Delete one or more rows from a table

Syntax :

Delete from TableName
Delete from TableName where <condition>

Example :

Delete from stud
Delete from stud where sno = 5

Additional Questions:

1. Explain the step, how can we retrieve data in DataSet?

To load data from a database into a DataSet, follow these steps:

1. Start Visual Studio .NET.
2. Create a new Console Application project in Visual VB.NET.
3. Make sure that the project references the System and System.Data namespaces.
4. Use the imports statement on System.Data.SqlClient namespaces so that you are not required to qualify declarations from these namespaces later in your code. You must use these statements before any other declarations.

```
imports System.Data.SqlClient
```

5. The first step to get data from the database to the DataSet is to establish a database connection, which requires a System.Data.SqlClient.SqlConnection object and a connection string. The connection string in the code to follow connects a SQL Server server that is located on the local computer (the computer where the code is running). You must modify this connection string as appropriate for your environment. After the SqlConnection object is created, call the Open method of that object to establish the actual database link.

```
Dim sConnectionString as String
sConnectionString = "Password=myPassword;User ID=myUserID ;Initial
Catalog=Pubs; Data Source=(local)"
Dim objConn as new SqlConnection
objConn.ConnectionString=sConnectionString
objConn.Open()
```

6. Create a DataAdapter object, which represents the link between the database and your DataSet object. You can specify SQL or another type of command that is used to retrieve data as part of the constructor object of the DataAdapter. This sample uses a SQL statement that retrieves records from the Authors table in the Pubs database.

```
Dim daAuthors as SqlDataAdapter= new SqlDataAdapter("Select * From
Authors", objConn)
```

7. You must declare and create an instance of a DataSet object; at which time you can supply a name for the entire DataSet before you can start to load any data. The name may contain several distinct tables.

```
Dim dsPubs as DataSet = new DataSet("Pubs");
```

8. The SqlDataAdapter class provides two methods, Fill and FillSchema, that are crucial to loading this data. Both of these methods load information into a DataSet. Fill loads the data itself, and FillSchema loads all of the available metadata about a particular table (such as column names, primary keys, and constraints). A good way to handle the data loading is to run FillSchema followed by Fill. For example:

```
daAuthors.FillSchema(dsPubs,SchemaType.Source, "Authors")
daAuthors.Fill(dsPubs,"Authors")
```

If you only use Fill, you can only load the basic metadata that is required to describe the column names and data types. The Fill method does not load primary key information. To change this default behavior, you can set the MissingSchemaAction property of the DataAdapter object to MissingSchemaAction.AddWithKey, which loads the primary key metadata along with the default information. For example:

```
daAuthors.MissingSchemaAction = MissingSchemaAction.AddWithKey
daAuthors.Fill(dsPubs,"Authors")
```

9. The data is now available as an individual DataTable object within the Tables collection of the DataSet. If you specified a table name in the calls to FillSchema and Fill, you can use that name to access the specific table that you want.

```
Dim tblAuthors as DataTable
tblAuthors = dsPubs.Tables("Authors")
```

10. You can use a For Each loop to loop through all of the DataRow objects within the Rows collection of a DataTable. This gives you access to each row of the table. You can access columns by name or by positional index (with '0' being the first column position). For example:

```
Dim drCurrent as DataRow

foreach (drCurrentin tblAuthors.Rows)
loop
Console.WriteLine("{0} {1}",
drCurrent("au_fname").ToString(),
drCurrent("au_lname").ToString())
End Loop
Console.ReadLine()
```

11. Save your project. On the Debug menu, click Start to run your project and make sure that it works.

2. Explain public methods of SqlCommand objects.

Method	Description
BeginExecuteNonQuery()	It is used to Initiate the asynchronous execution of the SQL statement described by this SqlCommand.
Cancel()	It tries to cancel the execution of a SqlCommand.
Clone()	It creates a new SqlCommand object that is a copy of the current instance.
CreateParameter()	It creates a new instance of a SqlParameter object.
ExecuteReader()	It is used to send the CommandText to the Connection and

	builds a SqlDataReader.
ExecuteXmlReader()	It is used to send the CommandText to the Connection and builds an XmlReader object.
ExecuteScalar()	It executes the query and returns the first column of the first row in the result set. Additional columns or rows are ignored.
Prepare()	It is used to create a prepared version of the command by using the instance of SQL Server.
ResetCommandTimeout()	It is used to reset the CommandTimeout property to its default value.

3. What are the features of ADO.Net?

- **Batch Update**

Batch update can provide a huge improvement in the performance by making just one round trip to the server for multiple batch updates, instead of several trips if the database server supports the batch update feature.

- **Data Paging**

Command object has an execute method called ExecutePageReader. This method takes three parameters - CommandBehavior, startIndex, and pageSize. So, if you want to get rows from 101 - 200, you can simply call this method with start index as 101 and page size as 100.

- **Connection Details**

You can get more details about a connection by setting Connection's StatisticsEnabled property to True. The Connection object provides two new methods - RetrieveStatistics and ResetStatistics. The RetrieveStatistics method returns a HashTable object filled with the information about the connection such as data transferred, user details, curser details, buffer information and transactions.

- **DataTable's Load and Save Methods**

In previous version of ADO.NET, only DataSet had Load and Save methods. The Load method can load data from objects such as XML into a DataSet object and Save method saves the data to a persistent media. Now DataTable also supports these two methods.

You can also load a DataReader object into a DataTable by using the Load method.

- **New Data Controls**

In Toolbox, you will see these new controls - DataGridView, DataConnector, and DataNavigator. Now using these controls, you can provide navigation (paging) support to the data in data bound controls.

- **DbProvidersFactories Class**

This class provides a list of available data providers on a machine. You can use this class and its members to find out the best suited data provider for your database when writing a database independent application.

- **Customized Data Provider**

By providing the factory classes now ADO.NET extends its support to custom data provider. Now you don't have to write a data provider dependent code. You use the base classes of data provider and let the connection string does the trick for you.

- **DataReader's New Execute Methods**

Now command object supports more execute methods. Besides old ExecuteNonQuery, ExecuteReader, ExecuteScaler, and ExecuteXmlReader, the new execute methods are ExecutePageReader, ExecuteResultSet, and ExecuteRow.

4. Which namespaces are required to enable the use of databases in ADO .net?

The following namespaces are required to enable the use of databases in ASP.NET pages:

- The System.Data namespace.
- The System.Data.OleDb namespace (to use any data provider, such as Access, Oracle, or SQL)
- The System.Data.SQLClient namespace (specifically to use SQL as the data provider)

5. What is connection string? Explain in brief.

Connection String is a normal String representation which contains Database connection information to establish the connection between Database and the Application. The Connection String includes parameters such as the name of the driver, Server name and Database name, as well as security information such as user name and password. Data providers use a connection string containing a collection of parameters to establish the connection with the database.

The .NET Framework provides mainly three data providers: Microsoft SQL Server, OLEDB and ODBC.

6. Which properties are used to bind a DataGridView control?

- The DataSource property and the DataMember property are used to bind a DataGridView control.
- Here, DataSource refers the object which binds DataGrid View, that's generally DataSet/DataTable/ or any object.
- DataMember refers to Table name.

7. Differentiate between Radio button and Checkbox.

- In Check box, you can select multiple options. In Option Button (Radio button) you can select one option.
- Radio buttons are circular and check boxes are square. When you click on a radio button, a little dot appears in the middle of the circle when you click the check box little check mark appears in the middle of the square.
- If you click a different choice, the dot will move to the center of the new circle that you have selected in the radio button when you click the check box another check appears the new box, the first check doesn't change.

8. Differentiate between Label and LinkLabel.

- A LinkLabel control is a label control that can display a *hyperlink*. A LinkLabel control is inherited from the *Label* class so it has all the functionality provided by the Windows Forms Label control.A **Label** control lets you place descriptive text, where the text does not need to be changed by the user. The **Label** class is defined in the System.Windows.Forms namespace.
- LinkLabel control does not participate in *user input or capture mouse or keyboard events*.

9. Differentiate between Listbox and Combobox.

- In listbox we can only select item. In combobox we can write/search and select item.
- We can see multiple items in listbox. We can see only single item in combobox.
- We have no other styles for listbox.

We have 3 styles in combo box.

A)drop down combo

B)simple combo

C)drop down list

- Listbox is only listed items box.combobox is combination of textbox and listbox.

- In listbox we have scroll down and scroll up facility. In combobox we have only dropdown facility.
- We can use checkbox with in listbox. We can't use checkbox within a combobox.
- Listbox is much easier to handle. Combobox is not easier as well as listbox to handle.
- We can't add image item in listbox.We can add image item in combobox.

10. Differentiate between Listbox and CheckedListBox.

- The Windows Forms CheckedListBox control extends the ListBox control. It does almost everything that a list box does and also can display a check mark next to items in the list.
- Other differences between the two controls are that checked list boxes only support DrawMode.Normal; and that checked list boxes can only have one item or none selected. Note that a selected item appears highlighted on the form and is not the same as a checked item.

11. What do you mean by debugging?

Debugging is the routine process of locating and removing computer program bugs, errors or abnormalities, which is methodically handled by software programmers via debugging tools. Debugging checks, detects and corrects errors or bugs to allow proper program operation according to set specifications.

Debugging is also known as debug.

12. Explain the function of STEP INTO and STEP OVER.

Step Into	Executes one line of code and then pauses; clicking again on this option will execute the next line. When the interpreter comes across a subroutine call it will *step into* and execute each line within the subroutine separately.
Step Over	Similar to 'Step Into' except when encountering subroutine calls. When coming across subroutine calls the interpreter will *step over* all the code within the subroutine in one go.
Step Out	All lines of code in the current subroutine are executed and the execution point is paused once again on the line after the subroutine call.

13. Differentiate between ADD WATCH and QUICK WATCH.

"Add to watch" adds the variable to the Watch window, so that you can see its value changing as you step through the code.

"Quick Watch" pops up a transient dialog showing the value, without permanently adding it any here. When you close that dialog, you can no longer see the value.