Course Title: GUI Programming using C#
Course Code: U-GPC-617

What is .net:

What is .NET?

.NET is a free, cross-platform, open source developer platform for building many different types of applications.

With .NET, you can use multiple languages, editors, and libraries to build for web, mobile, desktop, games, IoT, and more.

.NET is a free, open-source, and cross-platform for building modern, scalable, and high-performance desktop, web, cloud, and mobile applications. The current version of .NET is .NET 5.0, which is the successor of .NET Core 3.1 and .NET Framework 4.6. Prior to .NET 5.0, .NET had two versions, .NET Framework, and .NET Core, but in this version, both have merged and now there is only version.

.NET is a single unified platform to build desktop, web, cloud, mobile, gaming, IoT, and AI apps. The .NET ecosystem has a single common library, runtime, language compilers, and tools.

Formally, .NET is "an open source developer platform, created by Microsoft, for building many different types of applications. You can write .NET apps in C#, F#, Visual C++, or Visual Basic."
Informally, .NET is the tool that lets you build and run C# programs (we'll avoid F#, Visual C++, Visual Basic for now).

When you download .NET, you're really downloading a bunch of programs that:

Translate your C# code into instructions that a computer can understand

Provide utilities for building software, like tools for printing text to the screen and finding the current time

Define a set of data types that make it easier for you to store information in your programs, like text, numbers, and dates

There are a few versions of .NET. They do the same job but they are meant for different operating systems:

.NET Framework is the original version of .NET that only runs on Windows computers.

.NET Core is the new, cross-platform version of .NET that runs on Windows, MacOS, and Linux computers. Since it's more flexible and Microsoft is actively enhancing this version, we'll be using this one throughout the path.

Whenever we refer to ".NET" from now on, we really mean .NET Core. Conveniently, Microsoft itself plans to rename .NET Core to .NET towards the end of 2020.

.net Framework:

.NET is a framework to develop software applications. It is designed and developed by Microsoft and the first beta version released in 2000.

It is used to develop applications for web, Windows, phone. Moreover, it provides a broad range of functionalities and support.

This framework contains a large number of class libraries known as Framework Class Library (FCL). The software programs written in .NET are executed in the execution environment, which is called CLR (Common Language Runtime). These are the core and essential parts of the .NET framework.

This framework provides various services like memory management, networking, security, memory management, and type-safety.

The .Net Framework supports more than 60 programming languages such as C#, F#, VB.NET, J#, VC++, JScript.NET, APL, COBOL, Perl, Oberon, ML, Pascal, Eiffel, Smalltalk, Python, Cobra, ADA, etc.

Following is the .NET framework Stack that shows the modules and components of the Framework.

The .NET Framework is composed of four main components:

1. Common Language Runtime (CLR)
2. Framework Class Library (FCL),
3. Core Languages (WinForms, ASP.NET, and ADO.NET), and
4. Other Modules (WCF, WPF, WF, Card Space, LINQ, Entity Framework, Parallel LINQ, Task Parallel Library, etc.)

CLR (Common Language Runtime)

It is a program execution engine that loads and executes the program. It converts the program into native code. It acts as an interface between the framework and operating system. It does exception handling, memory management, and garbage collection. Moreover, it provides security, type-safety, interoperability, and portability.

FCL (Framework Class Library)

It is a standard library that is a collection of thousands of classes and used to build an application. The BCL (Base Class Library) is the core of the FCL and provides basic functionalities.

Visual Studio.net:

Visual Studio .NET is a Microsoft-integrated development environment (IDE) that can be used for developing consoles, graphical user interfaces (GUIs), Windows Forms, Web services and Web applications.

Visual Studio is used to write native code and managed code supported by Microsoft Windows, Windows Mobile, Windows CE, .NET Framework, .NET Compact Framework and Microsoft Silverlight. Visual Studio .NET's code editor supports IntelliSense and code refactoring, while the Visual Studio .NET integrated debugger supports both source and machine-level debugging. Visual Studio .NET includes other built-in tools, like a form designer, which is useful when building GUI applications; a Web designer that creates dynamic Web pages; a class designer that is used to create custom libraries, and a schema designer for database support.

VISUAL STUDIO.NET is an IDE (Integrated development Environment). This is basically designed to make software development easy. Visual Studio.NET comes with .NET framework, which provides an environment for creating console, windows, and web based applications. It supports all .NET compliant languages.

It is very popular IDE for .NET. It provides Intelligence that means developer does not require remembering methods and member of class.

They can get all information like methods, properties etc regarding any class or object by typing dot (.) after the class, object or related thing.

.net Languages:

What are .NET languages?

There are many popular .NET languages with benefits and negatives to each:

Visual Basic .NET (VB.NET). Once Microsoft's flagship programming language product, Visual Basic .NET is a redesigned version of the original Visual Basic that came out back in the 1990s. It's easy to program with and frequently taught in college. Visual Basic has always been a very common and very popular language, which means there are many tools, code snippets, and libraries available for its use. You can go through Github and find code to match practically any project, which also makes it an ideal solution. At the same time, Visual Basic can be difficult to optimize and improve on.

C++. Many programmers already know C++ and can easily transition their knowledge to the .NET environment rather than learning a new language entirely. C++ is one of the most popular languages in the world. Many programmers will learn C++ because it makes it easier for them to get hired; C++ programmers have always been in demand, even though the language is now a few decades old. C++ programmers also have many of the advantages of VB programmers; there's a lot of code and a strong community. At the same time, C++ is a little more difficult to learn.

C#. C# is rapidly becoming Microsoft's most popular language; a mix between Java and C++ (minus many of the problems in Java and C++). C# development is popular for everything from IoT programming to mobile application development. Many find C# easier to learn than the other C and Java languages. C# is also lightweight, scalable, and very easy to optimize and improve upon. Since C# is also newer than many other general-purpose languages, it has better memory allocation and load balancing.

Ada. Ada is a structured, object-oriented high-level programming language, derived from Pascal. Ada was originally designed for the US Department of Defense. The language was named after Augusta Ada King, Countess of Lovelace (1815-1852) who is considered by many to be the first programmer.

F#. F# gives an application simplicity and succinctness like Python but with robustness and performance better than Java or C#. F# is most often used as a cross-platform language on .NET, but it can also be used to for graphics processing and to generate JavaScript. F# is great for specialties like scientific programming or data analysis.

IronPython. For Python programmers, IronPython provides an all-around implementation of Python for .NET. Python is one of the most popular and most-used languages in the world.

IronRuby. Ruby can also be accessed through .NET through IronRuby, ideal for those who are already Ruby on Rails experts.

JScript .NET. A compiled version of the JScript language, JScript.NET is a good option for those who want to produce code quickly and easily.

PHP through Phalanger. PHP is one of the most-known web-based languages; Phalanger makes it possible to do PHP development through .NET.

Perl through Active Perl. With Active Perl, Perl programmers can transition their code into the .NET framework.

Visual COBOL .NET. Visual COBOL can be used to compile directly to Microsoft intermediate language for use in a Microsoft .NET project. Believe it or not, many large insurance companies are still running old COBOL programs written in the 1980s. Visual COBOL can be used to modernize them.

VBx. A variant of Visual Basic .NET, VBx is a dynamic version of Visual Basic that uses a Dynamic Language Runtime.

Those are just the top 12 languages that you can use for .NET; there are others. Of these, C# and VB.NET were those most commonly used for web development. JScript has a relationship to, but is not, JavaScript.

Project types:

The Following are some the different projects that can be created with Visual Studio.NET:

1)Console applications
2)Windows Applications
3)Web applications
4)Web services
5)Class library
6)Windows Control Library
7)Web Control Library

1) Console applications are light weight programs run inside the command prompt (DOS) window. They are commonly used for test applications.

2) Windows Applications are form based standard Windows desktop applications for common day to day tasks. (Ex: Microsoft word).

3) Web applications are programs that used to run inside some web server (Ex:IIS) to fulfill the user requests over the http. (Ex: Hotmail and Google).

4) Web services are web applications that provide services to other applications over the internet.

5) Class library contains components and libraries to be used inside other applications. A Class library cannot be executed and thus it does not have any

entry point.

6) Windows Control Library contains user defined windows controls to be used by Windows applications.

7) Web Control Library contains user defined web controls to be used by web applications.

Project types

In Visual Studio, projects are the containers that developers use to organize source code files and other resources that appear in **Solution Explorer**. Typically, projects are files, for example, a .csproj file for a Visual C# project, that store references to source code files and resources like bitmap files. Projects let you organize, build, debug, and deploy source code, references to Web services and databases, and other resources. VSPackages can extend the Visual Studio project system in three main ways: *project types*, *project subtypes*, and *custom tools*.

Visual Studio includes several project types for languages such as Visual C#, Visual Basic, and Visual J#, and the IronPython integration sample includes a project type for the IronPython language. Visual Studio also lets you create your own project types.

Visual Studio comes with many project templates to create the necessary code and files to start developing applications. Details on few common project types are listed as

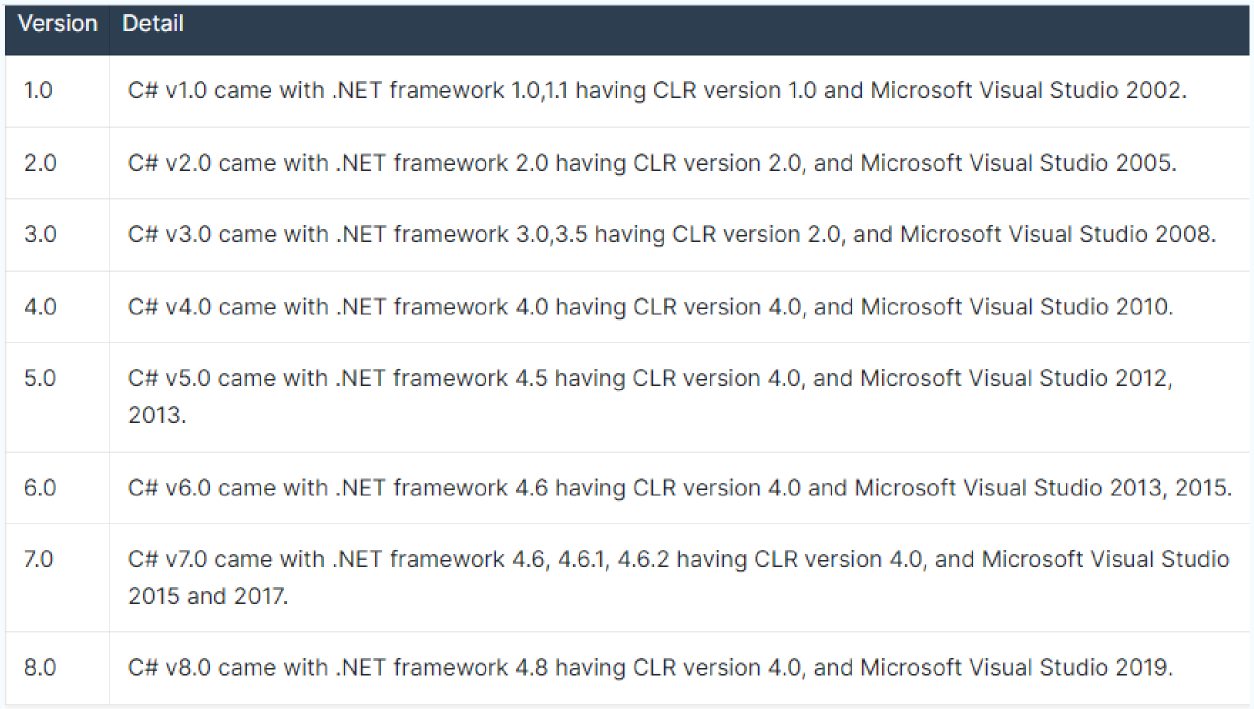
Project Type	**Description**
Class library	Component library with no user interface
Console application	Command line application
Database project	SQL script storage
Device application	Windows application for a smart device
Empty project	Blank project
SQL Server project	Management of stored procedures and SQL Server objects
Web service	ASP.NET Web application with no user interface; technically, no longer a project type
Web site	ASP.NET Web application; technically, no longer a project type
Windows	Windows application with a user interface application
Windows service	Windows application with no user interface

WPF Browser Application	Windows Presentation Foundation browser application

c#.net History:

C# is one of the most popular programming languages developed by Microsoft, along with the dot net framework development, which has been approved by ECMA-334 and ISO. C# was designed to be general-purpose, high-level, fully object-oriented, as well as a component-oriented programming language. **Mr. Anders Hejlsberg** is the creator of this widely used programming language. The language was based on C++ and Java, with additional extensions, libraries, and concepts for implementing different OOPS and component-oriented programming concepts.

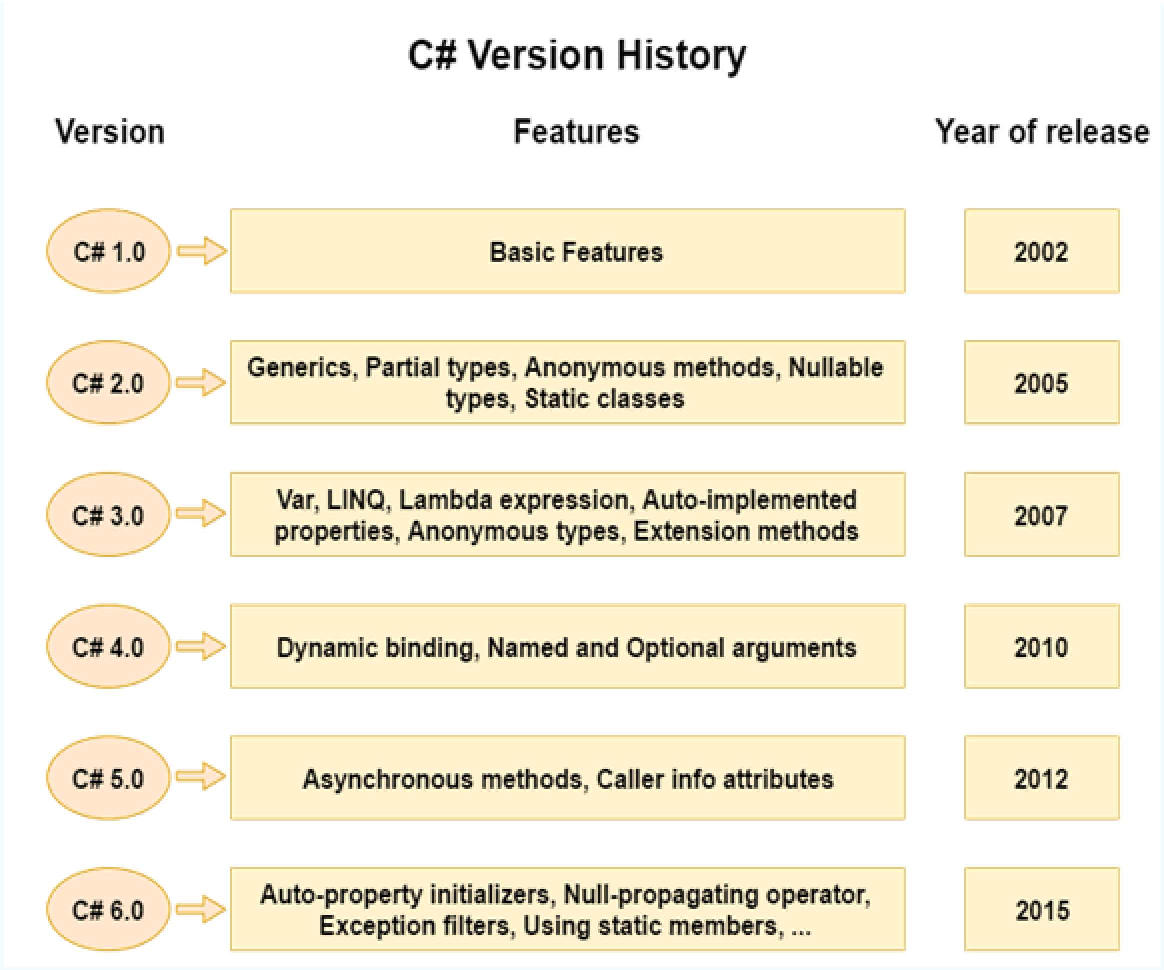
C# was first introduced in 2002 with the DOT NET Framework 1.0 and has since developed significantly. Here is a list of all the major releases of C#:



Version	Detail
1.0	C# v1.0 came with .NET framework 1.0,1.1 having CLR version 1.0 and Microsoft Visual Studio 2002.
2.0	C# v2.0 came with .NET framework 2.0 having CLR version 2.0, and Microsoft Visual Studio 2005.
3.0	C# v3.0 came with .NET framework 3.0,3.5 having CLR version 2.0, and Microsoft Visual Studio 2008.
4.0	C# v4.0 came with .NET framework 4.0 having CLR version 4.0, and Microsoft Visual Studio 2010.
5.0	C# v5.0 came with .NET framework 4.5 having CLR version 4.0, and Microsoft Visual Studio 2012, 2013.
6.0	C# v6.0 came with .NET framework 4.6 having CLR version 4.0 and Microsoft Visual Studio 2013, 2015.
7.0	C# v7.0 came with .NET framework 4.6, 4.6.1, 4.6.2 having CLR version 4.0, and Microsoft Visual Studio 2015 and 2017.
8.0	C# v8.0 came with .NET framework 4.8 having CLR version 4.0, and Microsoft Visual Studio 2019.

C# has evolved much since their first release in the year **2002**. It was introduced with **.NET Framework 1.0** and the current version of C# is 5.0.

Let's see the important features introduced in each version of C# are given below.



C# Version History

Version	Features	Year of release
C# 1.0	Basic Features	2002
C# 2.0	Generics, Partial types, Anonymous methods, Nullable types, Static classes	2005
C# 3.0	Var, LINQ, Lambda expression, Auto-implemented properties, Anonymous types, Extension methods	2007
C# 4.0	Dynamic binding, Named and Optional arguments	2010
C# 5.0	Asynchronous methods, Caller info attributes	2012
C# 6.0	Auto-property initializers, Null-propagating operator, Exception filters, Using static members, ...	2015

Design Goals:

The design goals of the language were software robustness, durability and programmer productivity. It can be used to create console applications, GUI applications, web applications, both on PCs and embedded systems. C# was created by Microsoft corporation.15-Dec-2021

Paradigms: Object-oriented programming

Inventor: Microsoft Corporation

C# is a modern, high-level, general-purpose, object-oriented programming language. It is the principal language of the .NET framework. It supports functional, procedural, generic, object-oriented, and component-oriented programming disciplines.

The design goals of the language were software robustness, durability and programmer productivity. It can be used to create console applications, GUI applications, web applications, both on PCs and embedded systems. C# was created by Microsoft corporation. The name "C sharp" was inspired by musical notation where a sharp indicates that the written note should be made a semitone higher in pitch.

Inherent within the Microsoft .NET Framework are many design goals that are practical yet extremely ambitious. In this section, we discuss the main design goals of the Microsoft .NET Framework, including better support for components, language integration, application interoperation across cyberspace, simple development and deployment, better reliability, and greater security.

Prior to the existence of COM technology, Microsoft developers had no simple way to integrate binary libraries without referring to or altering their source code. With the advent of COM, programmers were able to integrate binary components into their applications, similar to the way we plug-and-play hardware components into our desktop PCs. Although COM was great, the grungy details of COM gave developers and administrators many headaches.

While COM permits you to integrate binary components developed using any language, it does require you to obey the COM identity, lifetime, and binary layout rules. You must also write the plumbing code that is required to create a COM component, such as DllGetClassObject, CoRegisterClassObject, and others.

Realizing that all of these requirements result in frequent rewrites of similar code, .NET sets out to remove all of them. In the .NET world, all classes are ready to be reused at the binary level. You don’t have to write extra plumbing code to support componentization in the .NET Framework. You simply write a .NET class, which then becomes a part of an assembly (to be discussed in Chapter 2), and it will support plug-and-play.

How c# differs from c++:

C-Sharp is an object-oriented programming language developed by Microsoft that runs on .Net Framework. It has features like strong typing, imperative, declarative, object-oriented (class-based), and component-oriented programming. It was developed by Microsoft within the .NET platform.

The name “C sharp” was inspired by musical notations. Here ‘#’ symbol indicates that the written note must be made a semitone higher in pitch.

C# is a general-purpose, modern and object-oriented programming language pronounced as “C sharp”. It was developed by Microsoft led by Anders Hejlsberg and his team.
C++ is a statically typed, multiparadigm, and object-oriented programming language. In beginning, C++ was termed as C with classes. It was developed by Bjarne Stroustrup at AT & T Bell Laboratories.

Below are some major differences between C++ and C#:

Feature	C++	C#
Memory Management	In C++ memory management is performed manually by the programmer. If a programmer creates an object then he is responsible to destroy that object after the completion of that object’s task.	In C# memory management is performed automatically by the garbage collector. If the programmer creates an object and after the completion of that object’s task the garbage collector will automatically delete that object.
Platform Dependency	C++ code can be run on any platform. C++ is used where the application needed to directly communicate with hardware.	C# code is windows specific. Although Microsoft is working to make it global but till now the major system does not provide support for C#.
Multiple Inheritance	C++ support multiple inheritance through classes. Means that a class can extend more than one class at a time.	C# does not support any multiple inheritances through classes.
Bound Checking	In C++ bound checking is not performed by compiler. By mistake, if the programmer tries to access invalid array index then it will give the wrong result but will not show any compilation error.	In C# bound checking in array is performed by compiler. By mistake, if the programmer tries to access an invalid array index then it will give compilation error.
Pointers	In C++ pointers can be used	In C# pointers can be used only

Feature	C++	C#
	anywhere in the program.	in unsafe mode.
Language Type	C++ is a low level language.	C# is high level object oriented language.
Level of Difficulty	C++ includes very complex features.	C# is quite easy because it has the well-defined hierarchy of classes.
Application Types	C++ is typically used for console applications.	C# is used to develop mobile, windows, and console applications.
Compilation	C++ code gets converted into machine code directly after compilation.	C# code gets converted into intermediate language code after compilation.
Object Oriented	C++ is not a pure object-oriented programming language due to the primitive data types.	C# is a pure object-oriented programming language.
Access Specifiers	The access modifiers are public, private, protected. It does not contain internal & protected internal access modifiers.	In C# public, private, protected, internal & protected internal are used for access specifiers.
Test Variable	In switch statement, the test variable can not be a string.	In switch statement, the test variable can be a string.
Control statement	It does not contain such extra flow control statement.	In addition to for, while and do while; it has another flow control statement called for each.
Function Pointers	It does have the concept of function pointers.	It does not have the concept of function pointers.

Feature	C++	C#
Binaries	In C++ , size of binaries is low and lightweight.	In C# , size of binaries is high because of overhead libraries.
Garbage Collection	C++ do not support garbage collection.	Garbage collection is supported by C#
Types of Projects	It is mainly used for such projects that focus on accessing the hardware and better performance.	It is mainly used in modern application development.

Characteristics of c#.net:

C# Features

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

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

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

Explain the characteristics of C#

The characteristics of C# are the following:

1. **Simple**
2. **Modern**
3. **Object-oriented**
4. **Consistent**
5. **Versionable**
6. **Flexible**
7. **Interoperable**
8. **Compatible**
9. **Typesafe**
10. **Support Garbage collection**

11. **Exception Handling**
12. **Security**

1. simple:-

C# is designed to be easy for the professional programmer to learn and use effectively. If one has some programming experience, he will not find C# hard to master. Also, some of the confusing concepts from C++ are either left out of C# or implemented in a clearer manner. For example, the complex pointers in C++ are missing in C#. in C++ one has ., --> operators for namespaces, member access, and references respectively. But in C# a single dot (.) operator does all these operations.

2. Modern:-

The C# is emerged as a language for waiting for NGWS (next-generation windows services) applications. In C# memory management is automatic and is no longer the responsibility of the programmers. This automatic memory management is a responsibility of C# garbage collector. Another modern feature of C# is that it supports cross-language exception handling. A new data type called decimal is added in C# for monetary calculations. Another feature of C# for monetary calculations. Another modern feature of C# is that it supports a robust model.

3. Object-oriented:-

C# is an object-oriented programming language, thereby providing re-usability of existing code and reducing code redundancy. It supports major object-oriented programming features such as data encapsulation, inheritance, and polymorphism.

4. Consistent:-

C# supports a unified type system which eliminates the problem of varying ranges of integer types. All types are treated as objects and developers can extend the type system simply and easily.

5. Versionable:-

Versioning is the process of evolving an application in a compatible manner. C# supports both source and binary compatibility in applications. The source compatibility means that a new version of an application can work with the previous version of the application on recompiling the previous application. Binary compatibility means that the previous version of an application can work with the new version of the application without recompiling the previous version. Thus, C# supports versioning that enables the existing applications to run on different versions.

6. Flexible:-

In C#, code can be written either in safe mode or in unsafe mode. The default for C# code is the safe mode which does not support pointers. But one can declare which enables the use of pointers. This shows the flexibility of C#.

7. Interoperable:-

C# provides support for cross-language interoperability. This code produced by C# can work easily with the code produced by other languages. This is made possible through the CLS, Which defines a set of rules that every language in the .NET framework has to follow. The C# programs can use existing COM objects, irrespective of the language they are written in.

8. Compatible:-

C# enforces the .NET common language specification and therefore allows interoperation with another .NET language.

9. Typesafe:-

The common type system provides type safety, which in turn improves code stability. C# provides the following type safety measures:

10. Support Garbage collection:-

Garbage collection is the process of recovering computer memory that is no longer required by the program. It is the feature of .NET that C# supports. Thus garbage collection automatically reclaims memory occupied by unused objects.

11. Exception Handling:-

The .NET standardizes the exception handling across languages. Exception handling provides a structured and extensible approach to error detection and recovery. The C# language's exception handling features provide a way to deal with any unexpected or exceptional situations that arise while a program is running.

12. Security:-

C# security is designed to operate as part of the .NET runtime and provides several built-in security features.

C# is object oriented programming language. It provides a lot of **features** that are given below.

1. Simple
2. Modern programming language

3. Object oriented
4. Type safe
5. Interoperability
6. Scalable and Updateable
7. Component oriented
8. Structured programming language
9. Rich Library
10. Fast speed

c#.net and Console Vs. GUI Application:

A console application, in the context of C#, is an application that takes input and displays output at a command line console with access to three basic data streams: standard input, standard output and standard error.

A console application facilitates the reading and writing of characters from a console - either individually or as an entire line. It is the simplest form of a C# program and is typically invoked from the Windows command prompt. A console application usually exists in the form of a stand-alone executable file with minimal or no graphical user interface (GUI).

The program structure of a console application facilitates a sequential execution flow between statements. Designed for the keyboard and display screen, a console application is driven by keyboard and system events generated by network connections and objects.

A console application is an application that runs in a console window same as a C and C++ program.

It doesn’t have any graphical user interface. Console Applications will have character based interface.

A console application is primarily designed for the following reasons:

- To provide a simple user interface for applications requiring little or no user interaction, such as samples for learning C# language features and command-line utility programs.
- Automated testing, which can reduce automation implementation resources.

Console applications developed in C# have one main entry point (static main method) of execution, which takes an optional array of parameters as its only argument for command-line parameter representation.

The .NET Framework provides library classes to enable rapid console application development with output display capability in different formats. System.Console (a sealed class) is one of the main classes used in the development of console applications.

One console application functionality limitation is that strings returned by console functions using original equipment manufacturer (OEM) code page may not be correctly processed by functions using American National Standards Institute (ANSI) code page. This issue may be resolved by calling the SetFileApisToOEM function to produce OEM character strings, rather than ANSI character strings.

C# Graphical User Interface.

C# has all the features of any powerful, modern language. In C#, the most rapid and convenient way to create your user interface is to do so visually, using the **Windows Forms** Designer and **Toolbox**. Windows Forms controls are reusable components that encapsulate user interface functionality and are used in client side Windows based applications.

A control is a component on a form used to display information or accept user input. The Control class provides the base functionality for all controls that are displayed on a Form.

Windows Forms is a Graphical User Interface(GUI) class library which is bundled in *.Net Framework*. Its main purpose is to provide an easier interface to develop the applications for desktop, tablet, PCs. It is also termed as the **WinForms**. The applications which are developed by using Windows Forms or WinForms are known as the **Windows Forms Applications** that runs on the desktop computer. WinForms can be used only to develop the Windows Forms Applications not web applications. WinForms applications can contain the different type of controls like labels, list boxes, tooltip etc.

Creating a Windows Forms Application Using Visual Studio 2017

- First, open the Visual Studio then Go to **File -> New -> Project** to create a new project and then select the language as *Visual C#* from the left menu. Click on *Windows Forms App(.NET Framework)* in the middle of current window. After that give the project name and Click **OK**.

Chapter 2: Console Application Basic

I/O Statement:

What is C#?

C# is pronounced "C-Sharp".

It is an object-oriented programming language created by Microsoft that runs on the .NET Framework.

C# has roots from the C family, and the language is close to other popular languages like C++ and Java.

The first version was released in year 2002. The latest version, **C# 8**, was released in September 2019.

C# is used for:

- Mobile applications
- Desktop applications
- Web applications
- Web services
- Web sites
- Games
- VR
- Database applications
- And much, much more!

Why Use C#?

- It is one of the most popular programming language in the world
- It is easy to learn and simple to use
- It has a huge community support

- C# is an object oriented language which gives a clear structure to programs and allows code to be reused, lowering development costs
- As C# is close to C, C++ and Java, it makes it easy for programmers to switch to C# or vice versa.

C# IDE

- The easiest way to get started with C#, is to use an IDE.

- An IDE (Integrated Development Environment) is used to edit and compile code.

Chapter 2: Console Application Basic

I/O Statement:

The Console class allows using the Write () and the WriteLine () functions to display things on the screen. While the Console.Write () method is used to display something on the screen, the Console class provides the Read () method to get a value from the user. To use it, the name of a variable can be assigned to it. The syntax used is:

variableName = Console. Read();

This simply means that, when the user types something and presses Enter, what the user had typed would be given (the word is assigned) to the variable specified on the left side of the assignment operator.

Read() doesn't always have to assign its value to a variable. For example, it can be used on its own line, which simply means that the user is expected to type something but the value typed by the user would not be used for any significant purpose. For example some versions of C# (even including Microsoft's C# and Borland C#Builder) would display the DOS window briefly and disappear. You can use the Read() function to wait for the user to press any key in order to close the DOS window.

Besides Read(), the Console class also provides the ReadLine() method. Like the WriteLine() member function, after performing its assignment, the ReadLine() method sends the caret to the next line. Otherwise, it plays the same role as the Read() function.

string FirstName;

Console.write("Enter First Name: ");

FirstName = console.ReadLine();

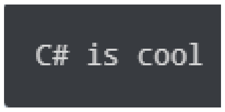
In C#, everything the user types is a string and the compiler would hardly analyze it without your explicit asking it to do so. Therefore, if you want to get a number from the user, first request a string. after getting the string, you must convert it to a number. To perform this conversion, each data type of the .NET Framework provides a mechanism called Parse. To use Parse(), type the data type, followed by a period, followed by Parse, and followed by parentheses. In the parentheses of Parse, type the string that you requested from the user.

Here is an example:

Example 1: Printing String using WriteLine()

```
 using System;

using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace aks1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("C# is cool");
            Console.ReadLine();
        }
    }
}
```

Output:



```
C# is cool
```

C# Output

In order to output something in C#, we can use

```
System.Console.WriteLine() OR

System.Console.Write()
```

Here, `System` is a namespace, `Console` is a class within namespace `System` and `WriteLine` and `Write` are methods of class `Console`.
Let's look at a simple example that prints a string to output screen.

Difference between WriteLine() and Write() method

The main difference between `WriteLine()` and `Write()` is that the `Write()` method only prints the string provided to it, while the `WriteLine()` method prints the string and moves to the start of next line as well.
Let's take at a look at the example below to understand the difference between these methods.

Example 2: How to use WriteLine() and Write() method?

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace aks1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Prints on ");
            Console.WriteLine("New line");

            Console.Write("Prints on ");
            Console.Write("Same line");
        }
    }
}
```

When we run the program, the output will be

```
Prints on
New line
Prints on Same line
```

Example 3: Printing Variables and Literals

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace aks1
{
    class Program
    {
        static void Main(string[] args)
        {
            int value = 10;

            // Variable
            Console.WriteLine(value);
            // Literal
            Console.WriteLine(50.05);
        }
    }
}
```

When we run the program, the output will be

```
10
50.05
```

Example 4: Printing Concatenated String using + operator

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace aks1
{
    class Program
    {
        static void Main(string[] args)
        {
            int val = 55;
            Console.WriteLine("Hello " + "World");
            Console.WriteLine("Value = " + val);
        }
    }
}
```

When we run the program, the output will be

```
Hello World
```

```
Value = 55
```

C# Input

In C#, the simplest method to get input from the user is by using the `ReadLine()` method of the `Console` class. However, `Read()` and `ReadKey()` are also available for getting input from the user. They are also included in `Console` class.

Example 6: Get String Input From User

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace aks1
{
    class Program
    {
        static void Main(string[] args)
        {
            string testString;
            Console.Write("Enter a string - ");
            testString = Console.ReadLine();
            Console.WriteLine("You entered '{0}'", testString);
        }
    }
}
```

When we run the program, the output will be:

```
Enter a string - Hello World
You entered 'Hello World'
```

Difference between ReadLine(), Read() and ReadKey() method:

The difference between `ReadLine()`, `Read()` and `ReadKey()` method is:

- `ReadLine()`: The `ReadLine()` method reads the next line of input from the standard input stream. It returns the same string.
- `Read()`: The `Read()` method reads the next character from the standard input stream. It returns the ascii value of the character.

- `ReadKey()`: The `ReadKey()` method obtains the next key pressed by user. This method is usually used to hold the screen until user press a key.

If you want to know more about these methods, here is an interesting discussion on StackOverflow on: Difference between Console.Read() and Console.ReadLine()?.

Example 7: Difference between Read() and ReadKey() method

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace aks1
{
    class Program
    {
        static void Main(string[] args)
        {
            int userInput;

            Console.WriteLine("Press any key to continue...");
            Console.ReadKey();
            Console.WriteLine();

            Console.Write("Input using Read() - ");
            userInput = Console.Read();
            Console.WriteLine("Ascii Value = {0}", userInput);
        }
    }
}
```

When we run the program, the output will be

```
Press any key to continue...
x
Input using Read() - Learning C#
Ascii Value = 76
```

From this example, it must be clear how `ReadKey()` and `Read()` method works. While using `ReadKey()`, as soon as the key is pressed, it is displayed on the screen.

When `Read()` is used, it takes a whole line but only returns the ASCII value of first character. Hence, `76` (ASCII value of `L`) is printed.

Reading numeric values (integer and floating point types)

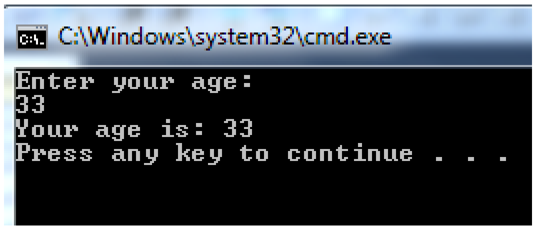
Reading a character or string is very simple in C#. All you need to do is call the corresponding methods as required.

But, reading numeric values can be slightly tricky in C#. We'll still use the same `ReadLine()` method we used for getting string values. But since the `ReadLine()` method receives the input as string, it needs to be converted into integer or floating point type.
One simple approach for converting our input is using the methods of `Convert` class.

Example 8: Reading Numeric Values from User using Convert class

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace aks2
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Enter your age:");
            int age = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("Your age is: " + age);
        }
    }
}
```

Output:



```
Enter your age:
33
Your age is: 33
Press any key to continue . . .
```

Console Output

Instead of using two Write() or a combination of Write() and WriteLine() to display data, you can convert a value to a string and display it directly. To do this, you can provide two strings to the Write() or WriteLine() and separate them with a comma:

1. The first part of the string provided to Write() or WriteLine() is the complete string that would display to the user. This first string itself can be made of different sections:

foreach loop:

The **foreach loop** in C# executes a block of code on each element in an array or a collection of items. When executing foreach loop it traversing items in a collection or an array . The foreach loop is useful for traversing each items in an array or a collection of items and displayed one by one.

In C#, the foreach loop iterates collection types such as Array, ArrayList, List, Hashtable, Dictionary, etc. It can be used with any type that implements the IEnumerable interface.

The foreach, in loop in C# is used to loop through the items in a collection.

The foreach loop is used to iterate over the elements of the collection. The collection may be an array or a list. It executes for each element present in the array. It is necessary to enclose the statements of foreach loop in curly braces {}

Looping in a programming language is a way to execute a statement or a set of statements multiple numbers of times depending on the result of a condition to be evaluated. The resulting condition should be true to execute statements within loops. The **foreach loop** is used to iterate over the elements of the collection. The collection may be an array or a list. It executes for each element present in the array.

1. It is necessary to enclose the statements of foreach loop in curly braces {}.
2. Instead of declaring and initializing a loop counter variable, you declare a variable that is the same type as the base type of the array, followed by a colon, which is then followed by the array name.
3. In the loop body, you can use the loop variable you created rather than using an indexed array element.

Difference between for loop and foreach loop:

- for loop executes a statement or a block of statement until the given condition is false. Whereas *foreach* loop executes a statement or a block of statements for each element present in the array and there is no need to define the minimum or maximum limit.
- In *for loop*, we iterate the array in both forward and backward directions, e.g from index 0 to 9 and from index 9 to 0. But in the foreach loop, we iterate an array only in the forward direction, not in a backward direction.
- In terms of a variable declaration, foreach loop has five variable declarations whereas for loop only have three variable declarations.
- The foreach loop copies the array and put this copy into the new array for operation. Whereas for loop doesn't do.

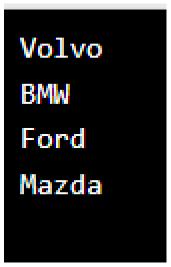
There is also a foreach loop, which is used exclusively to loop through elements in an **array**:

Syntax

```
foreach (type variableName in arrayName)

{

  // code block to be executed

}
```

```
using System;

namespace MyApplication
{
  class Program
  {
    static void Main(string[] args)
    {
      string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
      foreach (string i in cars)
      {
        Console.WriteLine(i);
      }
    }
  }
}
```

Output:



```
Volvo
BMW
Ford
Mazda
```

Array:

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

To declare an array, define the variable type with **square brackets**:

Exp. string[] cars;

We have now declared a variable that holds an array of strings.

To insert values to it, we can use an array literal - place the values in a comma-separated list, inside curly braces:

string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

To create an array of integers, you could write:

int[] myNum = {10, 20, 30, 40};

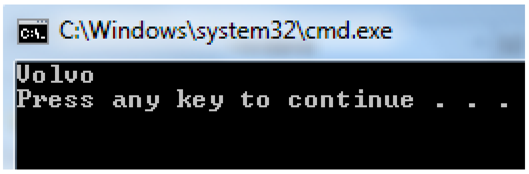
Access the Elements of an Array

You access an array element by referring to the index number.

This statement accesses the value of the first element in **cars**:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace aks2
{
    class Program
    {
        static void Main(string[] args)
        {
            string[] cars = { "Volvo", "BMW", "Ford", "Mazda" };
            Console.WriteLine(cars[0]);
        }
    }
}
```

Output:



```
Volvo
Press any key to continue . . .
```

ArrayList:

C# - ArrayList

In C#, the `ArrayList` is a non-generic collection of objects whose size increases dynamically. It is the same as Array except that its size increases dynamically.

An `ArrayList` can be used to add unknown data where you don't know the types and the size of the data.

Create an ArrayList

The `ArrayList` class included in the `System.Collections` namespace. Create an object of the `ArrayList` using the `new` keyword.

Example: Create an ArrayList

```
using System.Collections;

ArrayList arlist = new ArrayList();
// or
var arlist = new ArrayList(); // recommended
```

Adding Elements in ArrayList

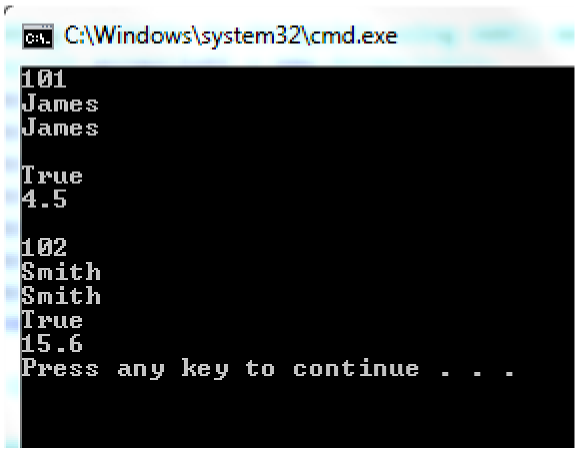
Use the `Add()` method or object initializer syntax to add elements in an `ArrayList`.

An `ArrayList` can contain multiple `null` and duplicate values.

Example: Adding Elements in ArrayList

```
// adding elements using ArrayList.Add() method
var arlist1 = new ArrayList();
arlist1.Add(1);
arlist1.Add("Bill");
arlist1.Add(" ");
arlist1.Add(true);
arlist1.Add(4.5);
arlist1.Add(null);

// adding elements using object initializer syntax
var arlist2 = new ArrayList()
                {
                    2, "Steve", " ", true, 4.5, null
                };
```

OUTPUT:


C:\Windows\system32\cmd.exe

```
101
James
James

True
4.5

102
Smith
Smith
True
15.6
Press any key to continue . . .
```

Jagged Array:

The elements of a jagged array in C# are arrays, and hence it is also called “array of arrays”. In a jagged array, the size of the elements can be different.

Jagged array is a **array of arrays** such that member arrays can be of different sizes. In other words, the length of each array index can differ. The elements of Jagged Array are reference types and initialized to null by default. Jagged Array can also be mixed with multidimensional arrays. Here, the number of rows will be fixed at the declaration time, but you can vary the number of columns.

n C#, jagged array is also known as "array of arrays" because its elements are arrays. The element size of jagged array can be different.

The elements of a jagged array in C# are arrays, and hence it is also called “array of arrays”. In a jagged array, the size of the elements can be different.

Declaration of Jagged array

Let's see an example to declare jagged array that has two elements.

```
int[][] arr = new int[2][];
```

Initialization of Jagged array

Let's see an example to initialize jagged array. The size of elements can be different.

```
arr[0] = new int[4];
arr[1] = new int[6];
```

Initialization and filling elements in Jagged array

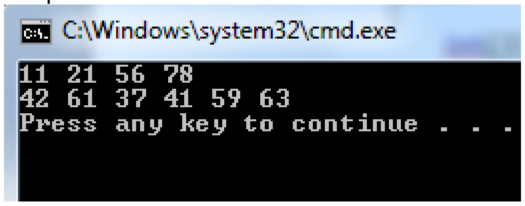
```
arr[0] = new int[4] { 11, 21, 56, 78 };
arr[1] = new int[6] { 42, 61, 37, 41, 59, 63 };
```

Here, size of elements in jagged array is optional. So, you can write above code as given below:

```
arr[0] = new int[] { 11, 21, 56, 78 };
arr[1] = new int[] { 42, 61, 37, 41, 59, 63 };
```

```
public class JaggedArrayTest
{
    public static void Main()
    {
        int[][] arr = new int[2][];// Declare the array

        arr[0] = new int[] { 11, 21, 56, 78 };// Initialize the array
        arr[1] = new int[] { 42, 61, 37, 41, 59, 63 };

        // Traverse array elements
        for (int i = 0; i < arr.Length; i++)
        {
            for (int j = 0; j < arr[i].Length; j++)
            {
                System.Console.Write(arr[i][j] + " ");
            }
            System.Console.WriteLine();
        }
    }
}
```

Output:



C:\Windows\system32\cmd.exe

```
11 21 56 78
42 61 37 41 59 63
Press any key to continue . . .
```

Hash Table:

A Hashtable is a collection of key/value pairs that are arranged based on the hash code of the key. Or in other words, a Hashtable is used to create a collection which uses a hash table for storage. It generally optimized the lookup by calculating the hash code of every key and store into another basket automatically and when you accessing the value from the hashtable at that time it matches the hashcode with the specified key. It is the non-generic type of collection which is defined in System.Collections namespace.

How to create a Hashtable?

Hashtable class provides **16** different types of constructors which are used to create a hashtable, here we only use *Hashtable() constructor*. This constructor is used to create an instance of the Hashtable class which is empty and having the default initial capacity, load factor, hash code provider, and comparer. Now, let’s see how to create a hashtable using Hashtable() constructor:

Step 1: Include System.Collections namespace in your program with the help of *using* keyword:

```
using System.Collections;
```

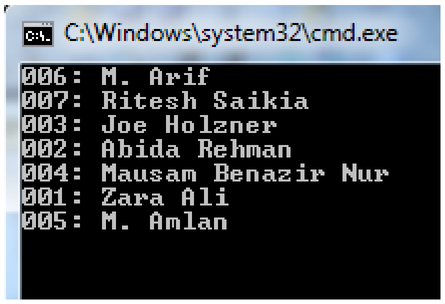
Step 2: Create a hashtable using Hashtable class as shown below:

```
Hashtable hashtable_name = new Hashtable();
```

Step 3: If you want to add a key/value pair in your hashtable, then use Add() method to add elements in your hashtable. And you can also store a key/value pair in your hashtable without using `Add() method`.

```
using System;
using System.Collections;

namespace CollectionsApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            Hashtable ht = new Hashtable();

            ht.Add("001", "Zara Ali");
            ht.Add("002", "Abida Rehman");
            ht.Add("003", "Joe Holzner");
            ht.Add("004", "Mausam Benazir Nur");
            ht.Add("005", "M. Amlan");
            ht.Add("006", "M. Arif");
            ht.Add("007", "Ritesh Saikia");



            // Get a collection of the keys.
            ICollection key = ht.Keys;

            foreach (string k in key)
            {
                Console.WriteLine(k + ": " + ht[k]);
            }
            Console.ReadKey();
        }
    }
}
```

Output:



```
006: M. Arif
007: Ritesh Saikia
003: Joe Holzner
002: Abida Rehman
004: Mausam Benazir Nur
001: Zara Ali
005: M. Amlan
```

Properties, Events & Delegates

Properties & its type:

Properties

You learned from the previous chapter that `private` variables can only be accessed within the same class (an outside class has no access to it). However, sometimes we need to access them - and it can be done with properties.

A property is like a combination of a variable and a method, and it has two methods: a `get` and a `set` method:

In c#, **Property** is an extension of the class variable. It provides a mechanism to read, write, or change the class variable's value without affecting the external way of accessing it in our applications.

In c#, properties can contain one or two code blocks called **accessors,** and those are called a get accessor and set accessor. By using get and set accessors, we can change the internal implementation of class variables and expose it without affecting the external way of accessing it based on our requirements.Generally, in object-oriented programming languages like c# you need to define fields as **private** and then use properties to access their values in a **public** way with get and set accessors.

Following is the syntax of defining a **property** with get and set accessor in c# programming language.

Properties

In C#, properties are a member that provides a flexible mechanism to read, write, or compute the value of a private field.

C# is one of the first languages that offers direct support of Properties. Properties looks similar to a public variable on which we can do get() or set() operations.

Syntax

The following is the syntax of Properties:

```
//Define the Properties Accessors
Public < type > < Property Name >
{
    Get
    {
            Return
            <var > ;
        }
        Set
        {
            If(Is Valid(Value))
            <var > = Value;
        }
}
```

In the syntax shown above:

1. The access modifier can be Private, Public, Protected or Internal.

2. The return type can be any valid C# data type, such as a string or integer.

3. The ”this” keyword shows that the object is for the current class.

4. The get() and set() operations of the syntax are known as accessors.

There are the following 4 types of Properties:

1. Read-Write Property

2. Read-Only Property

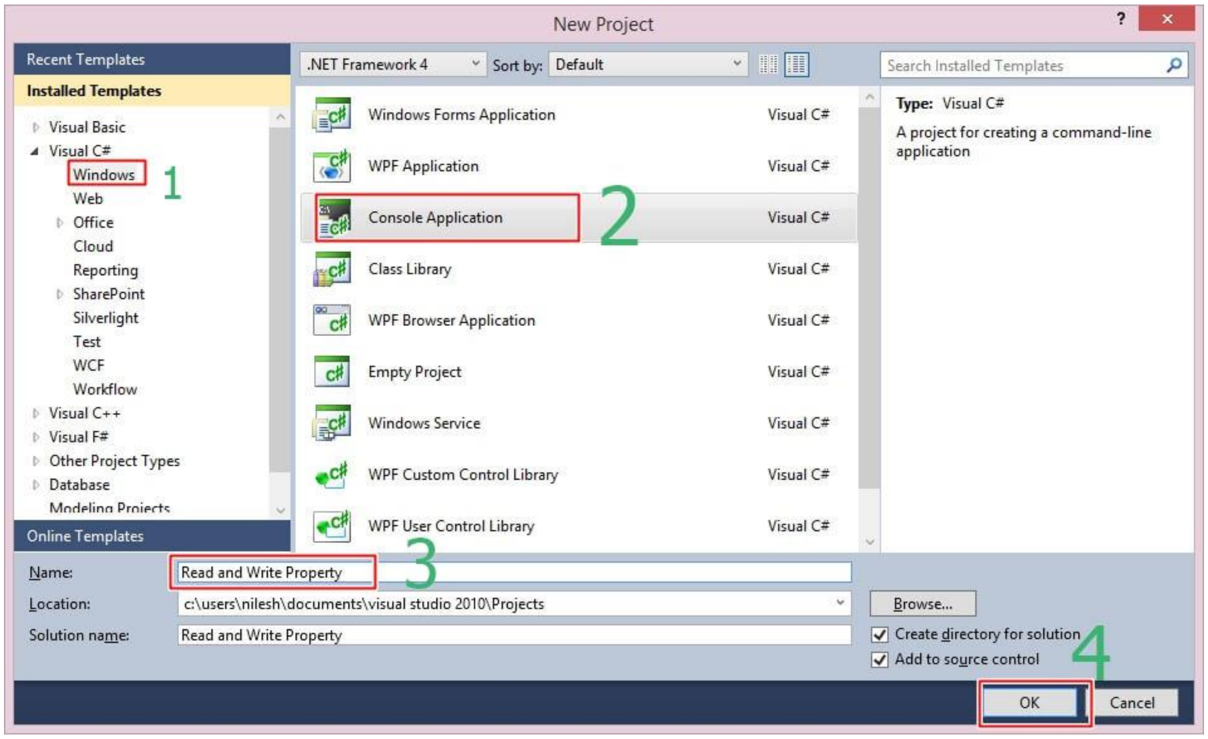
3. Static Property

4. Indexer Property

Read and write Property

Programmers allow you to access the internal data of a class in a less cumbersome manner. Earlier programmers were required to define two separate methods, one for assigning a value of a variable and the other for retrieving the value of a variable. When you create a property, the compiler automatically generates class methods to set() and get() the property value and makes calls to these methods automatically when a programmer uses the property .

Here is a simple program of a Read and Write Property.



Type: Visual C#

A project for creating a command-line application

Name: Read and Write Property

Location: c:\users\nilesh\documents\visual studio 2010\Projects

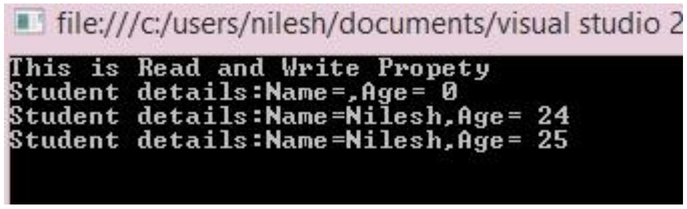
Solution name: Read and Write Property

Figure 1: Read and write property

Here is the code for for a read-write property.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Read_and_Write_Property
{
    class student
    {
        public string Myname = "";
        public int Myage = 0;

        //Declare a Name Property of type String
        public string Name
        {
            get
            {
                return Myname;
            }
            set
            {
                Myname = value;
            }
        }

        //Declare an Age Property of type int

        public int Age
        {
            get
            {
                return Myage;
            }
            set
            {
                Myage = value;
            }
        }

        public override string ToString() {
            return ("Name=" + Name + ",Age= " + Age);
        }

    }

    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("This is Read and Write Propety");

            // Create a new object for student class
            student s = new student();
            Console.WriteLine("Student details:" + s);
            s.Name = "Nilesh";
            s.Age = 24;
            Console.WriteLine("Student details:" + s);
            //increment the age property
            s.Age += 1;
            Console.Write("Student details:" + s);
            Console.ReadKey();

        }
    }
}
```

Read-Write Property Output


file:///c:/users/nilesh/documents/visual studio 2

```
This is Read and Write Propety
Student details:Name=,Age= 0
Student details:Name=Nilesh,Age= 24
Student details:Name=Nilesh,Age= 25
```

Figure 2: Output for read and write property

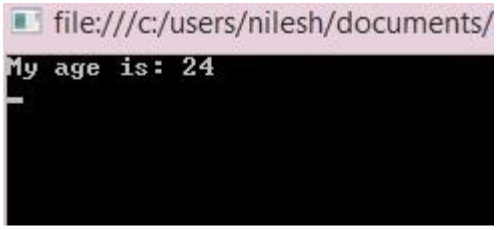
Read-Only Property

A read-only Property has a get accessor but does not have any set() operation. This means that you can retrieve the value of a variable using the read-only property but you cannot assign a value to the variable.

Here is the Code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
    public class PropertyHolder
    {
        private int Myage = 0;
        public PropertyHolder(int PropVal)
        {
            Myage = PropVal;
        }
        public int age
        {
            get {
                return Myage;
            }
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            PropertyHolder p = new PropertyHolder(24);
            Console.WriteLine("My age is: " + p.age);
            Console.ReadKey();
        }
    }
}
```

Read-Only Output



file:///c:/users/nilesh/documents/

```
My age is: 24
```

Figure 3: Read only property output

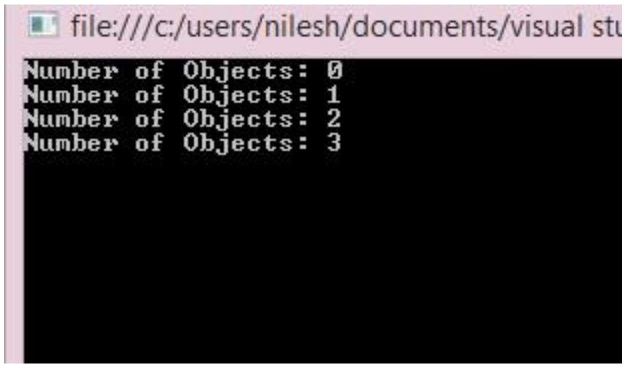
Static Property

A static Property can be used to access only the static members of the class.

Here is the code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Static_Property
{
    public class CounterClass
    {
        private static int number = 0;
        public CounterClass()
        {
            number++;
        }
        public static int NumberofObjects
        {
            get {
                return number;
            }

            set {
                number = value;
            }
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Number of Objects: {0}", CounterCl
ass.NumberofObjects);

            CounterClass object1 = new CounterClass();
            Console.WriteLine("Number of Objects: {0}", CounterCl
ass.NumberofObjects);

            CounterClass object2 = new CounterClass();
            Console.WriteLine("Number of Objects: {0}", CounterCl
ass.NumberofObjects);

            CounterClass object3 = new CounterClass();
            Console.WriteLine("Number of Objects: {0}", CounterCl
ass.NumberofObjects);
            Console.ReadKey();


        }
    }
}
```

Static Property Output



file:///c:/users/nilesh/documents/visual st

```
Number of Objects: 0
Number of Objects: 1
Number of Objects: 2
Number of Objects: 3
```

Figure 4: Static property output

Example

class Person

{

```
  private string name; // field


  public string Name   // property

  {

    get { return name; }   // get method

    set { name = value; }  // set method

  }

}
```

Example explained

The Name property is associated with the name field. It is a good practice to use the same name for both the property and the private field, but with an uppercase first letter.

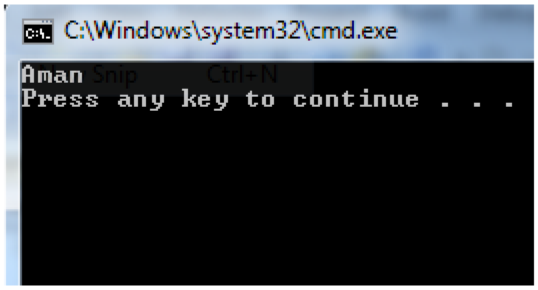
The get method returns the value of the variable name.

The set method assigns a value to the name variable. The value keyword represents the value we assign to the property.

Now we can use the Name property to access and update the private field of the Person class:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Aman
{
    class Person
    {
        private string name; // field
        public string Name   // property
        {
            get { return name; }
            set { name = value; }
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Person myObj = new Person();
            myObj.Name = "Aman";
            Console.WriteLine(myObj.Name);
        }
    }
}
```

Output:



```
Aman
Press any key to continue . . .
```

Events:

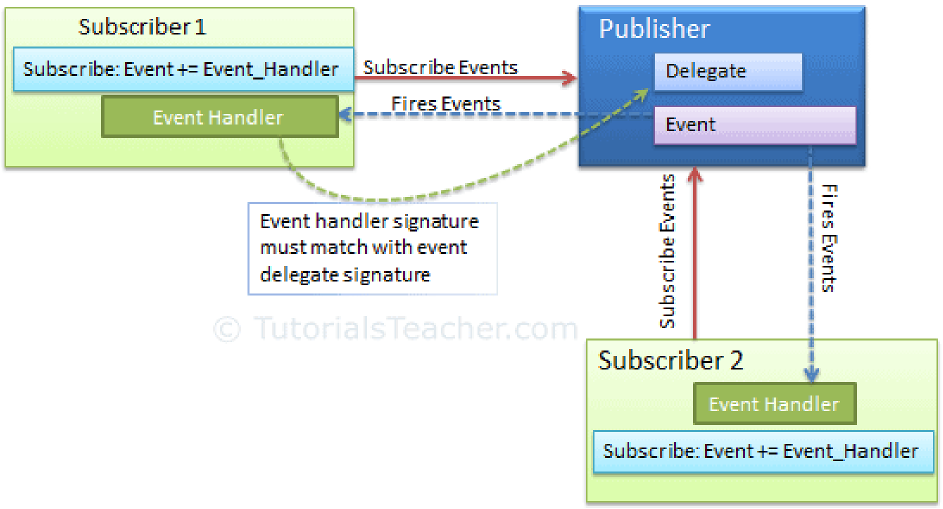
Events are user actions such as key press, clicks, mouse movements, etc., or some occurrence such as system generated notifications. Applications need to respond to events when they occur. For example, interrupts. Events are used for inter-process communication.

An event is a notification sent by an object to signal the occurrence of an action.

The class who raises events is called Publisher, and the class who receives the notification is called Subscriber. There can be multiple subscribers of a single event. Typically, a publisher raises an event when some action occurred. The subscribers, who are interested in getting a notification when an action occurred, should register with an event and handle it.

In C#, an event is an encapsulated delegate. It is dependent on the delegate. The delegate defines the signature for the event handler method of the subscriber class.

The following figure illustrates the event in C#.



Subscriber 1
Subscribe: Event += Event_Handler
Event Handler
Subscribe Events
Fires Events
Publisher
Delegate
Event
Event handler signature must match with event delegate signature
Subscribe Events
Fires Events
Subscriber 2
Event Handler
Subscribe: Event += Event_Handler

Event Publisher & Subscriber

Below is the UML - Class Diagram of "Publisher-Subscriber" or Event Driven concepts:

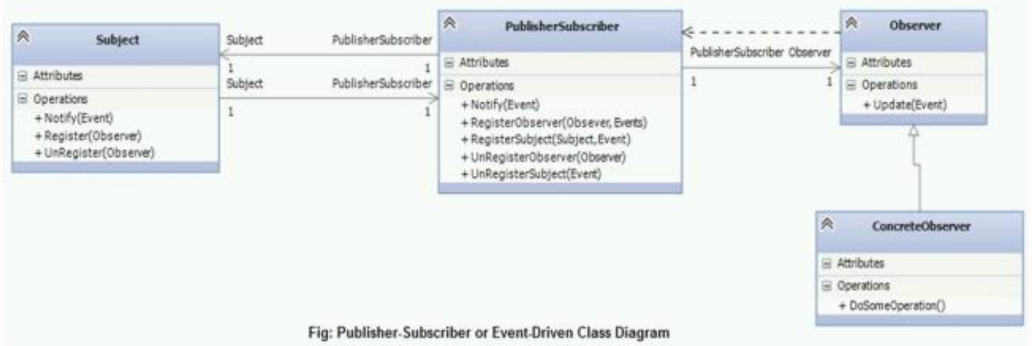


Fig: Publisher-Subscriber or Event-Driven Class Diagram

Using Delegates with Events

The events are declared and raised in a class and associated with the event handlers using delegates within the same class or some other class. The class containing the event is used to publish the event. This is called the **publisher** class. Some other class that accepts this event is called the **subscriber** class. Events use the **publisher-subscriber** model.

A **publisher** is an object that contains the definition of the event and the delegate. The event-delegate association is also defined in this object. A publisher class object invokes the event and it is notified to other objects.

A **subscriber** is an object that accepts the event and provides an event handler. The delegate in the publisher class invokes the method (event handler) of the subscriber class.

Events in C#

The Event is something special that is going to happen. Here we will take an example of an event, where Microsoft launches the events for the developer. In this Event, Microsoft wants to aware the developer about the feature of the existing or new products. For this, Microsoft will use Email or other advertisement options to aware the developer about the Event. So, in this case, Microsoft will work as a publisher who raises the Event and notifies the developers about it. Developers will work as the subscriber of the Event who handles the Event.

Similarly, in C#, Events follow the same concept. In C#, Event can be subscriber, publisher, subscriber, notification, and a handler. Generally, the User Interface uses the events. Here we will take an example of Button control in Windows. Button performs multiple events such as click, mouseover, etc. The custom class contains the Event through which we will notify the other subscriber class about the other things which is going to happen. So, in this case, we will define the Event and inform the other classes about the Event, which contains the event handler.

The event is an encapsulated delegate. C# and .NET both support the events with the delegates. When the state of the application changes, events and delegates give the notification to the client application. Delegates and Events both are tightly coupled for dispatching the events, and event handling require the implementation of the delegates. The sending event class is known as the publisher, and the receiver class or handling the Event is known as a subscriber.

1. In C#, event handler will take the two parameters as input and return the void.
2. The first parameter of the Event is also known as the source, which will publish the object.
3. The publisher will decide when we have to raise the Event, and the subscriber will determine what response we have to give.
4. Event can contain many subscribers.
5. Generally, we used the Event for the single user action like clicking on the button.

6. If the Event includes the multiple subscribers, then synchronously event handler invoked.

Declaring Events

Events are user actions such as key press, clicks, mouse movements, etc., or some occurrence such as system generated notifications. Applications need to respond to events when they occur. For example, interrupts. Events are used for inter-process communication.

Using Delegates with Events

The events are declared and raised in a class and associated with the event handlers using delegates within the same class or some other class. The class containing the event is used to publish the event. This is called the **publisher** class. Some other class that accepts this event is called the **subscriber** class. Events use the **publisher-subscriber** model.

A **publisher** is an object that contains the definition of the event and the delegate. The event-delegate association is also defined in this object. A publisher class object invokes the event and it is notified to other objects.

A **subscriber** is an object that accepts the event and provides an event handler. The delegate in the publisher class invokes the method (event handler) of the subscriber class.

Declaring Events

To declare an event inside a class, first of all, you must declare a delegate type for the even as:

```
public delegate string BoilerLogHandler(string str);
```

then, declare the event using the **event** keyword −

```
event BoilerLogHandler BoilerEventLog;
```

The preceding code defines a delegate named *BoilerLogHandler* and an event named *BoilerEventLog*, which invokes the delegate when it is raised.

```
Example:

using System;

namespace SampleApp
{
    public delegate string MyDel(string str);
    class EventProgram
    {
        event MyDel MyEvent;

        public EventProgram()
        {
            this.MyEvent += new MyDel(this.WelcomeUser);
        }
        public string WelcomeUser(string username)
        {
            return "Welcome " + username;
        }
        static void Main(string[] args)
        {
            EventProgram obj1 = new EventProgram();
            string result = obj1.MyEvent("Rajarshi Shahu Mahavidyalaya Latur");
            Console.WriteLine(result);
            Console.ReadLine();
        }
    }
}
```

When the above code is compiled and executed, it produces the following result −

Output:

When the above code is compiled and executed, it produces the following result −

```
Welcome Rajarshi Shahu Mahavidyalaya Latur
```

Events enable a class or object to notify other classes or objects when something of interest occurs. The class that sends (or *raises*) the event is called the *publisher* and the classes that receive (or *handle*) the event are called *subscribers*.

In a typical C# Windows Forms or Web application, you subscribe to events raised by controls such as buttons and list boxes. You can use the Visual C# integrated development environment (IDE) to browse the events that a control publishes and select the ones that you want to handle. The IDE provides an easy way to automatically add an empty event handler method and the code to subscribe to the event.

To declare an event inside a class, first of all, you must declare a delegate type for the even as:

```
public delegate string BoilerLogHandler(string str);
```

then, declare the event using the **event** keyword −

```
event BoilerLogHandler BoilerEventLog;
```

The preceding code defines a delegate named *BoilerLogHandler* and an event named *BoilerEventLog*, which invokes the delegate when it is raised.

Delegates:

C# delegates are similar to pointers to functions, in C or C++. A **delegate** is a reference type variable that holds the reference to a method. The reference can be changed at runtime.

Delegates are especially used for implementing events and the call-back methods. All delegates are implicitly derived from the **System.Delegate** class.

A delegate is an object which refers to a method or you can say it is a reference type variable that can hold a reference to the methods. Delegates in C# are similar to the function pointer in C/C++. It provides a way which tells which method is to be called when an event is triggered.
For example, if you click on a *Button* on a form (Windows Form application), the program would call a specific method. In simple words, it is a type that represents references to methods with a particular parameter list and return type and then calls the method in a program for execution when it is needed.
Important Points About Delegates:

- Provides a good way to encapsulate the methods.
- Delegates are the library class in System namespace.
- These are the type-safe pointer of any method.
- Delegates are mainly used in implementing the call-back methods and events.
- Delegates can be chained together as two or more methods can be called on a single event.
- It doesn’t care about the class of the object that it references.
- Delegates can also be used in “anonymous methods” invocation.
- Anonymous Methods(C# 2.0) and Lambda expressions(C# 3.0) are compiled to delegate types in certain contexts. Sometimes, these features together are known as anonymous functions.

Declaration of Delegates

Delegate type can be declared using the **delegate** keyword. Once a delegate is declared, delegate instance will refer and call those methods whose return type and parameter-list matches with the delegate declaration.
Syntax:

```
[modifier] delegate [return_type] [delegate_name] ([parameter_list]);
```

Declaring Delegates

Delegate declaration determines the methods that can be referenced by the delegate. A delegate can refer to a method, which has the same signature as that of the delegate.

For example, consider a delegate −

public delegate int MyDelegate (string s);

The preceding delegate can be used to reference any method that has a single *string* parameter and returns an *int* type variable.

Syntax for delegate declaration is −

delegate <return type> <delegate-name> <parameter list>

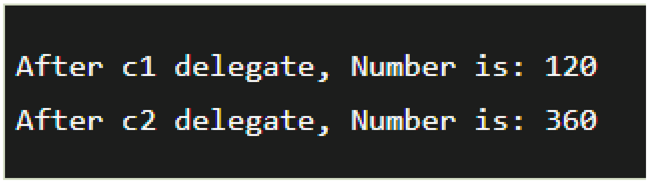
Example:

C# Delegate Example

Let's see a simple example of delegate in C# which calls add() and mul() methods.

```
using System;
delegate int Calculator(int n);//declaring delegate

public class DelegateExample
{
    static int number = 100;
    public static int add(int n)
    {
        number = number + n;
        return number;
    }
    public static int mul(int n)
    {
        number = number * n;
        return number;
    }
    public static int getNumber()
    {
        return number;
    }
    public static void Main(string[] args)
    {
        Calculator c1 = new Calculator(add);//instantiating delegate
        Calculator c2 = new Calculator(mul);
        c1(20);//calling method using delegate
        Console.WriteLine("After c1 delegate, Number is: " + getNumber());
        c2(3);
        Console.WriteLine("After c2 delegate, Number is: " + getNumber());

    }
}
```

Output:



```
After c1 delegate, Number is: 120
After c2 delegate, Number is: 360
```

Instantiating Delegates

Once a delegate type is declared, a delegate object must be created with the **new** keyword and be associated with a particular method. When creating a delegate, the argument passed to the **new** expression is written similar to a method call, but without the arguments to the method. For example −

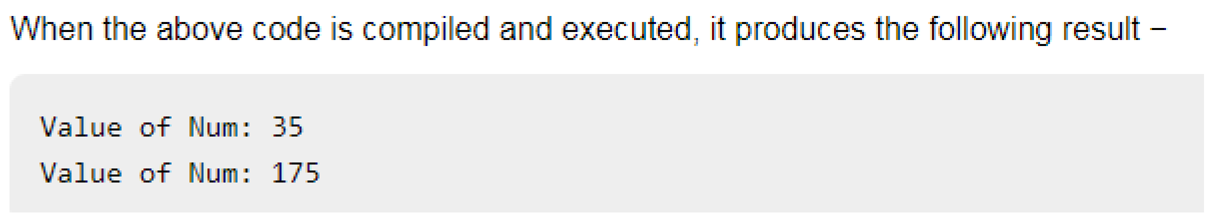
```
public delegate void printString(string s);
...
printString ps1 = new printString(WriteToScreen);
printString ps2 = new printString(WriteToFile);
```

Following example demonstrates declaration, instantiation, and use of a delegate that can be used to reference methods that take an integer parameter and returns an integer value.

Example:

```
using System;

delegate int NumberChanger(int n);
namespace DelegateAppl
{

    class TestDelegate
    {
        static int num = 10;

        public static int AddNum(int p)
        {
            num += p;
            return num;
        }
        public static int MultNum(int q)
        {
            num *= q;
            return num;
        }
        public static int getNum()
        {
            return num;
        }
        static void Main(string[] args)
        {
            //create delegate instances
            NumberChanger nc1 = new NumberChanger(AddNum);
            NumberChanger nc2 = new NumberChanger(MultNum);

            //calling the methods using the delegate objects
            nc1(25);
            Console.WriteLine("Value of Num: {0}", getNum());
            nc2(5);
            Console.WriteLine("Value of Num: {0}", getNum());
            Console.ReadKey();
        }
    }
}
```

Output:



When the above code is compiled and executed, it produces the following result −

```
Value of Num: 35
Value of Num: 175
```

Multicast Delegates:

Multicasting of a Delegate

Delegate objects can be composed using the "+" operator. A composed delegate calls the two delegates it was composed from. Only delegates of the same type can be composed. The "-" operator can be used to remove a component delegate from a composed delegate.

Using this property of delegates you can create an invocation list of methods that will be called when a delegate is invoked. This is called **multicasting** of a delegate. The following program demonstrates multicasting of a delegate −

Example:

```
using System;

delegate int NumberChanger(int n);
namespace DelegateAppl {
   class TestDelegate {
      static int num = 10;

      public static int AddNum(int p) {
         num += p;
         return num;
      }
      public static int MultNum(int q) {
         num *= q;
         return num;
      }
      public static int getNum() {
         return num;
      }
      static void Main(string[] args) {
         //create delegate instances
         NumberChanger nc;
         NumberChanger nc1 = new NumberChanger(AddNum);
         NumberChanger nc2 = new NumberChanger(MultNum);

         nc = nc1;
         nc += nc2;

         //calling multicast
         nc(5);
         Console.WriteLine("Value of Num: {0}", getNum());
         Console.ReadKey();
      }
   }
}
```

Output:

When the above code is compiled and executed, it produces the following result –

```
Value of Num: 75
```

Creating & Starting thread:

A **thread** is defined as the execution path of a program. Each thread defines a unique flow of control. If your application involves complicated and time consuming operations, then it is often helpful to set different execution paths or threads, with each thread performing a particular job.

Threads are **lightweight processes**. One common example of use of thread is implementation of concurrent programming by modern operating systems. Use of threads saves wastage of CPU cycle and increase efficiency of an application.

So far we wrote the programs where a single thread runs as a single process which is the running instance of the application. However, this way the application can perform one job at a time. To make it execute more than one task at a time, it could be divided into smaller threads.

Thread Life Cycle

The life cycle of a thread starts when an object of the System.Threading.Thread class is created and ends when the thread is terminated or completes execution.

Following are the various states in the life cycle of a thread −

- **The Unstarted State −** It is the situation when the instance of the thread is created but the Start method is not called.
- **The Ready State −** It is the situation when the thread is ready to run and waiting CPU cycle.
- **The Not Runnable State −** A thread is not executable, when
    - Sleep method has been called
    - Wait method has been called
    - Blocked by I/O operations
- **The Dead State −** It is the situation when the thread completes execution or is aborted.

The Main Thread

In C#, the **System.Threading.Thread** class is used for working with threads. It allows creating and accessing individual threads in a multithreaded application. The first thread to be executed in a process is called the **main** thread.

When a C# program starts execution, the main thread is automatically created. The threads created using the **Thread** class are called the child threads of the main thread. You can access a thread using the **CurrentThread** property of the Thread class.

The following program demonstrates main thread execution –

Example:

```
using System;
using System.Threading;

namespace MultithreadingApplication
{
    class MainThreadProgram
    {
        static void Main(string[] args)
        {
            Thread th = Thread.CurrentThread;
            th.Name = "MainThread";

            Console.WriteLine("This is {0}", th.Name);
            Console.ReadKey();
        }
    }
}
```

Output:

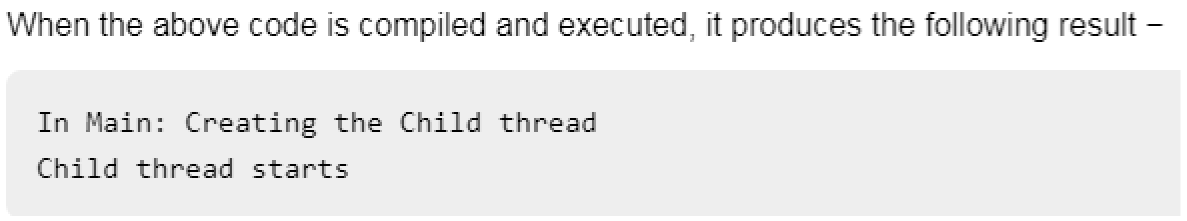
```
This is MainThread
```

Creating Threads

Threads are created by extending the Thread class. The extended Thread class then calls the **Start()** method to begin the child thread execution.

The following program demonstrates the concept –

```
using System;
using System.Threading;

namespace MultithreadingApplication {
  class ThreadCreationProgram {
    public static void CallToChildThread() {
      Console.WriteLine("Child thread starts");
    }
    static void Main(string[] args) {
      ThreadStart childref = new ThreadStart(CallToChildThread);
      Console.WriteLine("In Main: Creating the Child thread");
      Thread childThread = new Thread(childref);
      childThread.Start();
      Console.ReadKey();
    }
  }
}
```


When the above code is compiled and executed, it produces the following result −

```
In Main: Creating the Child thread
Child thread starts
```

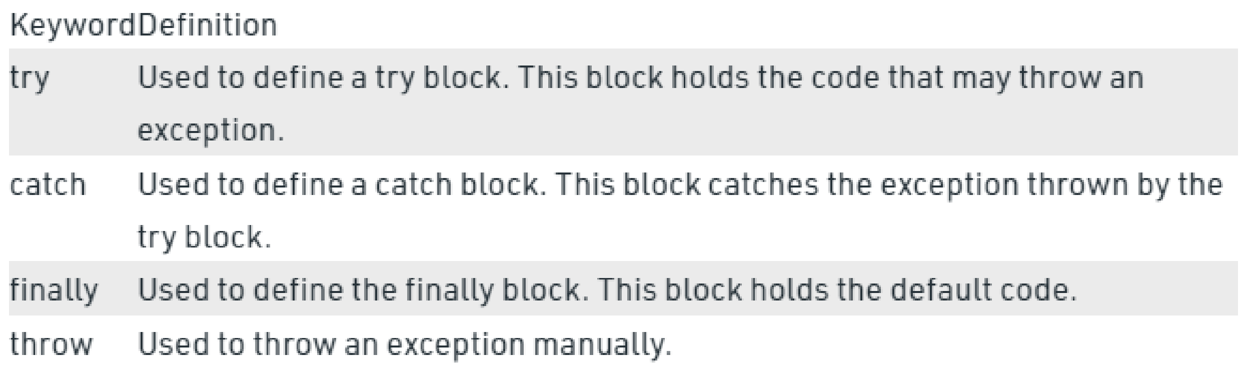
Exception handling:

An exception is defined as an event that occurs during the execution of a program that is unexpected by the program code. The actions to be performed in case of occurrence of an exception is not known to the program. In such a case, we create an exception object and call the exception handler code. The execution of an exception handler so that the program code does not crash is called exception handling. Exception handling is important because it gracefully handles an unwanted event, an exception so that the program code still makes sense to the user.

An exception is a problem that arises during the execution of a program. A C# exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

Exceptions provide a way to transfer control from one part of a program to another. C# exception handling is built upon four keywords: **try**, **catch**, **finally**, and **throw**.

- **try** − A try block identifies a block of code for which particular exceptions is activated. It is followed by one or more catch blocks.
- **catch** − A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.
- **finally** − The finally block is used to execute a given set of statements, whether an exception is thrown or not thrown. For example, if you open a file, it must be closed whether an exception is raised or not.
- **throw** − A program throws an exception when a problem shows up. This is done using a throw keyword.



Keyword	Definition
try	Used to define a try block. This block holds the code that may throw an exception.
catch	Used to define a catch block. This block catches the exception thrown by the try block.
finally	Used to define the finally block. This block holds the default code.
throw	Used to throw an exception manually.

C# Exceptions

When executing C# code, different errors can occur: coding errors made by the programmer, errors due to wrong input, or other unforeseeable things.

When an error occurs, C# will normally stop and generate an error message. The technical term for this is: C# will throw an **exception** (throw an error).

C# try and catch

The `try` statement allows you to define a block of code to be tested for errors while it is being executed.

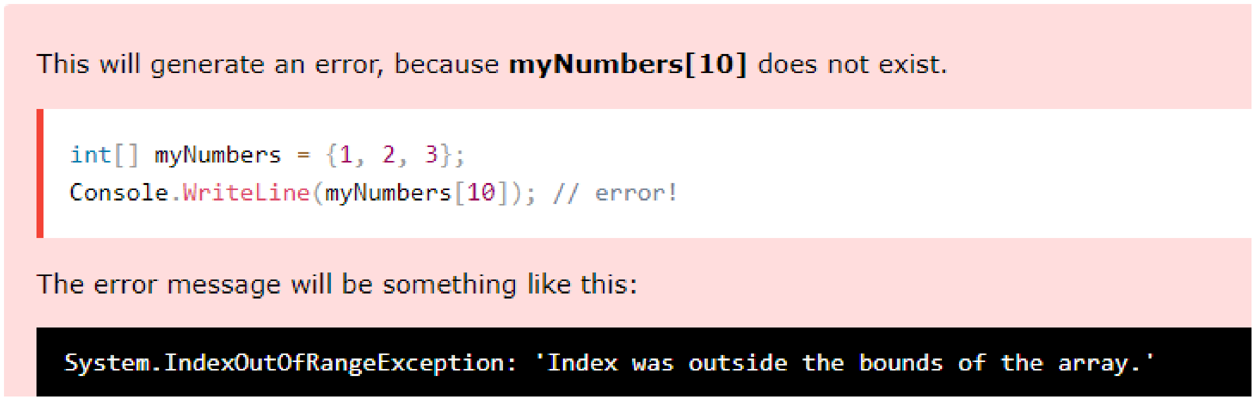
The `catch` statement allows you to define a block of code to be executed, if an error occurs in the try block.

The `try` and `catch` keywords come in pairs:

Syntax

```
try

{

  //  Block of code to try

}

catch (Exception e)

{

  //  Block of code to handle errors

}
```

Consider the following example, where we create an array of three integers:



This will generate an error, because **myNumbers[10]** does not exist.

```
int[] myNumbers = {1, 2, 3};
Console.WriteLine(myNumbers[10]); // error!
```

The error message will be something like this:

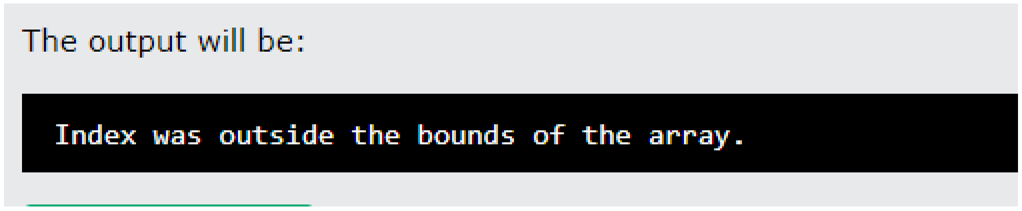
```
System.IndexOutOfRangeException: 'Index was outside the bounds of the array.'
```

If an error occurs, we can use `try...catch` to catch the error and execute some code to handle it.

In the following example, we use the variable inside the catch block (`e`) together with the built-in `Message` property, which outputs a message that describes the exception:

Example

```
try

{

  int[] myNumbers = {1, 2, 3};

  Console.WriteLine(myNumbers[10]);

}

catch (Exception e)

{

  Console.WriteLine(e.Message);

}
```



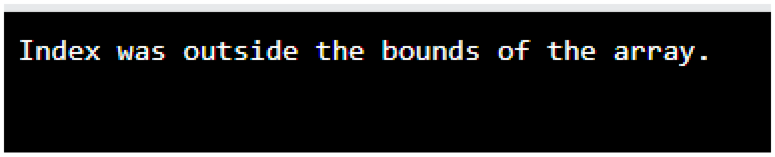
The output will be:

```
Index was outside the bounds of the array.
```

Example:

```
using System;

namespace MyApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                int[] myNumbers = { 1, 2, 3 };
                Console.WriteLine(myNumbers[10]);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

Output:

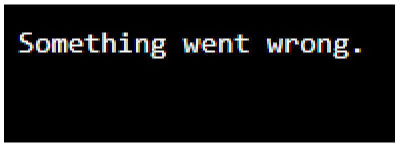


```
Index was outside the bounds of the array.
```

You can also output your own error message:

```
using System;

namespace MyApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                int[] myNumbers = { 1, 2, 3 };
                Console.WriteLine(myNumbers[10]);
            }
            catch (Exception e)
            {
                Console.WriteLine("Something went wrong.");
            }
        }
    }
}
```

Output:



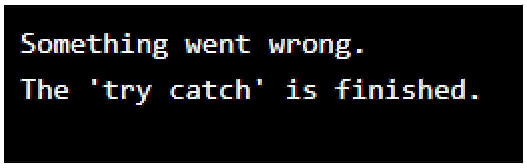
```
Something went wrong.
```

Finally

The `finally` statement lets you execute code, after `try...catch`, regardless of the result:

```
using System;

namespace MyApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                int[] myNumbers = { 1, 2, 3 };
                Console.WriteLine(myNumbers[10]);
            }
            catch (Exception e)
            {
                Console.WriteLine("Something went wrong.");
            }
            finally
            {
                Console.WriteLine("The 'try catch' is finished.");
            }
        }
    }
}
```

Output:



```
Something went wrong.
The 'try catch' is finished.
```

Chapter 4: Windows Form

Windows Form:

Windows Forms is a Graphical User Interface(GUI) class library which is bundled in *.Net Framework*. Its main purpose is to provide an easier interface to develop the applications for desktop, tablet, PCs. It is also termed as the **WinForms**. The applications which are developed by using Windows Forms or WinForms are known as the **Windows Forms Applications** that runs on the desktop computer. WinForms can be used only to develop the Windows Forms Applications not web applications. WinForms applications can contain the different type of controls like labels, list boxes, tooltip etc.

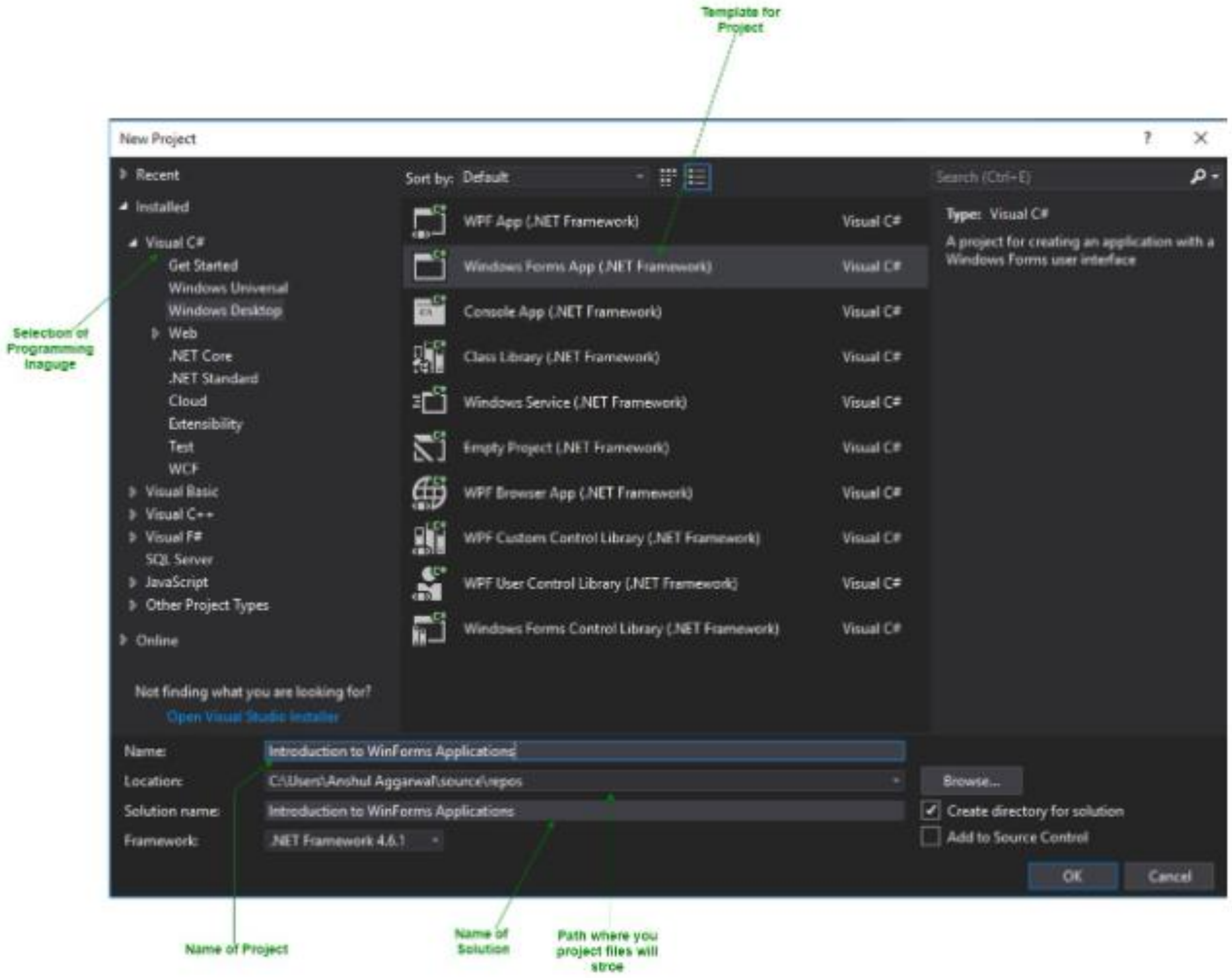
A windows form application is an application, which is designed to run on a computer. It will not run on web browser because then it becomes a web application.

A Windows forms application is one that runs on the desktop computer. A Windows forms application will normally have a collection of controls such as labels, textboxes, list boxes, etc.

Below is an example of a simple Windows form application C#. It shows a simple Login screen, which is accessible by the user. The user will enter the required credentials and then will click the Login button to proceed.

Creating a Windows Forms Application Using Visual Studio 2017

- First, open the Visual Studio then Go to **File -> New -> Project** to create a new project and then select the language as *Visual C#* from the left menu. Click on *Windows Forms App(.NET Framework)* in the middle of current window. After that give the project name and Click **OK**.


Template for Project
Selection of Programming inaguge
Name of Project
Name of Solution
Path where you project files will stroe
New Project
Recent
Installed
Visual C#
Get Started
Windows Universal
Windows Desktop
Web
.NET Core
.NET Standard
Cloud
Extensibility
Test
WCF
Visual Basic
Visual C++
Visual F#
SQL Server
JavaScript
Other Project Types
Online
Not finding what you are looking for?
Open Visual Studio Installer
Sort by: Default
Search (Ctrl+E)
WPF App (.NET Framework) Visual C#
Windows Forms App (.NET Framework) Visual C#
Console App (.NET Framework) Visual C#
Class Library (.NET Framework) Visual C#
Windows Service (.NET Framework) Visual C#
Empty Project (.NET Framework) Visual C#
WPF Browser App (.NET Framework) Visual C#
WPF Custom Control Library (.NET Framework) Visual C#
WPF User Control Library (.NET Framework) Visual C#
Windows Forms Control Library (.NET Framework) Visual C#
Browse...
Create directory for solution
Add to Source Control
OK
Cancel

Type: Visual C#

A project for creating an application with a Windows Forms user interface

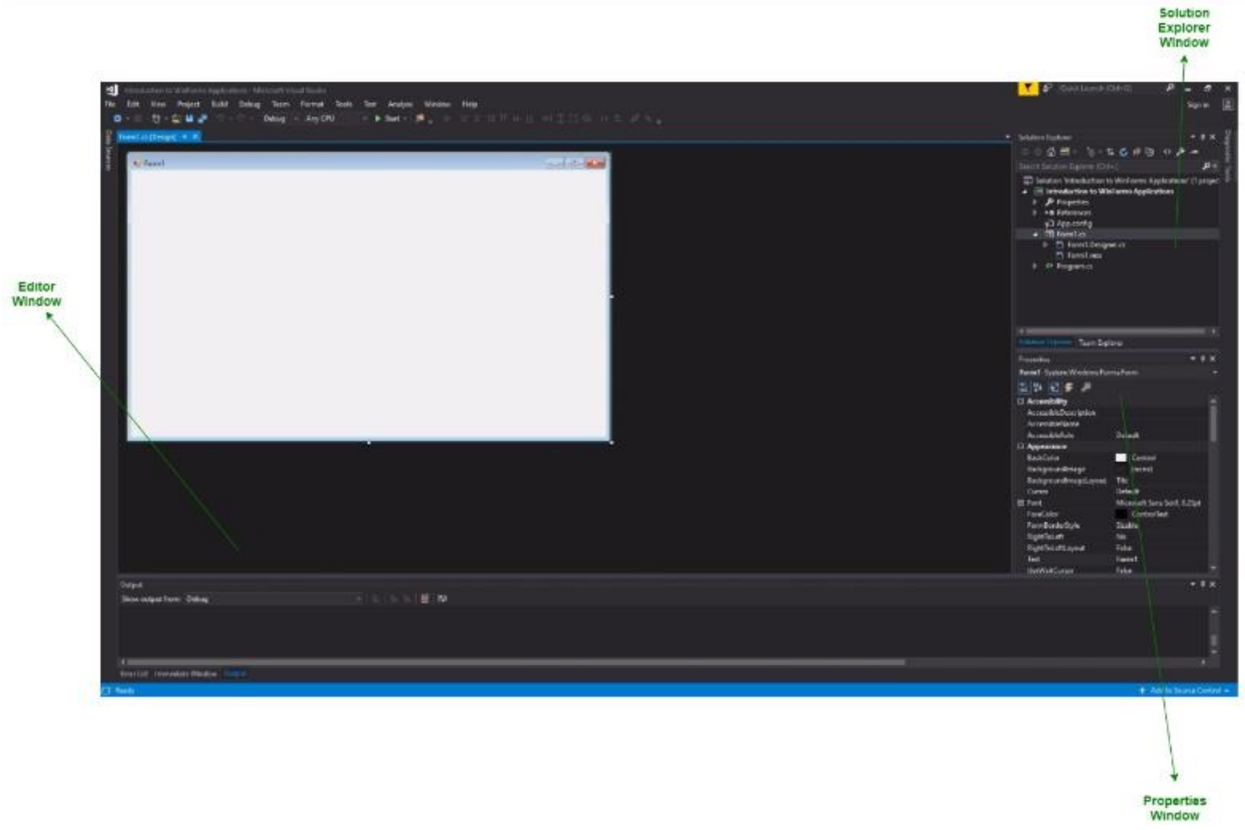
Name: Introduction to WinForms Applications

Location: C:\Users\Anshul Aggarwal\source\repos

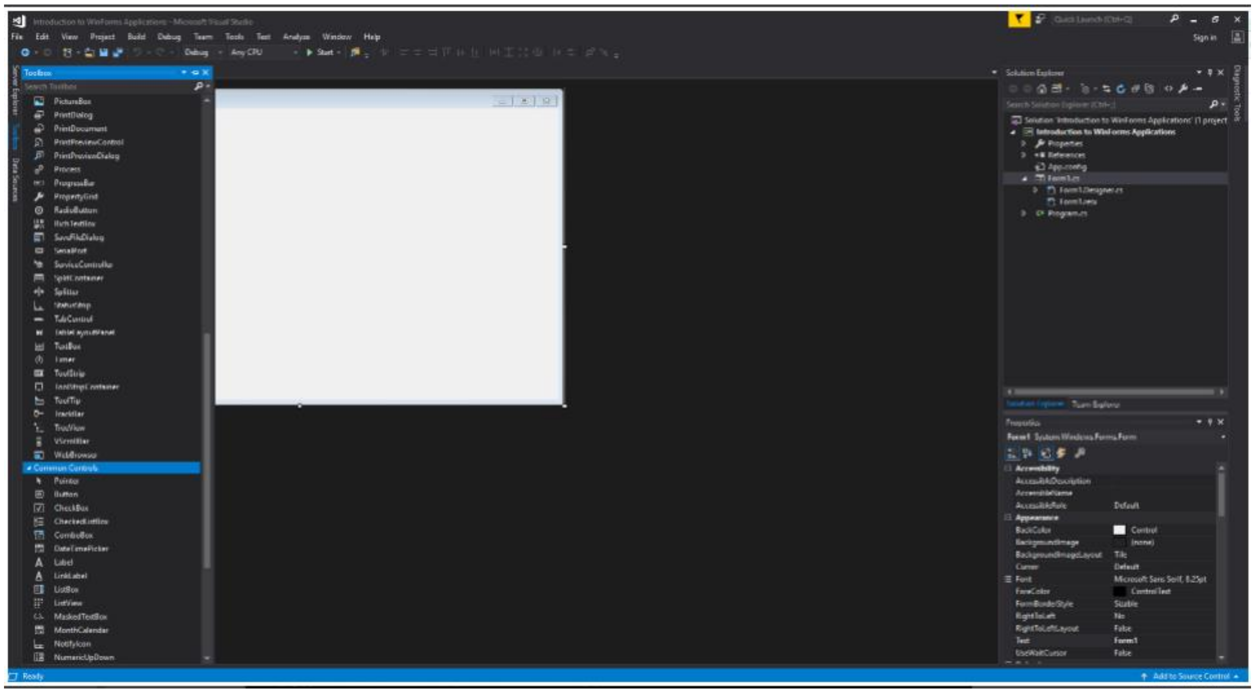
Solution name: Introduction to WinForms Applications

Framework: .NET Framework 4.6.1

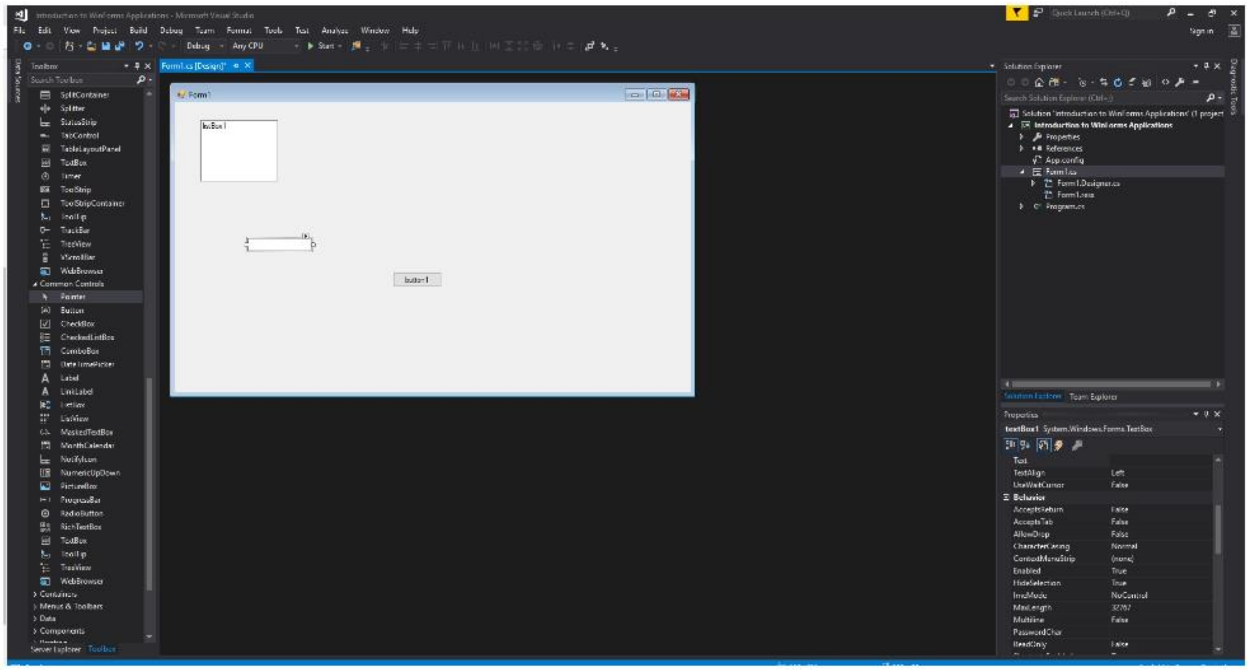
- Here the solution is like a container which contains the projects and files that may be required by the program.
- After that following window will display which will be divided into three parts as follows:
  1. **Editor Window or Main Window:** Here, you will work with forms and code editing. You can notice the layout of form which is now blank. You will double click the form then it will open the code for that.
  2. **Solution Explorer Window:** It is used to navigate between all items in solution. For example, if you will select a file form this window then particular information will be display in the property window.
  3. **Properties Window:** This window is used to change the different properties of the selected item in the Solution Explorer. Also, you can change the properties of components or controls that you will add to the forms.



- You can also reset the window layout by setting it to default. To set the default layout, go to **Window -> Reset Window Layout** in Visual Studio Menu.
- Now **to add the controls to your WinForms application** go to **Toolbox** tab present in the extreme left side of Visual Studio. Here, you can see a list of controls. To access the most commonly used controls go to **Common Controls** present in Toolbox tab.



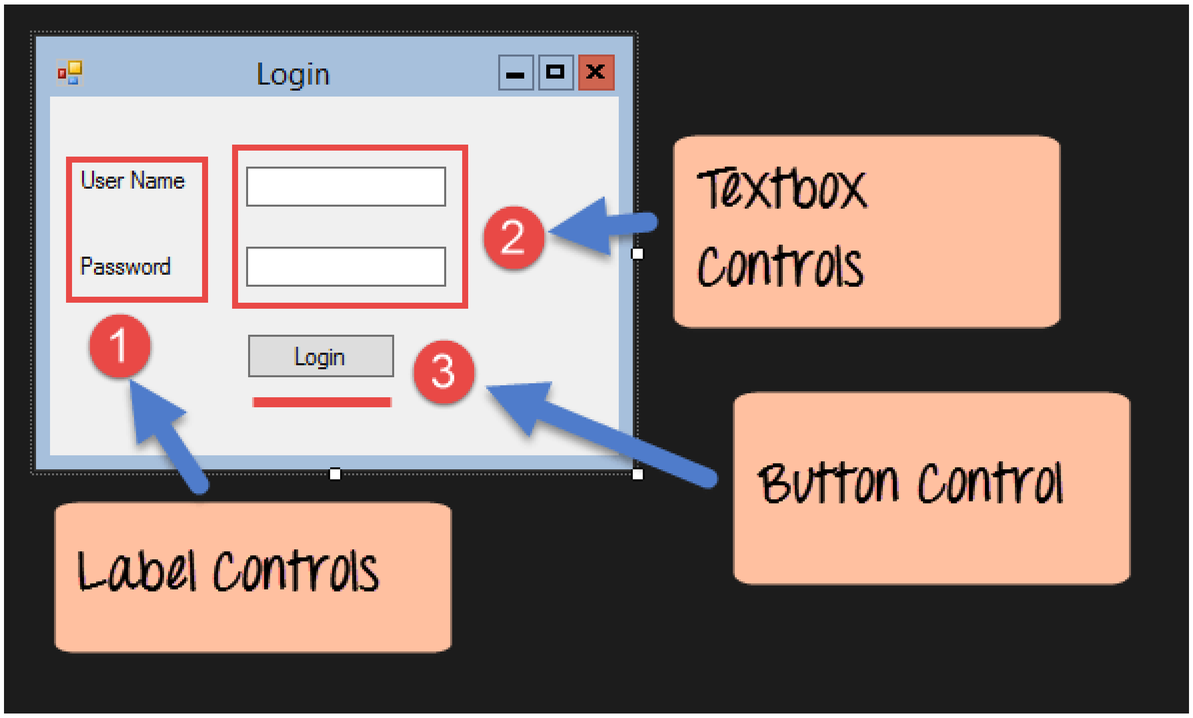
Now drag and drop the controls that you needed on created Form. For example, if you can add TextBox, ListBox, Button etc. as shown below. By clicking on the particular dropped control you can see and change its properties present in the right most corner of Visual Studio.



- In the above image, you can see the TextBox is selected and its properties like TextAlign, MaxLength etc. are opened in right most corner.

You can change its properties' values as per the application need. The code of controls will be automatically added in the background. You can check the *Form1.Designer.cs* file present in the Solution Explorer Window.

- To run the program you can use an **F5 key** or **Play button** present in the toolbar of Visual Studio. To stop the program you can use pause button present in the ToolBar. You can also run the program by going to **Debug->Start Debugging** menu in the menubar.



1. This is a collection of label controls which are normally used to describe adjacent controls. So in our case, we have 2 textboxes, and the labels are used to tell the user that one textbox is for entering the user name and the other for the password.
2. The 2 textboxes are used to hold the username and password which will be entered by the user.
3. Finally, we have the button control. The button control will normally have some code attached to perform a certain set of actions. So for example in the above case, we could have the button perform an action of validating the user name and password which is entered by the user.

MDI Form:

Multiple-document interface (MDI) applications enable you to display multiple documents at the same time, with each document displayed in its own window. MDI applications often have a Window menu item with submenus for switching between windows or documents.

The Multiple-Document Interface (MDI) is a specification that defines a user interface for applications that enable the user to work with more than one document at the same time under one parent form (window).

Visualize the working style of an application in which you are allowed to open multiple forms in one parent container window, and all the open forms will get listed under the Windows menu. Whereas having an individual window for each instance of the same application is termed as single document interface (SDI); applications such as Notepad, Microsoft Paint, Calculator, and so on, are SDI applications. SDI applications get opened only in their own windows and can become difficult to manage, unlike when you have multiple documents or forms open inside one MDI interface.

Hence, MDI applications follow a parent form and child form relationship model. MDI applications allow you to open, organize, and work with multiple documents at the same time by opening them under the context of the MDI parent form; therefore, once opened, they can't be dragged out of it like an individual form.

The parent (MDI) form organizes and arranges all the child forms or documents that are currently open. You might have seen such options in many Windows applications under a Windows menu, such as Cascade, Tile Vertical, and so on.

How to: Create MDI Parent Forms

The foundation of a Multiple-Document Interface (MDI) application is the MDI parent form. This is the form that contains the MDI child windows, which are the sub-windows wherein the user interacts with the MDI application. Creating an MDI parent form is easy, both in the Windows Forms Designer and programmatically.

Create an MDI parent form at design time

1. Create a Windows Application project in Visual Studio.
2. In the **Properties** window, set the IsMdiContainer property to **true**.

This designates the form as an MDI container for child windows.

3. From the **Toolbox**, drag a **MenuStrip** control to the form. Create a top-level menu item with the **Text** property set to **&File** with submenu items called **&New** and **&Close**. Also create a top-level menu item called **&Window**.

   The first menu will create and hide menu items at run time, and the second menu will keep track of the open MDI child windows. At this point, you have created an MDI parent window.

4. Press **F5** to run the application. For information about creating MDI child windows that operate within the MDI parent form, see How to: Create MDI Child Forms.

How to: Create MDI child forms

MDI child forms are an essential element of Multiple-Document Interface (MDI) applications, as these forms are the center of user interaction.

In the following procedure, you'll use Visual Studio to create an MDI child form that displays a RichTextBox control, similar to most word-processing applications. By substituting the System.Windows.Forms control with other controls, such as the DataGridView control, or a mixture of controls, you can create MDI child windows (and, by extension, MDI applications) with diverse possibilities.

Create MDI child forms

1. Create a new Windows Forms application project in Visual Studio. In the **Properties** window for the form, set its IsMdiContainer property to `true` and its `WindowsState` property to `Maximized`.

   This designates the form as an MDI container for child windows.

2. From the `Toolbox`, drag a MenuStrip control to the form. Set its `Text` property to **File**.
3. Click the ellipsis (…) next to the **Items** property, and click **Add** to add two child tool strip menu items. Set the `Text` property for these items to **New** and **Window**.
4. In **Solution Explorer**, right-click the project, and then select **Add** > **New Item**.

5. In the **Add New Item** dialog box, select **Windows Form** (in Visual Basic or in Visual C#) or **Windows Forms Application (.NET)** (in Visual C++) from the **Templates** pane. In the **Name** box, name the form **Form2**. Select **Open** to add the form to the project.
6. From the **Toolbox**, drag a **RichTextBox** control to the form.
7. In the **Properties** window, set the `Anchor` property to **Top, Left** and the `Dock` property to **Fill**.

   This causes the RichTextBox control to completely fill the area of the MDI child form, even when the form is resized.

8. Double click the **New** menu item to create a Click event handler for it.
9. Insert code similar to the following to create a new MDI child form when the user clicks the **New** menu item.
10. In the drop-down list at the top of the **Properties** window, select the menu strip that corresponds to the **File** menu strip and set the MdiWindowListItem property to the Window ToolStripMenuItem.

    This enables the **Window** menu to maintain a list of open MDI child windows with a check mark next to the active child window.

11. Press **F5** to run the application. By selecting **New** from the **File** menu, you can create new MDI child forms, which are kept track of in the **Window** menu item.

Menustrip:

The MenuStrip class is the foundation of menus functionality in Windows Forms. If you have worked with menus in .NET 1.0 and 2.0, you must be familiar with the MainMenu control. In .NET 3.5 and 4.0, the MainMenu control is replaced with the MenuStrip control.

Creating a MenuStrip

We can create a MenuStrip control using a Forms designer at design-time or using the MenuStrip class in code at run-time or dynamically.

To create a MenuStrip control at design-time, you simply drag and drop a MenuStrip control from Toolbox to a Form in Visual Studio. After you drag and drop a MenuStrip on a Form, the MenuStrip1 is added to the Form and looks like Figure 1. Once a MenuStrip is on the Form, you can add menu items and set its properties and events.

MenuStrip adds a menu bar to your Windows Forms program. With this control, we add a menu area and then add the default menus or create custom menus directly in Visual Studio. We demonstrate the MenuStrip and provide some usage tips.

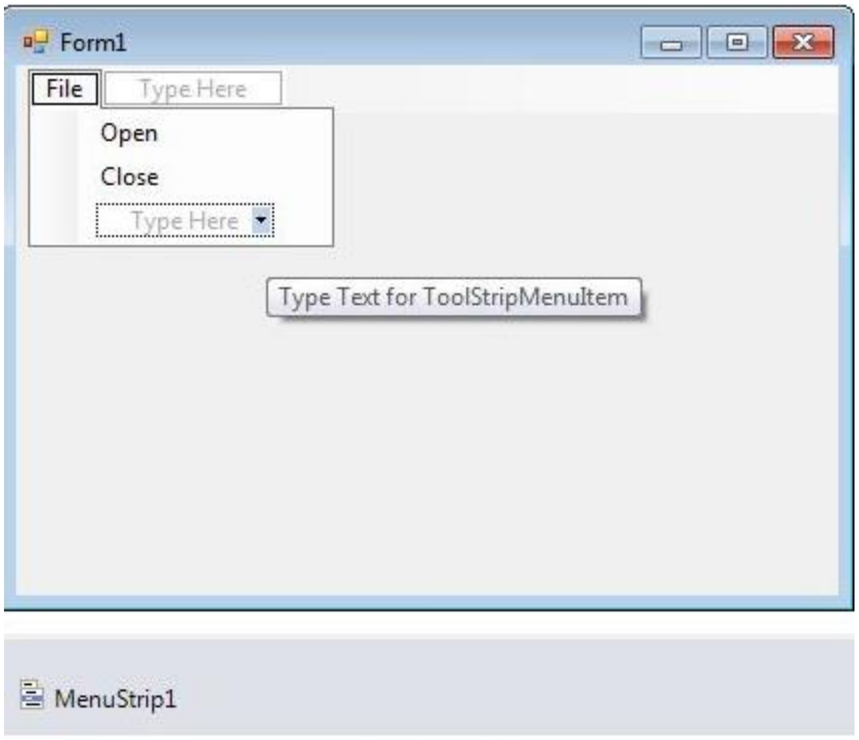


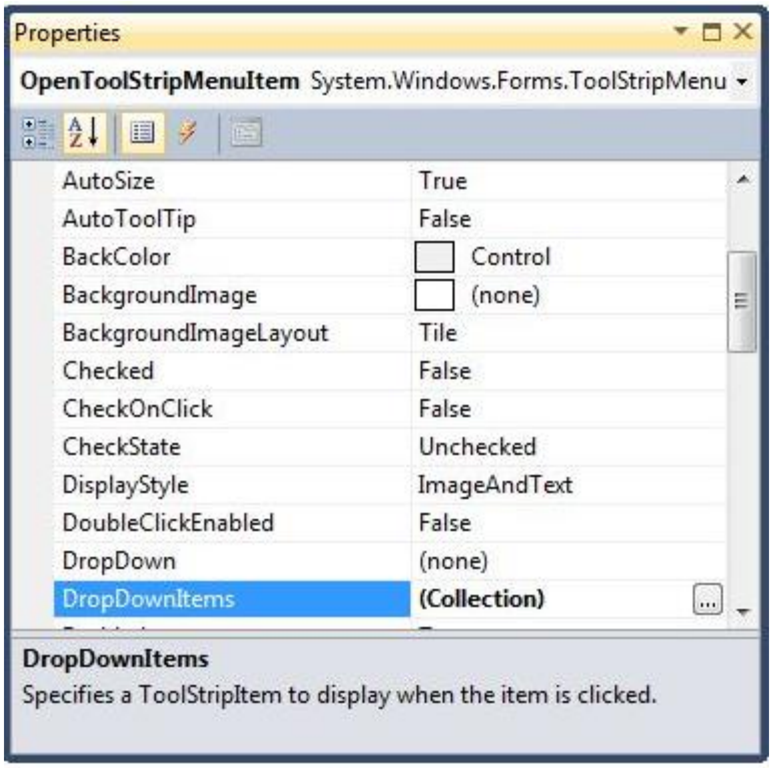
Figure 1

Creating a MenuStrip control at run-time is merely a work of creating an instance of MenuStrip class, setting its properties and adding MenuStrip class to the Form controls.

The first step to create a dynamic MenuStrip is to create an instance of MenuStrip class. The following code snippet creates a MenuStrip control object.

After you place a MenuStrip control on a Form, the next step is to set properties.

The easiest way to set properties is from the Properties Window. You can open Properties window by pressing F4 or right clicking on a control and selecting Properties menu item. The Properties window looks like Figure 2.


Properties

OpenToolStripMenuItem System.Windows.Forms.ToolStripMenu

AutoSize	True
AutoToolTip	False
BackColor	Control
BackgroundImage	(none)
BackgroundImageLayout	Tile
Checked	False
CheckOnClick	False
CheckState	Unchecked
DisplayStyle	ImageAndText
DoubleClickEnabled	False
DropDown	(none)
DropDownItems	**(Collection)**

DropDownItems
Specifies a ToolStripItem to display when the item is clicked.

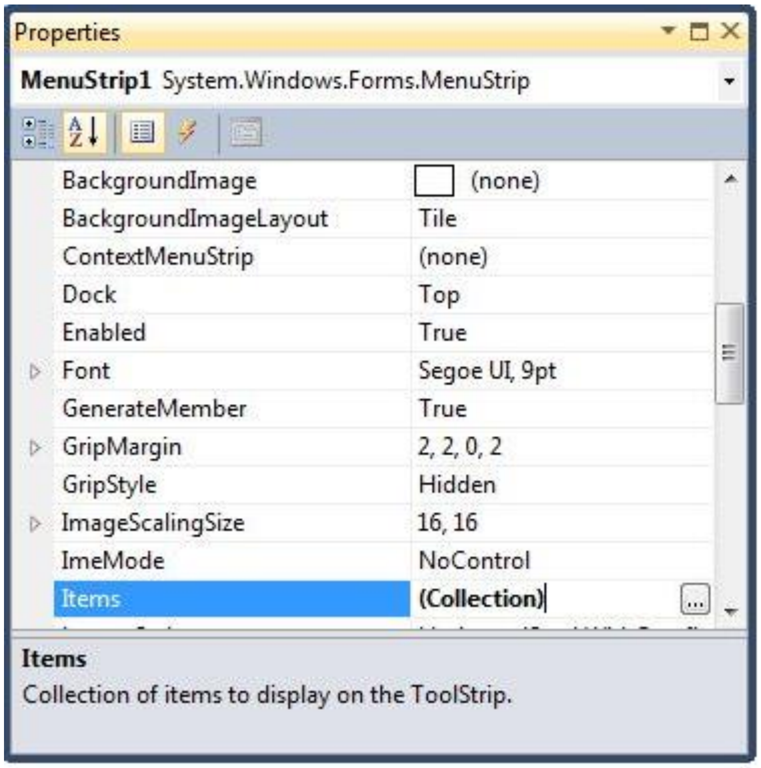
Figure 2

Positioning a MenuStrip

The Dock property is used to set the position of a MenuStrip. It is of type DockStyle that can have values Top, Bottom, Left, Right, and Fill. The following code snippet sets Location, Width, and Height properties of a MenuStrip control.

MenuStrip Items

A Menu control is nothing without menu items. The Items property is used to add and work with items in a MenuStrip. We can add items to a MenuStrip at design-time from Properties Window by clicking on Items Collection as you can see in Figure 4.



Properties

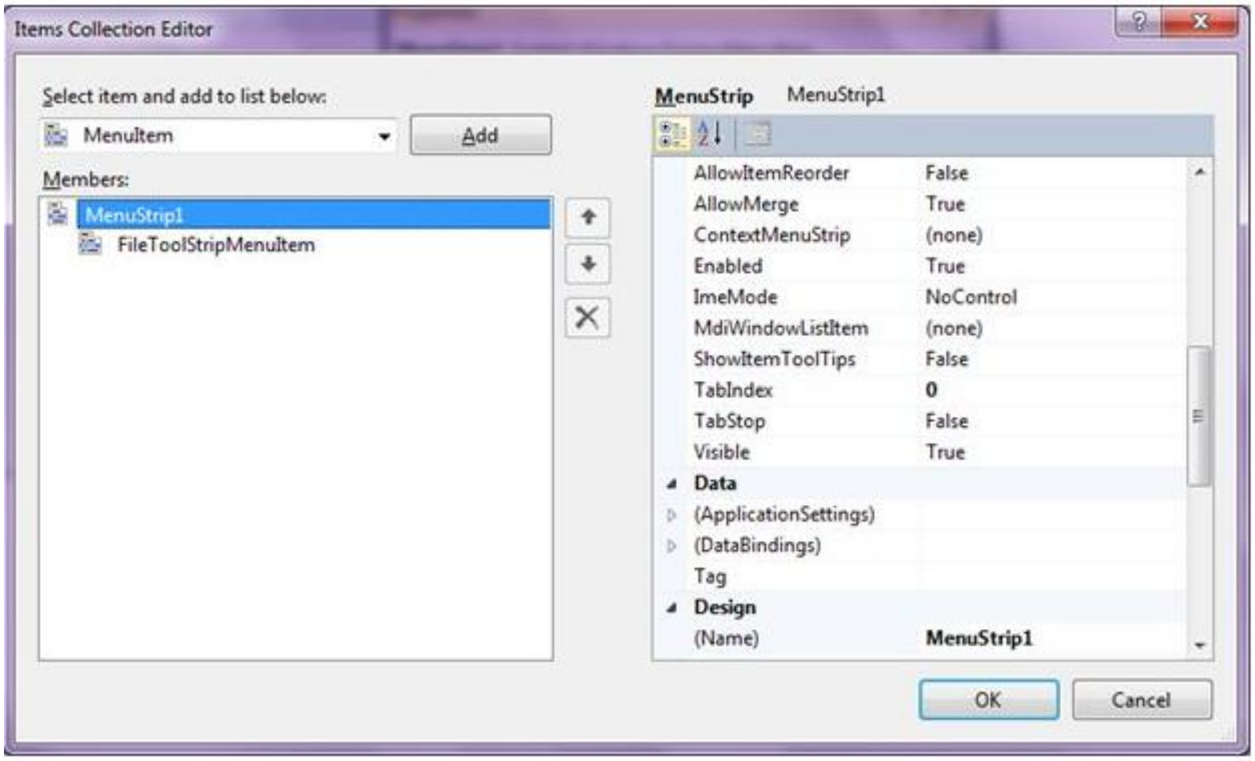
MenuStrip1 System.Windows.Forms.MenuStrip

BackgroundImage	(none)
BackgroundImageLayout	Tile
ContextMenuStrip	(none)
Dock	Top
Enabled	True
Font	Segoe UI, 9pt
GenerateMember	True
GripMargin	2, 2, 0, 2
GripStyle	Hidden
ImageScalingSize	16, 16
ImeMode	NoControl
Items	(Collection)

Items
Collection of items to display on the ToolStrip.

Figure 4

When you click on the Collections, the String Collection Editor window will pop up where you can type strings. Each line added to this collection will become a MenuStrip item. I add four items as you can see from Figure 5.



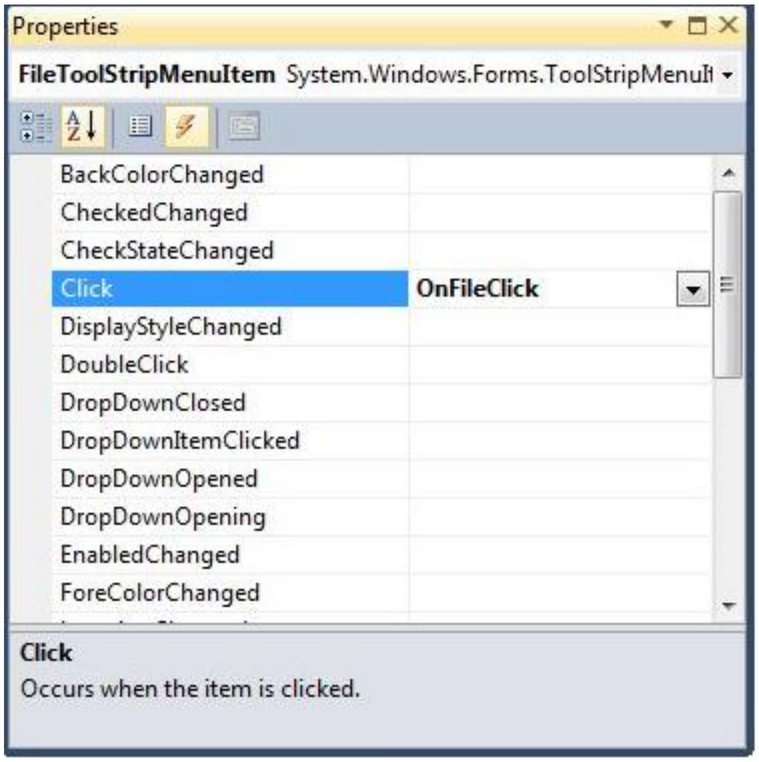
AllowItemReorder	False
AllowMerge	True
ContextMenuStrip	(none)
Enabled	True
ImeMode	NoControl
MdiWindowListItem	(none)
ShowItemToolTips	False
TabIndex	0
TabStop	False
Visible	True
Data	
(ApplicationSettings)	
(DataBindings)	
Tag	
Design	
(Name)	**MenuStrip1**

Figure 5

Adding Menu Item Click Event Handler

The main purpose of a menu item is to add a click event handler and write code that we need to execute on the menu item click event handler. For example, on File >> New menu item click event handler, we may want to create a new file.

To add an event handler, you go to Events window and double click on Click and other as you can see in Figure 6.



BackColorChanged	
CheckedChanged	
CheckStateChanged	
Click	OnFileClick
DisplayStyleChanged	
DoubleClick	
DropDownClosed	
DropDownItemClicked	
DropDownOpened	
DropDownOpening	
EnabledChanged	
ForeColorChanged	

Click
Occurs when the item is clicked.

Figure 6

Menustrip:

The MenuStrip class is the foundation of menus functionality in Windows Forms. If you have worked with menus in .NET 1.0 and 2.0, you must be familiar with the MainMenu control. In .NET 3.5 and 4.0, the MainMenu control is replaced with the MenuStrip control.

Creating a MenuStrip

We can create a MenuStrip control using a Forms designer at design-time or using the MenuStrip class in code at run-time or dynamically.

To create a MenuStrip control at design-time, you simply drag and drop a MenuStrip control from Toolbox to a Form in Visual Studio. After you drag and drop a MenuStrip on a Form, the MenuStrip1 is added to the Form and looks like Figure 1. Once a MenuStrip is on the Form, you can add menu items and set its properties and events.

ToolStrip:

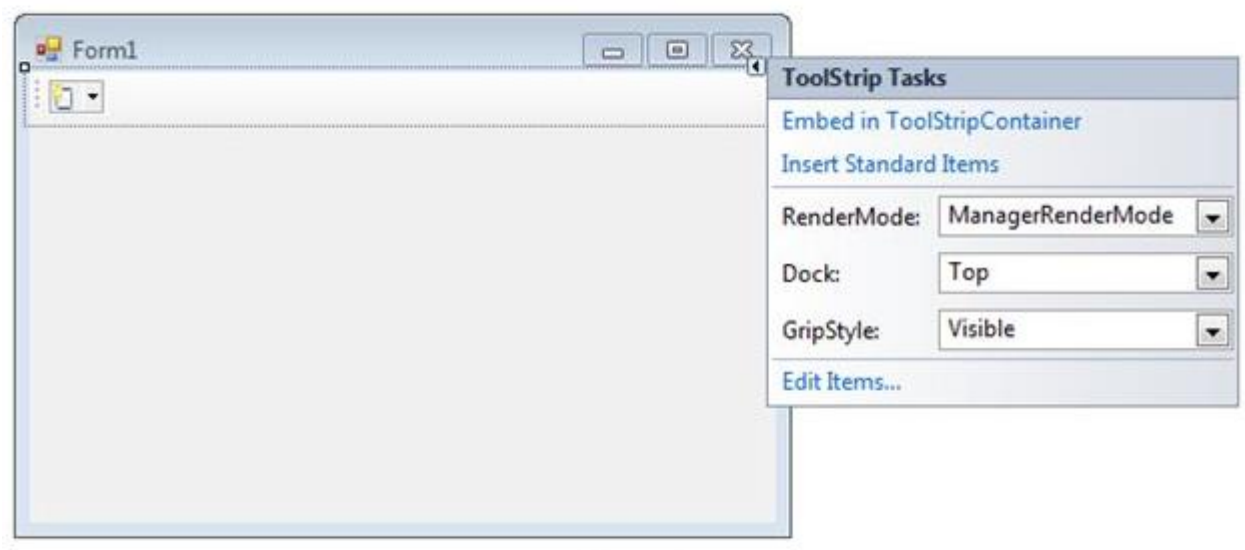
ToolStrip control provides functionality of Windows toolbar controls in Visual Studio 2010. ToolStrip class represents a ToolStrip control in Windows Forms and serves as a base class of MenuStrip, StatusStrip, and ContextMenuStrip classes.

In this article, I will discuss how to create and use a ToolStrip control and use its properties and methods.

We can create a ToolStrip control at design-time using Visual Studio designer or using the ToolStrip class at run-time.

Using Designer

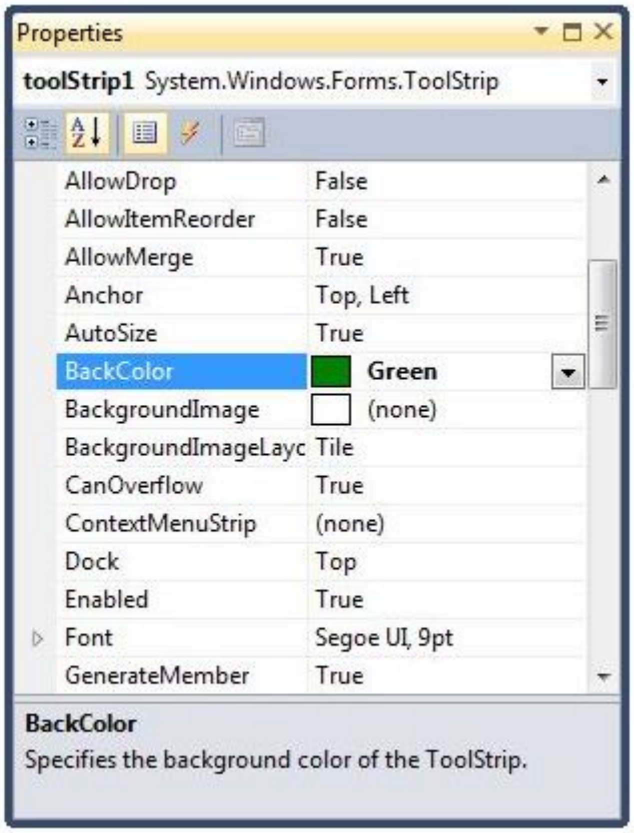
To create a ToolStrip control at design-time, simply drag and drop a ToolStrip control from Toolbox onto a Form. By default, a ToolStrip control is docked at the top of the Form. We can change docking and other properties using the Tasks control as shown in Figure 1.



RenderMode: ManagerRenderMode
Dock: Top
GripStyle: Visible

Figure 1

The next step is to set ToolStrip control properties. We can set properties by calling Properties window by using Right click on the control and selecting Properties menu item or simply hitting F4. Figure 2 shows Properties window where I set BackColor to green and GripStyle to Visible.



Properties

toolStrip1 System.Windows.Forms.ToolStrip

AllowDrop	False
AllowItemReorder	False
AllowMerge	True
Anchor	Top, Left
AutoSize	True
BackColor	🟩 **Green**
BackgroundImage	☐ (none)
BackgroundImageLayc	Tile
CanOverflow	True
ContextMenuStrip	(none)
Dock	Top
Enabled	True
Font	Segoe UI, 9pt
GenerateMember	True

BackColor
Specifies the background color of the ToolStrip.

Figure 2

A ToolStrip control is nothing but a container without adding its child controls. A ToolStrip control is capable of hosting Button, Label, SplitButton, DropDownButton, Separator, ComboBox, TextBox and ProgressBar controls.

Now let's add a few controls to ToolStrip control.

If you click on a little dropdown handle on a ToolStrip control in designer, you will see a list of possible controls (See Figure 3) that can be added to a ToolStrip. To add a control, simply select that control and it will be added to the ToolStrip control.

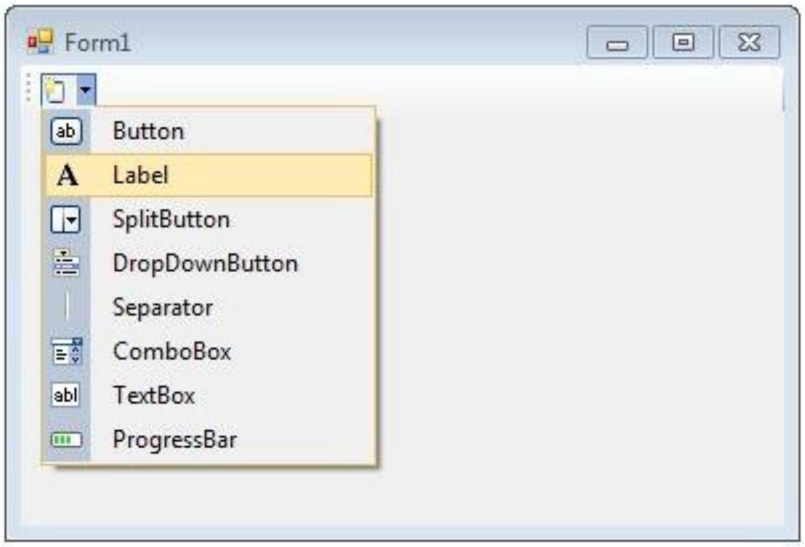
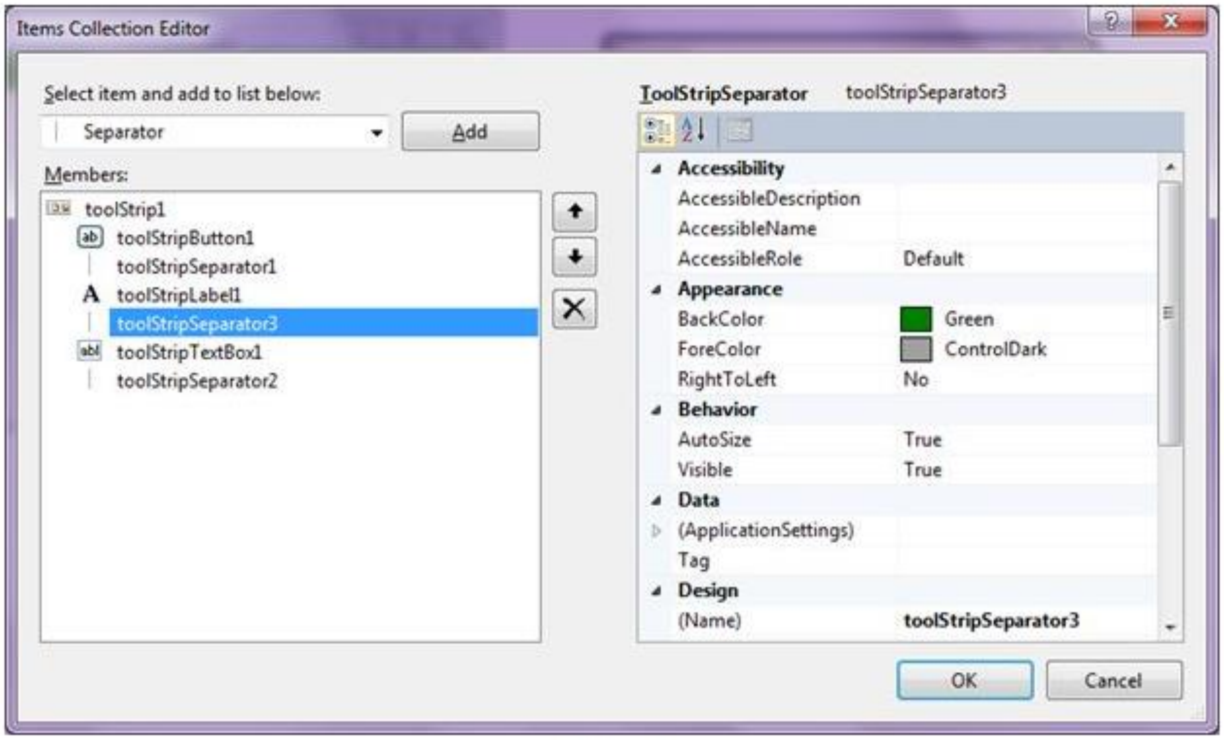


Figure 3

Another way to add controls to a ToolStrip control is by using the Items property.

Click on the Items property and you will see Item Collection Editor that looks like Figure 4. We are going to add a Button, a TextBox, and a Label control to ToolStrip with a few separators.


Items Collection Editor

Select item and add to list below: Separator [Add]

Members:

```
toolStrip1
    toolStripButton1
    toolStripSeparator1
    toolStripLabel1
    toolStripSeparator3
    toolStripTextBox1
    toolStripSeparator2
```

ToolStripSeparator toolStripSeparator3

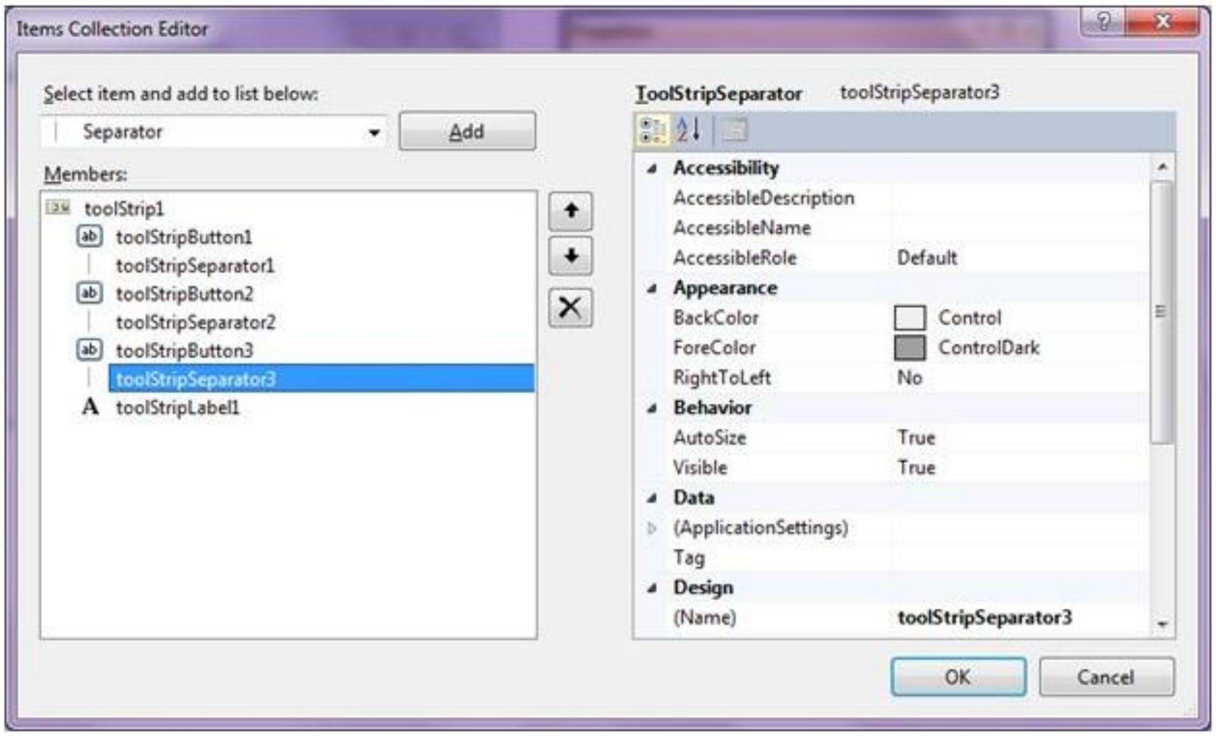
Accessibility	
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
Appearance	
BackColor	🟩 Green
ForeColor	▨ ControlDark
RightToLeft	No
Behavior	
AutoSize	True
Visible	True
Data	
(ApplicationSettings)	
Tag	
Design	
(Name)	**toolStripSeparator3**

Figure 4

If you need to reposition controls, you can simply use up and down arrows.

Here is the good part. Once these controls are added to a ToolStrip control, they all act as individual controls and you can access them by using their Name property anywhere in the code.

Now let's create a sample. We are going to add three Button controls, three separators and a Label control as you can see in Figure 5.



toolStrip1
toolStripButton1
toolStripSeparator1
toolStripButton2
toolStripSeparator2
toolStripButton3
toolStripSeparator3
toolStripLabel1

ToolStripSeparator toolStripSeparator3

Accessibility	
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
Appearance	
BackColor	Control
ForeColor	ControlDark
RightToLeft	No
Behavior	
AutoSize	True
Visible	True
Data	
(ApplicationSettings)	
Tag	
Design	
(Name)	**toolStripSeparator3**

Figure 5

StatusStrip:

StatusStrip Control

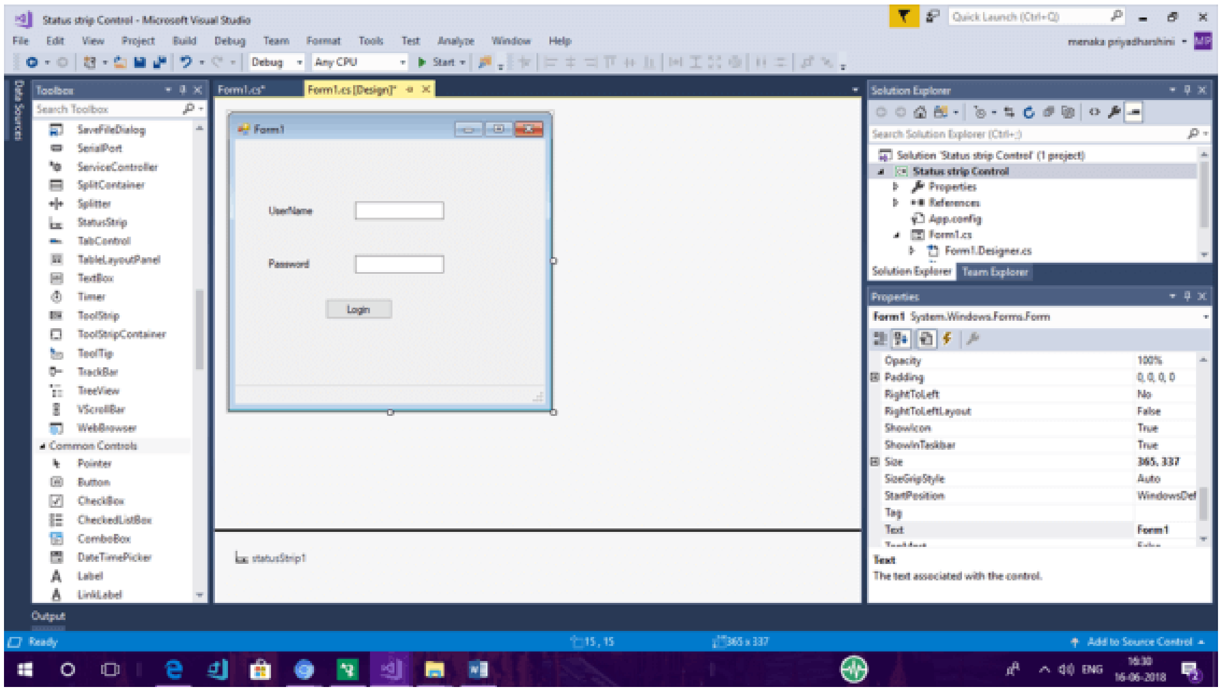
StatusStrip control displays windows status. It is usually at the bottom of a window. We use a ToolStrip hyperlink in the C# windows forms Status Bar.

ToolStrip

StatusStrip replaces and extends the Status Bar control of the previous version. StatusStrip control displays information about an object being viewed on a form, the object components or contextual information that relates to that object's operation within the application.

Typically, a StatusStrip consists of ToolStrip objects, but by default, StatusStrip has no panels. ToolStrip control is capable of hosting Button, Label, SplitButton, DropDownButton, Separator, ComboBox, TextBox, and ProgressBar controls.

Now, let's add another control to the form by dragging a n control from the Toolbox to the form. You may also want to change the properties of the other controls.



Form1 System.Windows.Forms.Form

Opacity	100%
Padding	0, 0, 0, 0
RightToLeft	No
RightToLeftLayout	False
ShowIcon	True
ShowInTaskbar	True
Size	365, 337
SizeGripStyle	Auto
StartPosition	WindowsDef
Tag	
Text	Form1

Text
The text associated with the control.

Docking Controls:

If you're designing a form that the user can resize at run time, the controls on your form should resize and reposition properly. Controls have two properties that help with automatic placement and sizing, when the form changes size.

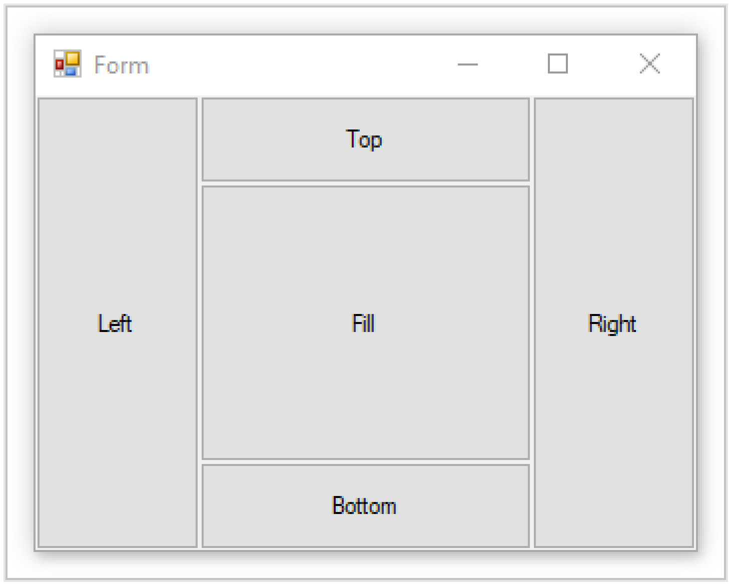
What is a docking control?

Docking refers to **how much space you want the control to take up on the form**. If you dock a control to the left of the form, it will stretch itself to the height of the form, but its width will stay the same.

We can dock controls to the edges of your form or have them fill the controls container (either a form or a container control). For example, Windows Explorer docks its TreeView control to the left side of the window and its ListView control to the right side of the window.

Control.Dock

Controls that are docked fill the edges of the control's container, either the form or a container control. For example, Windows Explorer docks its TreeView control to the left side of the window and its ListView control to the right side of the window. The docking mode can be any side of the control's container, or set to fill the remaining space of the container.



Controls are docked in reverse z-order and the Dock property interacts with the AutoSize property. For more information, see Automatic sizing.

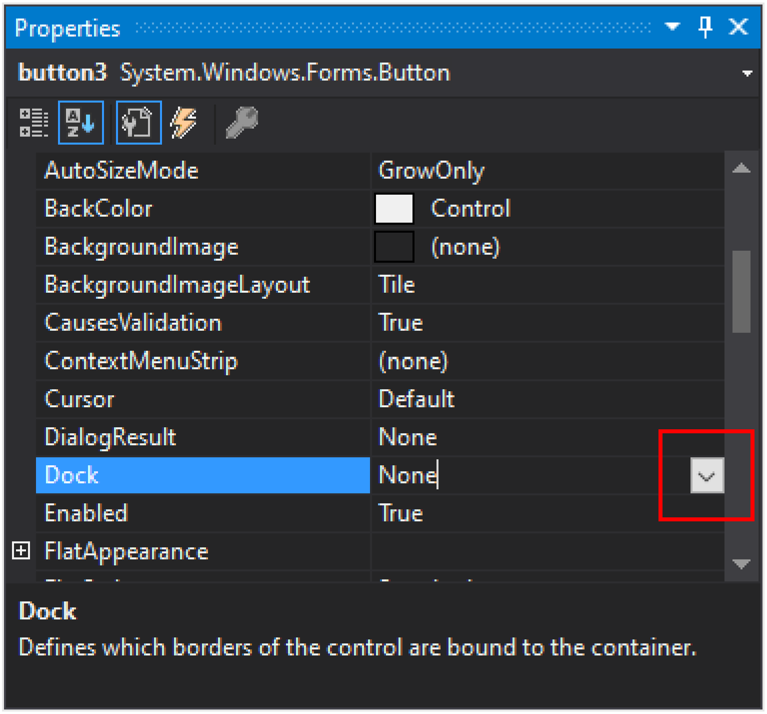
Dock a control

A control is docked by setting its Dock property.

Use the designer

Use the Visual Studio designer **Properties** window to set the docking mode of a control.

1. Select the control in the designer.
2. In the **Properties** window, select the arrow to the right of the **Dock** property.



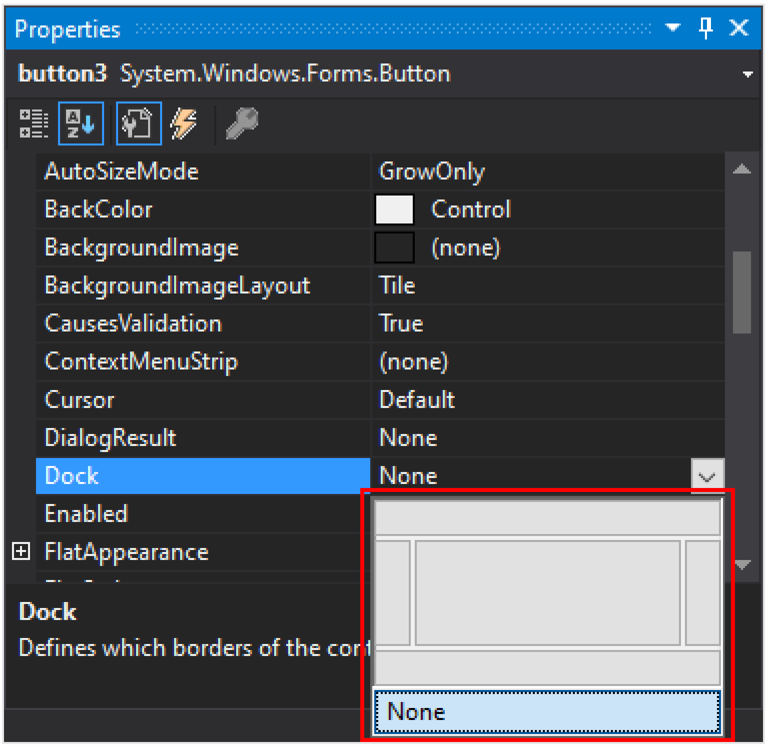
Properties

button3 System.Windows.Forms.Button

AutoSizeMode	GrowOnly
BackColor	Control
BackgroundImage	(none)
BackgroundImageLayout	Tile
CausesValidation	True
ContextMenuStrip	(none)
Cursor	Default
DialogResult	None
Dock	None
Enabled	True
⊞ FlatAppearance	

Dock
Defines which borders of the control are bound to the container.

Select the button that represents the edge of the container where you want to dock the control. To fill the contents of the control's form or container control, press the center box. Press **(none)** to disable docking.



Properties

button3 System.Windows.Forms.Button

AutoSizeMode	GrowOnly
BackColor	Control
BackgroundImage	(none)
BackgroundImageLayout	Tile
CausesValidation	True
ContextMenuStrip	(none)
Cursor	Default
DialogResult	None
Dock	None
Enabled	
⊞ FlatAppearance	

Dock
Defines which borders of the con

The control is automatically resized to fit the boundaries of the docked edge.

Set Dock programmatically

1. Set the `Dock` property on a control. In this example, a button is docked to the right side of its container:

   button1.Dock = DockStyle.Right;

   MessageBox:

MessageBox is a class in C# and Show is a method that displays a message in a small window in the center of the Form.

MessageBox is used to provide confirmations of a task being done or to provide warnings before a task is done.

Create a Windows Forms app in Visual Studio and add a button on it. Something like this below.

Example:

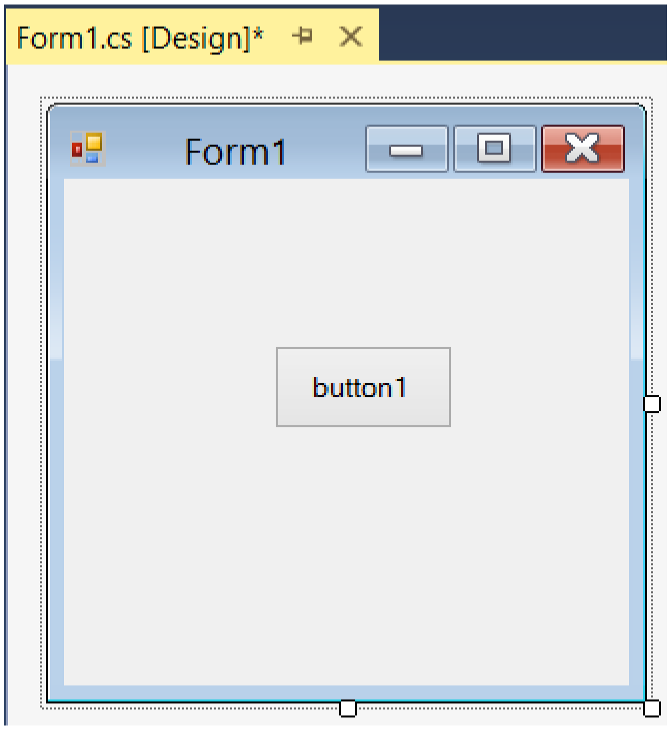


Figure 1 Windows Form

Let's say, you want to show a message on a button click event handler. Here is the code for that.

Unit: III
Chapter 5: Basic Controls

Button:

C# Button class in .NET Framework class library represents a Windows Forms Button control. A Button control is a child control placed on a Form and used to process click event and can be clicked by a mouse click or by pressing ENTER or ESC keys.

Creating a C# Button

To create a Button control, you simply drag and drop a Button control from Toolbox to Form in Visual Studio. After you drag and drop a Button to a Form, the Button looks like Figure 1. Once a Button is on the Form, you can move it around and resize it.

A Button is an essential part of an application, or software, or webpage. It allows the user to interact with the application or software. For example, if a user wants to exit from the current application so, he/she click the exit button which closes the application. It can be used to perform many actions like to submit, upload, download, etc. according to the requirement of your program. It can be available with different shape, size, color, etc. and you can reuse them in different applications. In .NET Framework, Button class is used to represent windows button control and it is inherited from *ButtonBase class*. It is defined under *System.Windows.Forms* namespace.
In C# you can create a button on the windows form by using two different ways:
1. Design-Time: It is the easiest method to create a button. Use the below steps:

- **Step 1:** Create a windows form as shown in the below image:
  Visual Studio -> File -> New -> Project -> WindowsFormApp

2. Run-Time: It is a little bit trickier than the above method. In this method, you can create your own Button using the Button class.

- **Step 1:** Create a button using the Button() constructor is provided by the Button class.

```
// Creating Button using Button class

Button MyButton = new Button();
```

- **Step 2:** After creating Button, set the properties of the Button provided by the Button class.:

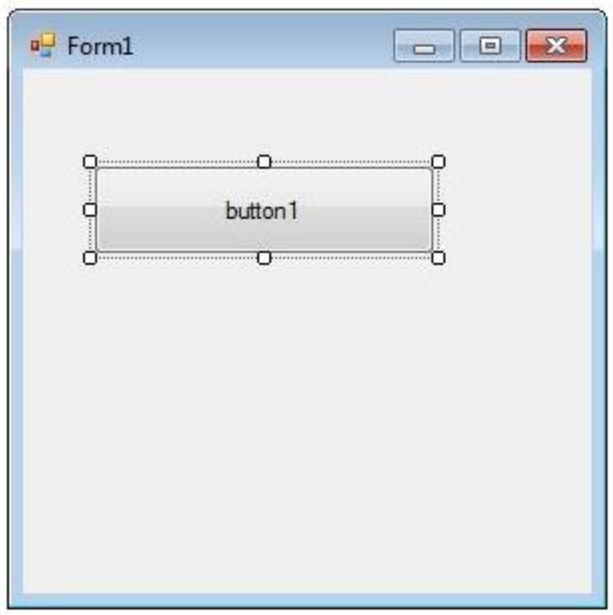
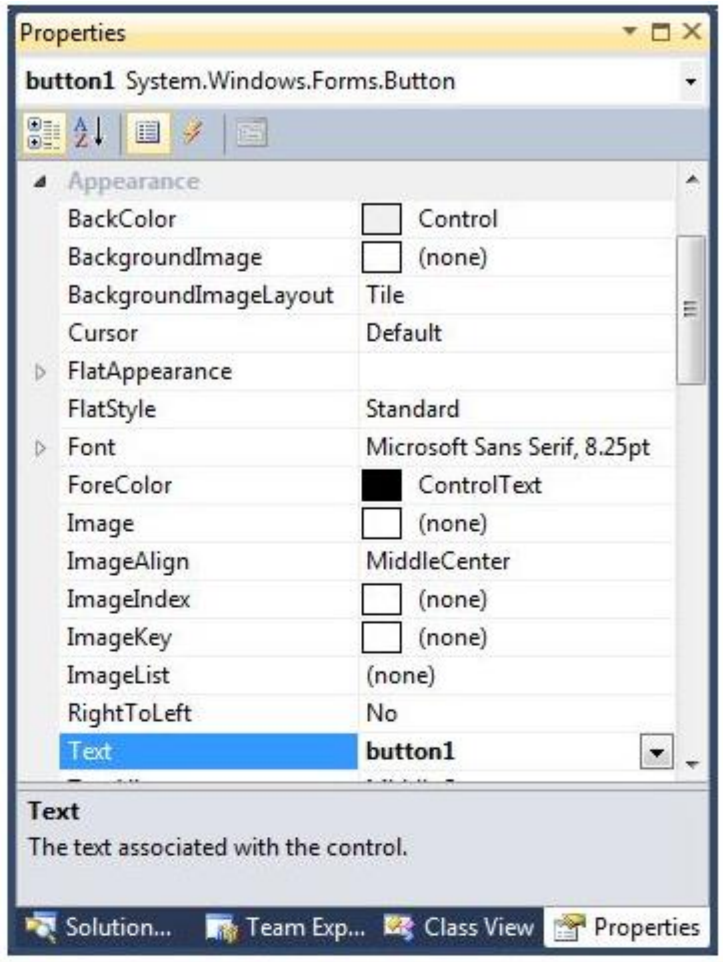


Figure 1

Setting Button Properties

After you place a Button control on a Form, the next step is to set button properties.

The easiest way to set a Button control properties is by using the Properties Window. You can open Properties window by pressing F4 or right click on a control and select Properties menu item. The Properties window looks like Figure 2



Appearance	
BackColor	Control
BackgroundImage	(none)
BackgroundImageLayout	Tile
Cursor	Default
FlatAppearance	
FlatStyle	Standard
Font	Microsoft Sans Serif, 8.25pt
ForeColor	ControlText
Image	(none)
ImageAlign	MiddleCenter
ImageIndex	(none)
ImageKey	(none)
ImageList	(none)
RightToLeft	No
Text	**button1**

Text
The text associated with the control.

TextBox:

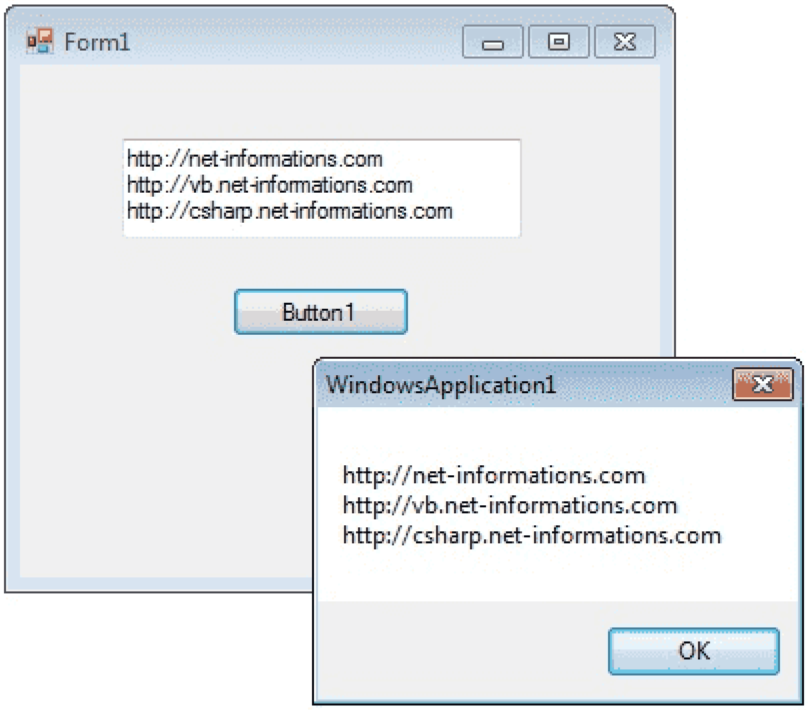
C# TextBox Control

A TextBox control is used to display, or accept as input, a single line of text. This control has additional functionality that is not found in the standard Windows text box control, including multiline editing and password character masking.

In Windows forms, TextBox plays an important role. With the help of TextBox, the user can enter data in the application, it can be of a single line or of multiple lines. The TextBox is a class and it is defined under *System.Windows.Forms* namespace. In C#, you can create a TextBox in two different ways:

1. Design-Time: It is the simplest way to create a TextBox as shown in the following steps:

- **Step 1:** Create a windows form. As shown in the below image:
  Visual Studio -> File -> New -> Project -> WindowsFormApp



http://net-informations.com
http://vb.net-informations.com
http://csharp.net-informations.com

http://net-informations.com
http://vb.net-informations.com
http://csharp.net-informations.com

A text box object is used to display text on a form or to get user input while a C# program is running. In a text box, a user can type data or paste it into the control from the clipboard.

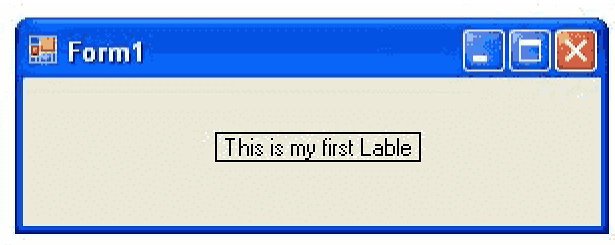
For displaying a text in a TextBox control , you can code like this.

```
textBox1.Text = " https://www.shahucollegelatur.org.in/ ";
```

Label:

C# Label Control

Labels are one of the most frequently used C# control. We can use the Label control to display text in a set location on the page. Label controls can also be used to add descriptive text to a Form to provide the user with helpful information. The Label class is defined in the System.Windows.Forms namespace.



Add a Label control to the form - Click Label in the Toolbox and drag it over the forms Designer and drop it in the desired location.

If you want to change the display text of the Label, you have to set a new text to the Text property of Label.

label1.Text = "This is my first Label";

In addition to displaying text, the Label control can also display an image using the Image property, or a combination of the ImageIndex and ImageList properties.

In Windows Forms, Label control is used to display text on the form and it does not take part in user input or in mouse or keyboard events. The Label is a class and it is defined under System.Windows.Forms namespace. In windows form, you can create Label in two different ways:

1. Design-Time: It is the easiest method to create a Label control using the following steps:

- **Step 1:** Create a windows form as shown in the below image:
  Visual Studio -> File -> New -> Project -> WindowsFormApp

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. LinkLabel control does not participate in user input or capture mouse or keyboard events.

Creating a LinkLabel

There are two ways to create a control.

Design-time

First, we can use the Form designer of Visual Studio to create a control at design-time. In design-time mode, we can use visual user interfaces to create a control properties and write methods.

To create a LinkLabel control at design-time, you simply drag and drop a LinkLabel control from Toolbox to a Form. After you drag and drop a LinkLabel on a Form. The LinkLabel looks like Figure 1. Once a LinkLabel is on the Form, you can move it around and resize it using mouse and set its properties and events.

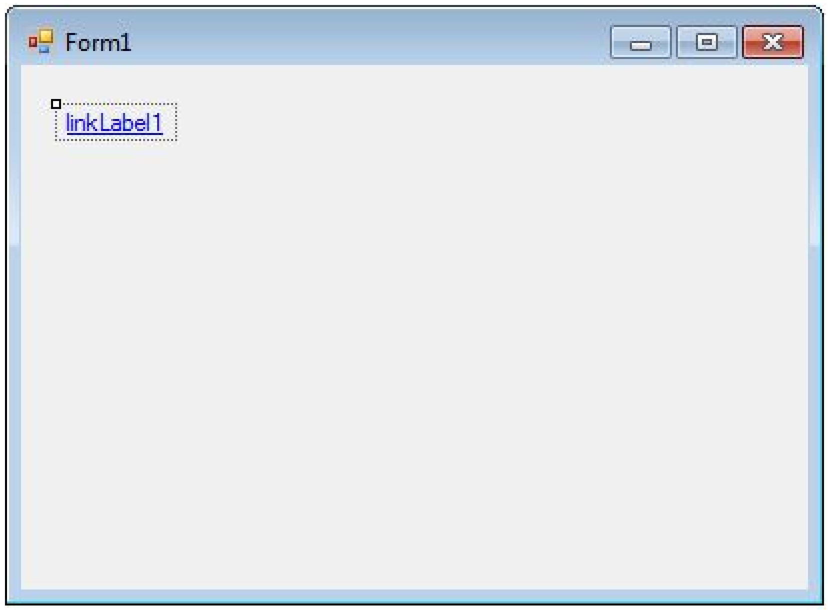


Figure 1

Run-time

LinkLabel class represents a hyperlink Label control. We simply create an instance of LinkLabel class, set its properties and add this it to the Form controls.

In the first step, we create an instance of the LinkLabel class. The following code snippet creates a LinkLabel control object.

LinkLabel dynamicLinkLabel = newLinkLabel();

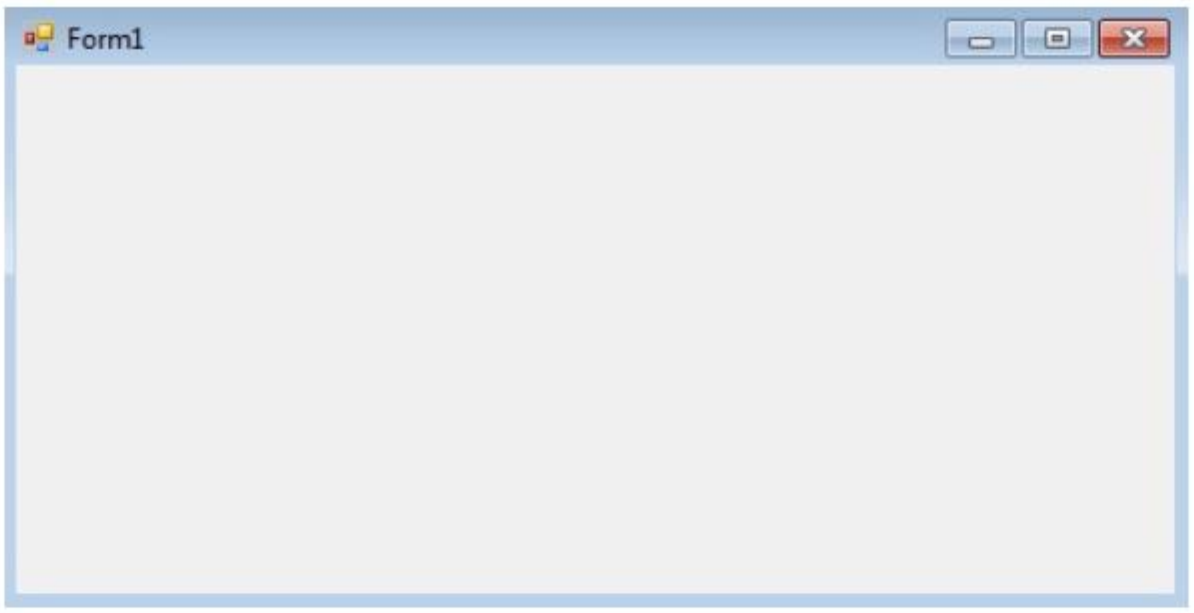
In the next step, we set properties of a LinkLabel control. The following code snippet sets background color, foreground color, Text, Name, and Font properties of a LinkLabel.

RadioButton:

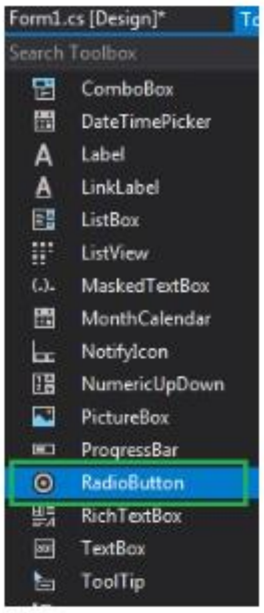
RadioButton in C#

In Windows Forms, RadioButton control is used to select a single option among the group of the options. For example, select your gender from the given list, so you will choose only one option among three options like Male or Female or Transgender. In C#, RadioButton is a class and it is defined under *System.Windows.Forms* namespace. In RadioButton, you are allowed to display text, image, or both and when you select one radio button in a group other radio buttons automatically clear. You can create RadioButton in two different ways:
1. Design-Time: It is the easiest way to create a RadioButton control as shown in the following steps:

- **Step 1:** Create a windows form as shown in the below image:
  Visual Studio -> File -> New -> Project -> WindowsFormApp



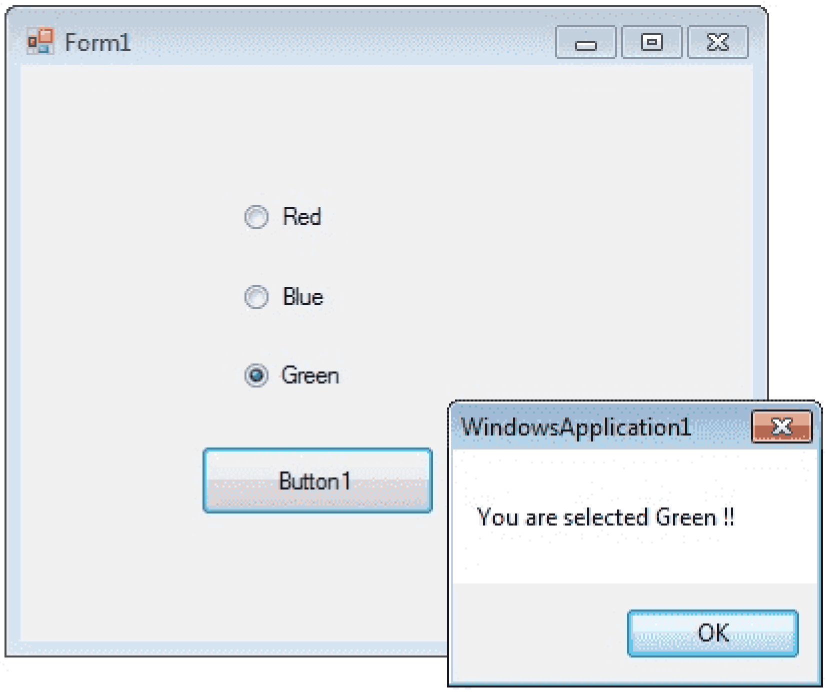
- **Step 2:** Drag the RadioButton control from the ToolBox and drop it on the windows form. You are allowed to place a RadioButton control anywhere on the windows form according to your need.



- **Step 3:** After drag and drop you will go to the properties of the RadioButton control to modify RadioButton control according to your requirement.

C# RadioButton Control

A radio button or option button enables the user to select a single option from a group of choices when paired with other RadioButton controls. When a user clicks on a radio button, it becomes checked, and all other radio buttons with same group become unchecked.



You are selected Green !!

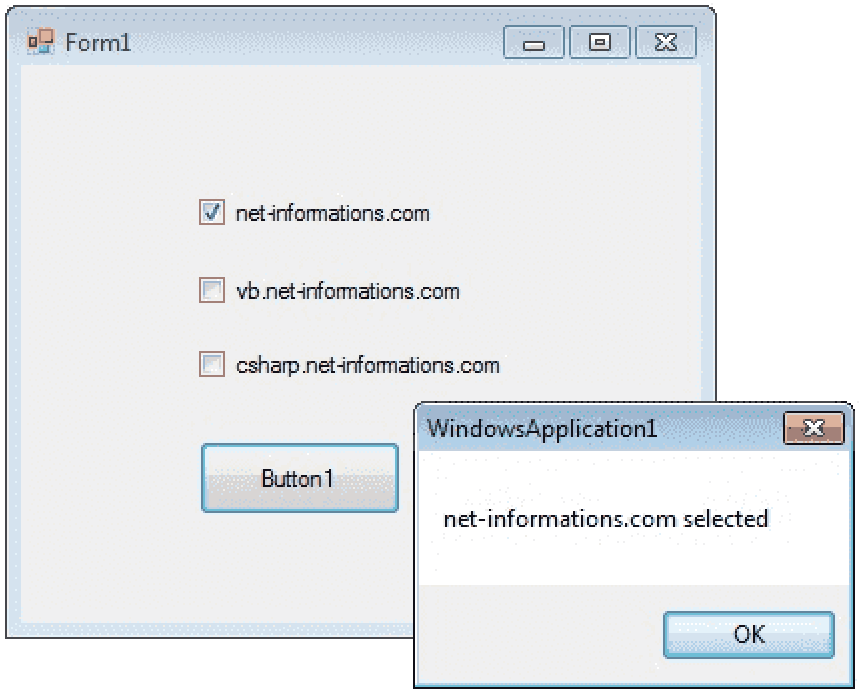
The RadioButton control can display text, an Image, or both. Use the Checked property to get or set the state of a RadioButton.

The radio button and the check box are used for different functions. Use a radio button when you want the user to choose only one option. When you want the user to choose all appropriate options, use a check box. Like check boxes, radio buttons support a Checked property that indicates whether the radio button is selected.

CheckBox:

C# CheckBox Control

CheckBoxes allow the user to make multiple selections from a number of options. CheckBox to give the user an option, such as true/false or yes/no. You can click a check box to select it and click it again to deselect it.



net-informations.com selected

The CheckBox control can display an image or text or both. Usually CheckBox comes with a caption, which you can set in the Text property.

You can use the CheckBox control ThreeState property to direct the control to return the Checked, Unchecked, and Indeterminate values. You need to set the check boxs ThreeState property to True to indicate that you want it to support three states.

The radio button and the check box are used for different functions. Use a radio button when you want the user to choose only one option.When you want the user to choose all appropriate options, use a check box. The following C# program shows how to find a checkbox is selected or not.

Creating a CheckBox

We can create a CheckBox control using a Forms Designer at design-time or use the CheckBox class in code at run-time (also known as dynamically).

To create a CheckBox control at design-time, you simply drag and drop a CheckBox control from Toolbox to a Form in Visual Studio. After you drag and drop a CheckBox on a Form, the CheckBox looks like Figure 1. Once a CheckBox is on the Form, you can move it around and resize it using a mouse and set its properties and events.

The CheckBox control is the part of windows form which is used to take input from the user. Or in other words, CheckBox control allows us to select single or multiple elements from the given list or it can provide us options like yes or no, true or false, etc. It can be displayed as an image or text or both. The CheckBox is a class and defined under *System.Windows.Forms* namespace. In Windows form, you can create CheckBox in two different ways:
1. Design-Time: It is the simplest way to create a CheckBox using the following steps:

- **Step 1:** Create a windows form as shown in the below image:
  Visual Studio -> File -> New -> Project -> WindowsFormApp

C# | DateTimePicker Class:

In Windows Forms, the DateTimePicker control is used to select and display the date/time with a specific format in your form. The FlowLayoutPanel class is used to represent windows DateTimePicker control and also provide different types of properties, methods, and events. It is defined under **System.Windows.Forms** namespace. You can create two different types of DateTimePicker, as a drop-down list with a date represented in the text, or as a calendar which appears when you click on the down-arrow next to the given list. In C#, you can create a DateTimePicker in the windows form by using two different ways:
1. Design-Time: It is the easiest way to create a DateTimePicker control as shown in the following steps:

- **Step 1:** Create a windows form as shown in the below image:
  Visual Studio -> File -> New -> Project -> WindowsFormApp

A DateTimePicker control allows users to select a date and time in Windows Forms applications. In this tutorial, we will see how to create a DateTimePicker control at design-time as well as at run-time, set its properties and call its methods.

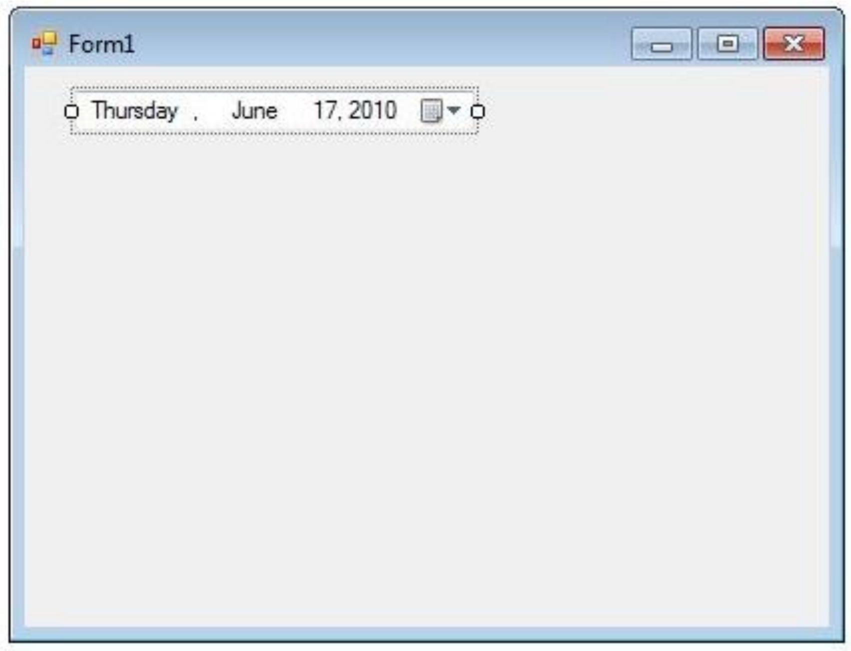


Figure 1.

Creating a DateTimePicker

We can create a DateTimePicker control using the Forms designer at design-time or using the DateTimePicker class in code at run-time (also known as dynamically).

Design-time

To create a DateTimePicker control at design-time, you simply drag and drop a DateTimePicker control from Toolbox to a Form in Visual Studio. After you drag and drop a DateTimePicker on a Form, the DateTimePicker looks like Figure 1. Once a DateTimePicker is on the Form, you can move it around and resize it using mouse and set its properties and events.

Now if you run the application and click on the calendar icon in the control, you will see a monthly calendar pops us for the current month as shown in Figure 2.

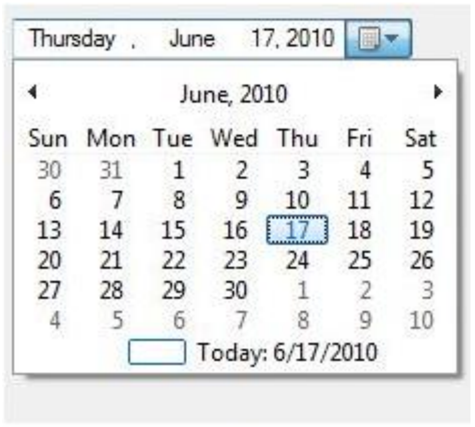
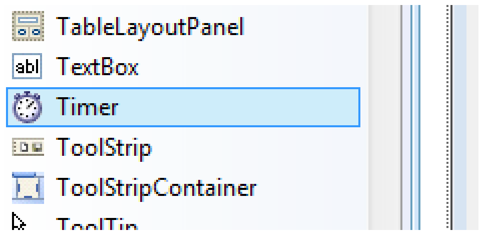


Figure 2

C# Timer Control

What is C# Timer Control ?

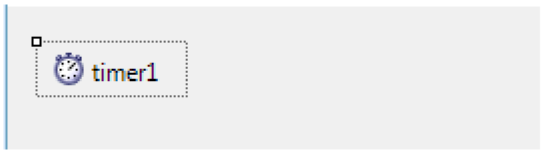
The **Timer Control** plays an important role in the development of programs both Client side and Server side development as well as in Windows Services. With the **Timer Control** we can raise events at a specific interval of time without the interaction of another thread.



Use of Timer Control

We require **Timer Object** in many situations on our development environment. We have to use Timer Object when we want to set an interval between events, periodic checking, to start a process at a fixed time schedule, to increase or decrease the speed in an animation graphics with **time schedule** etc.

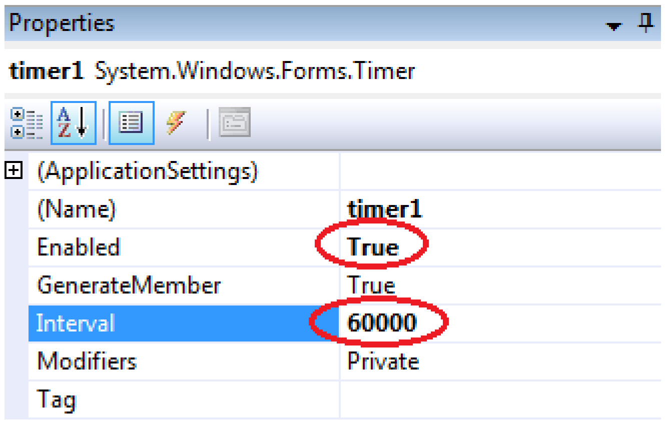
A **Timer control** does not have a visual representation and works as a component in the background.



How to use C# Timer Control ?

We can control programs with Timer Control in **millisecond** , seconds, minutes and even in hours. The Timer Control allows us to set **Interval property** in milliseconds. That is, one second is equal to 1000 milliseconds. For example, if we want to set an interval of 1 minute we set the value at Interval property as 60000, means 60x1000 .

By default the **Enabled property** of Timer Control is False. So before running the program we have to set the Enabled property is True , then only the Timer Control starts its function.



timer1 System.Windows.Forms.Timer

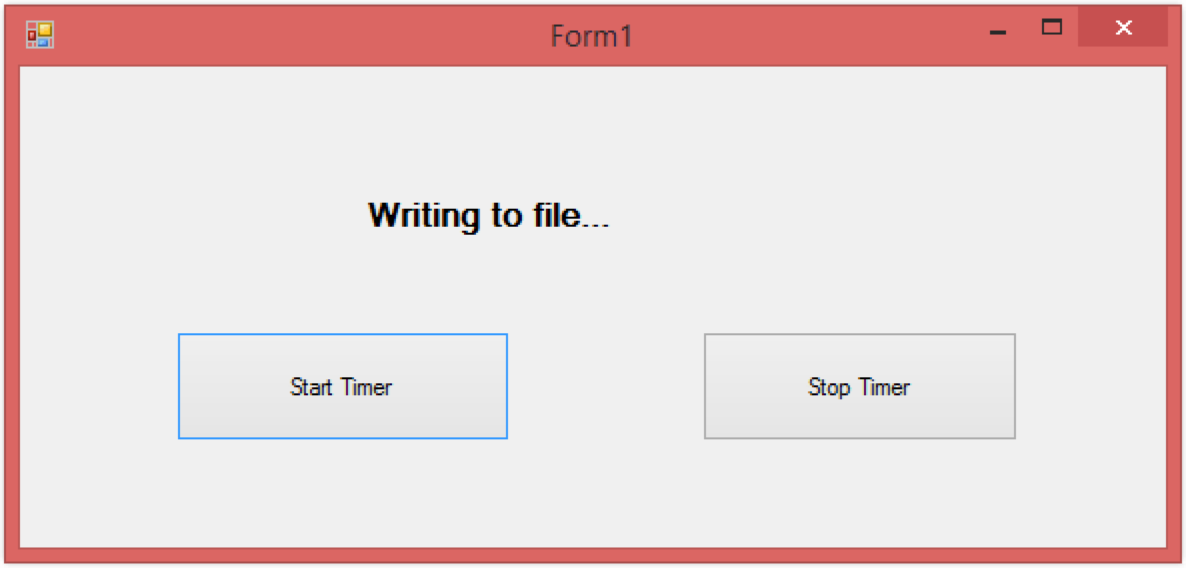
(ApplicationSettings)	
(Name)	timer1
Enabled	True
GenerateMember	True
Interval	60000
Modifiers	Private
Tag	

C# Timer example

In the following program we display the **current time** in a Label Control. In order to develop this program, we need a **Timer Control** and a Label Control. Here we set the timer interval as 1000 milliseconds, that means one second, for displaying current system time in Label control for the interval of one second.

Start and Stop Timer Control

The Timer control have included the Start and Stop methods for start and stop the Timer control functions. The following **C# program** shows how to use a timer to write some text to a text file each seconds. The program has two buttons, Start and Stop. The application will write a line to a text file every 1 second once the **Start button** is clicked. The application stops writing to the text file once the **Stop button** is clicked.



```
using System;

using System.Windows.Forms;

using System.IO;

namespace WindowsFormsApplication1

{

   public partial class Form1 : Form

   {

      public Form1()
```

```
        {
            InitializeComponent();
        }
        TextWriter tsw;
        private void Form1_Load(object sender, EventArgs e)
        {
            tsw = new StreamWriter(@"D:\timer.txt", true);
        }
        private void timer1_Tick(object sender, EventArgs e)
        {
            label1.Text = DateTime.Now.ToString();
            tsw.WriteLine(DateTime.Now.ToString());
        }
        private void button1_Click(object sender, EventArgs e)
        {
            timer1.Interval = 1000;
            timer1.Start();
        }
        private void button2_Click(object sender, EventArgs e)
        {
            timer1.Stop();
            tsw.Close();
        }
    }
}
```

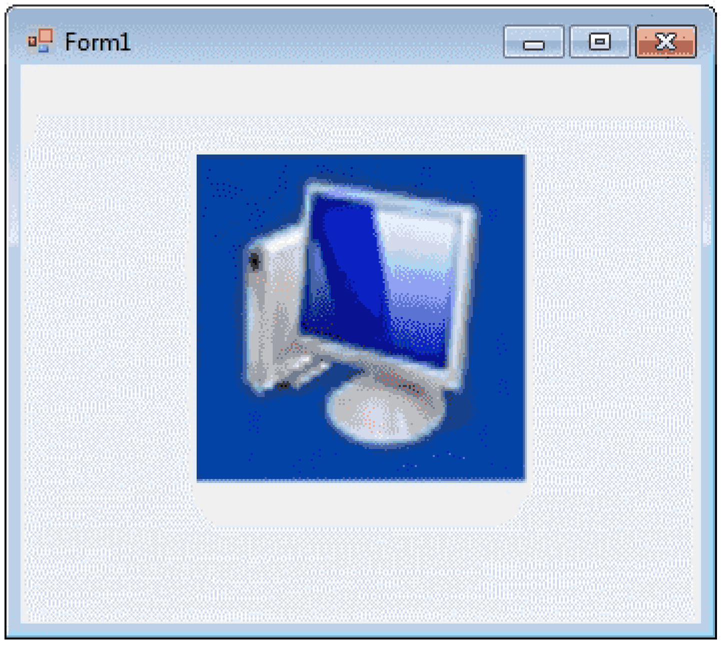
Timer Tick Event

Timer **Tick event** occurs when the specified timer interval has elapsed and the timer is enabled.

PictureBox:

C# PictureBox Control

The Windows Forms PictureBox control is used to display images in bitmap, GIF , icon , or JPEG formats.



You can set the Image property to the Image you want to display, either at design time or at run time. You can programmatically change the image displayed in a picture box, which is particularly useful when you use a single form to display different pieces of information.

```
pictureBox1.Image = Image.FromFile("c:\\testImage.jpg");
```

The SizeMode property, which is set to values in the PictureBoxSizeMode enumeration, controls the clipping and positioning of the image in the display area.

pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;

There are five different PictureBoxSizeMode is available to PictureBox control.

- AutoSize - Sizes the picture box to the image.
- CenterImage - Centers the image in the picture box.
- Normal - Places the upper-left corner of the image at upper left in the picture box
- StretchImage - Allows you to stretch the image in code

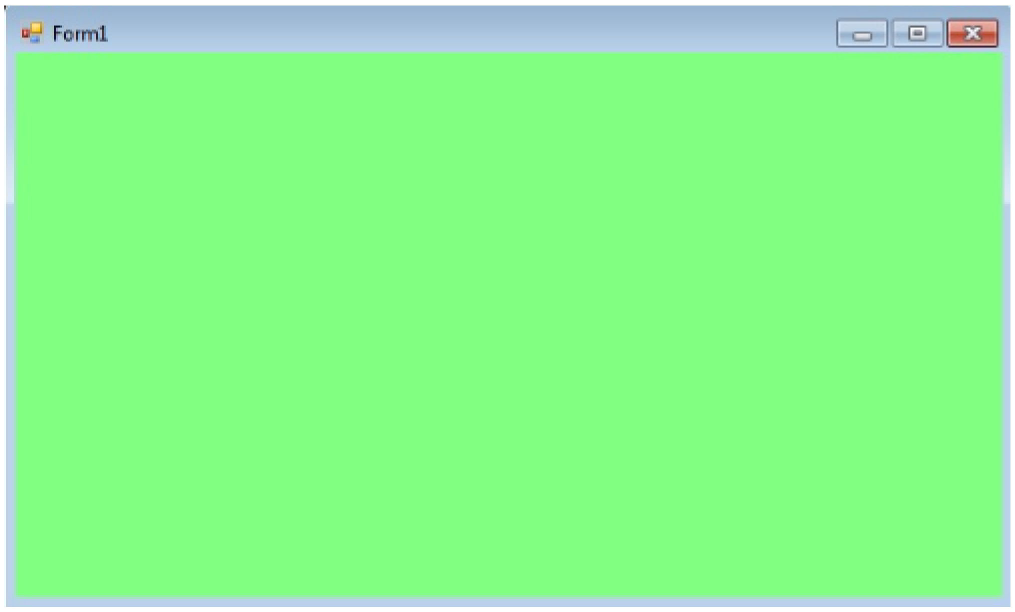
The PictureBox is not a selectable control, which means that it cannot receive input focus. The following C# program shows how to load a picture from a file and display it in streach mode.

ComboBox in C#

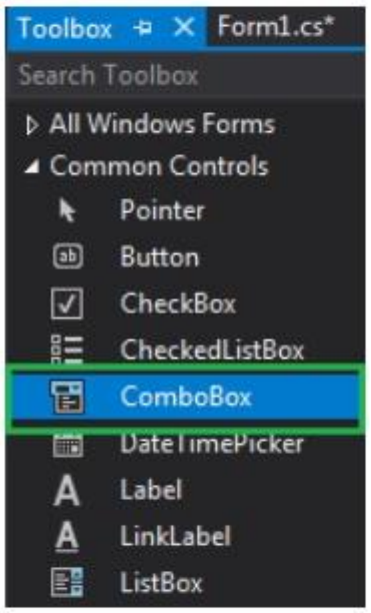
In Windows Forms, ComboBox provides two different features in a single control, it means ComboBox works as both TextBox and ListBox. In ComboBox, only one item is displayed at a time and the rest of the items are present in the drop-down menu. The ComboBox is a class in C# and defined under *System.Windows.Forms* Namespace. You can create ComboBox using the two different ways:

1. Design-Time: It is the easiest method to create a ComboBox control using the following steps:

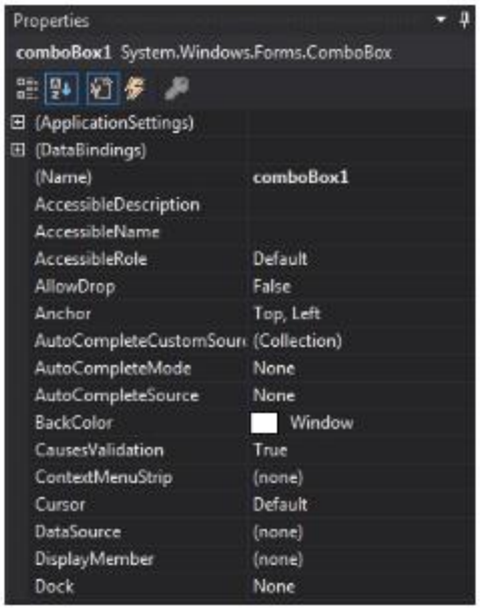
- **Step 1:** Create a windows form as shown in the below image:
  Visual Studio -> File -> New -> Project -> WindowsFormApp



Step 2: Drag the ComboBox control from the ToolBox and drop it on the windows form. You are allowed to place a ComboBox control anywhere on the windows form according to your need.



Step 3: After drag and drop you will go to the properties of the ComboBox control to set the properties of the ComboBox according to your need.

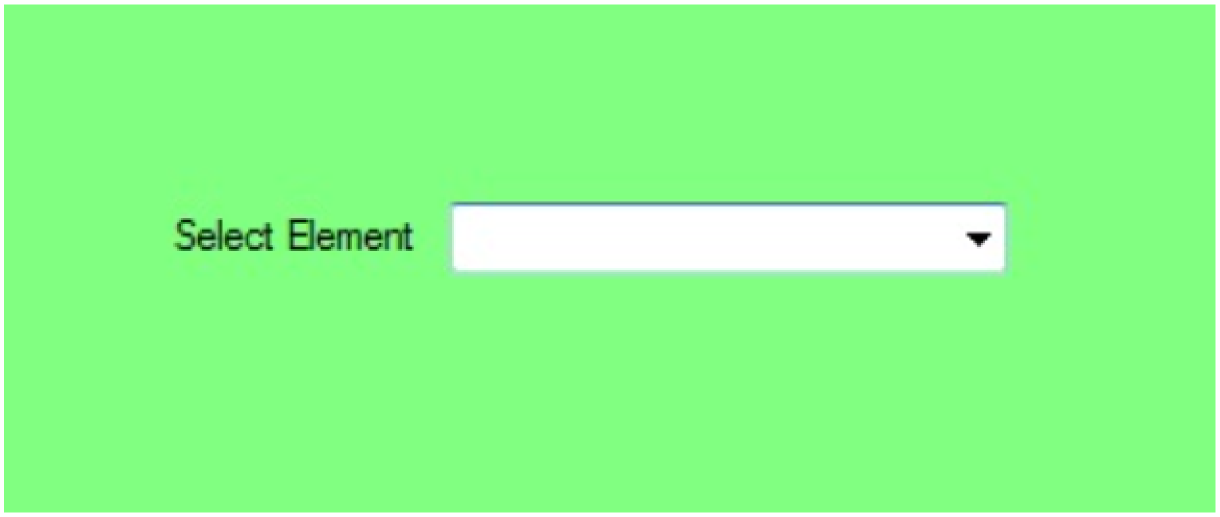


Properties

comboBox1 System.Windows.Forms.ComboBox

⊞ (ApplicationSettings)	
⊞ (DataBindings)	
(Name)	**comboBox1**
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
Anchor	Top, Left
AutoCompleteCustomSour	(Collection)
AutoCompleteMode	None
AutoCompleteSource	None
BackColor	⬜ Window
CausesValidation	True
ContextMenuStrip	(none)
Cursor	Default
DataSource	(none)
DisplayMember	(none)
Dock	None

Output:



C# ComboBox Control

The ComboBox control provides combined functionality of a text box and a listbox in a single control. Only one list item is displayed at one time in a ComboBox and rest of the available items are loaded in a drop down list.

Creating a ComboBox

We can create a ComboBox control using a Forms designer at design-time or using the ComboBox class in C# code at run-time.

To create a ComboBox control at design-time, you simply drag and drop a ComboBox control from Toolbox to a Form in Visual Studio. After you drag and drop a ComboBox on a Form, the ComboBox looks like Figure 1.

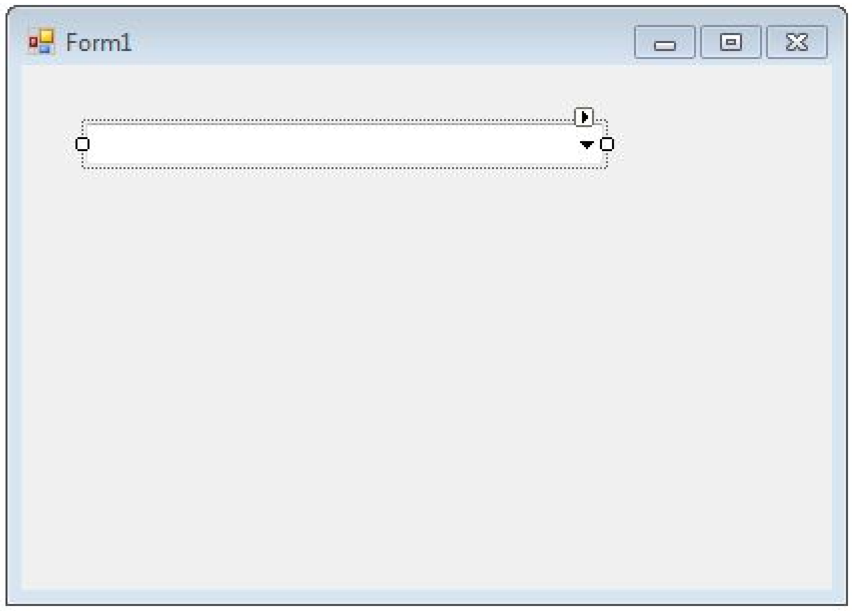


Figure 1

Once a ComboBox is on the Form, you can move it around and resize it and set its properties and events using the Properties and Events windows.

Creating a ComboBox control at run-time includes creating an instance of ComboBox class, set its properties and add ComboBox instance to the Form controls.

First step to create a dynamic ComboBox is to create an instance of ComboBox class.

The following code snippet creates a ComboBox control object.

```
ComboBox comboBox1 = new ComboBox();
```

In the next step, you set properties of a ComboBox control.

The following code snippet sets location, width, height, background color, foreground color, Text, Name, and Font properties of a ComboBox.

```
comboBox1.Location = new System.Drawing.Point(20, 60);
comboBox1.Name = "comboBox1";
comboBox1.Size = new System.Drawing.Size(245, 25);
comboBox1.BackColor = System.Drawing.Color.Orange;
comboBox1.ForeColor = System.Drawing.Color.Black;
```

Once the ComboBox control is ready with its properties, the next step is to add the ComboBox to a Form.

To do so, we use Form.Controls.Add method.

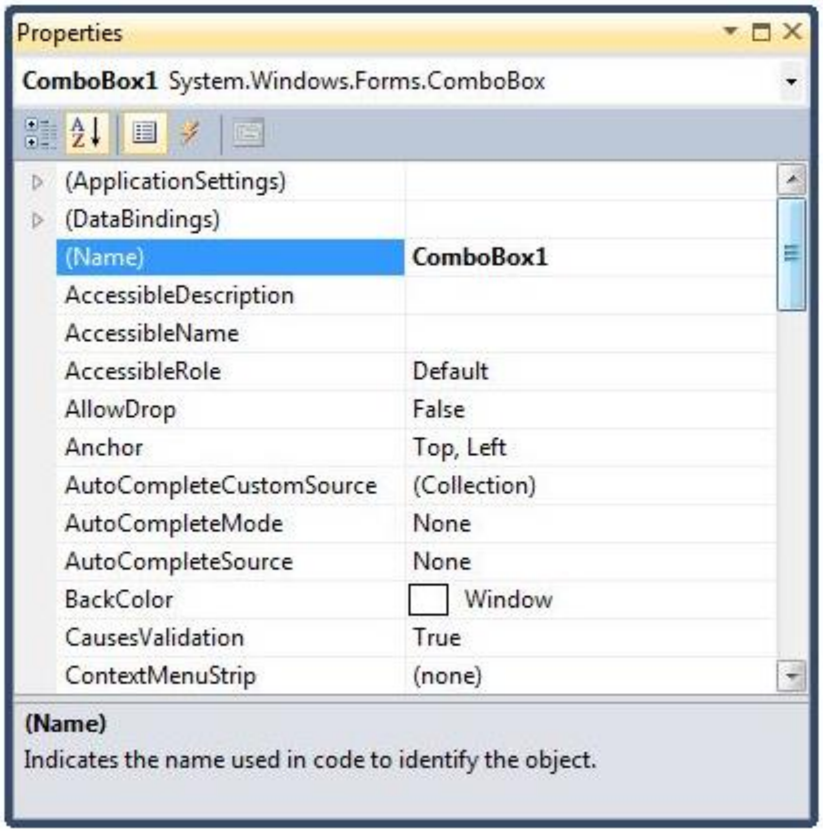
The following code snippet adds a ComboBox control to the current Form.

```
Controls.Add(comboBox1);
```

2. Setting ComboBox Properties

3.

Alternatively, you can set control properites at design time. The easiest way to set properties is from the Properties Window. You can open Properties window by pressing F4 or right click on a control and select Properties menu item. The Properties window looks like Figure 2.



Properties

ComboBox1 System.Windows.Forms.ComboBox

(ApplicationSettings)	
(DataBindings)	
(Name)	**ComboBox1**
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
Anchor	Top, Left
AutoCompleteCustomSource	(Collection)
AutoCompleteMode	None
AutoCompleteSource	None
BackColor	☐ Window
CausesValidation	True
ContextMenuStrip	(none)

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

Figure 2

ListBox Control in C#

The ListBox Control provides us a user interface that will display the List of the items. From there, the users can select one or more items from the List. We can use the ListBox to show the multiple columns, and these columns can contain images and other controls.

ListBox Control Creation in C#

For creating the ListBox, we will follow the two approaches in Windows Form. To create the ListBox control, either we can use the Forms Designer at the time of the designing, or we can use the ListBox class for creating control at the run time.

ListBox Creation at the time of the designing

In our first approach, we will create the ListBox control at the time of the designing using the Windows Forms.

To create the ListBox control, we simply drag the ListBox control from the toolbox and will drop it to the Form. After the drag and drop of Listbox, the Form will look like as below figure. When the ListBox is shown on the Form, now we will resize its size by using the mouse and will set its property and events.

C# | ListBox Class

In Windows Forms, ListBox control is used to show multiple elements in a list, from which a user can select one or more elements and the elements are generally displayed in multiple columns. The ListBox class is used to represent the windows list box and also provide different types of properties, methods, and events. It is defined under *System.Windows.Forms* namespace. The ListBox class contains three different types of collection classes, i.e.

- **ListBox.ObjectCollection:** This class holds all the elements contained in the ListBox control.
- **ListBox.SelectedObjectCollection:** This class holds a collection of the selected items which is a subset of the items contained in the ListBox control.
- **ListBox.SelectedIndexCollection:** This class holds a collection of the selected indexes, which is a subset of the indexes of the *ListBox.ObjectCollection* and these indexes specify elements that are selected.

In C# you can create a ListBox in the windows form by using two different ways:

1. Design-Time: It is the easiest way to create a ListBox as shown in the following steps:

- **Step 1:** Create a windows form as shown in the below image:
  Visual Studio -> File -> New -> Project -> WindowsFormApp

Create a ListBox at Design-time

In our first approach, we are going to create a ListBox control at design-time using the Forms designer.

To create a ListBox control at design-time, we simply drag a ListBox control from the Toolbox and drop it to a Form in Visual Studio. After you drag and drop a ListBox onto a Form, the ListBox looks as in Figure 1. Once a ListBox is on the Form, you can move it around and resize it using the mouse and set its properties and events.

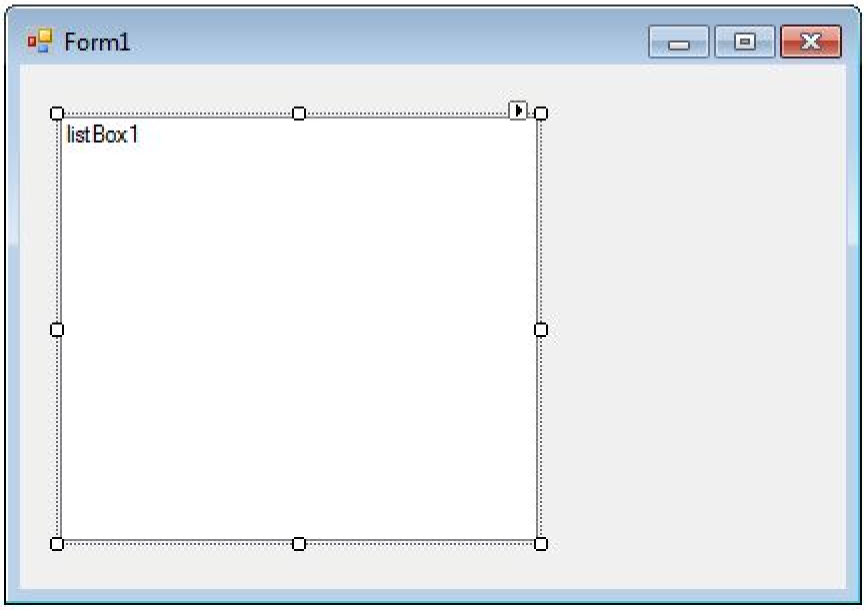


Figure 1

Create a ListBox Dynamically

The ListBox class represents a ListBox control in Windows Forms. To create a ListBox control at run-time, we create an instance of the ListBox class, set its properties and add a ListBox object to the Form controls.

The first step to create a dynamic ListBox is to create an instance of the ListBox class. The following code snippet creates a ListBox control object:

```
ListBox listBox1 = new ListBox();
```

In the next step, you may set the properties of a ListBox control. The following code snippet sets the location, width, height, background color, foreground color, Text, Name, and Font properties of a ListBox:

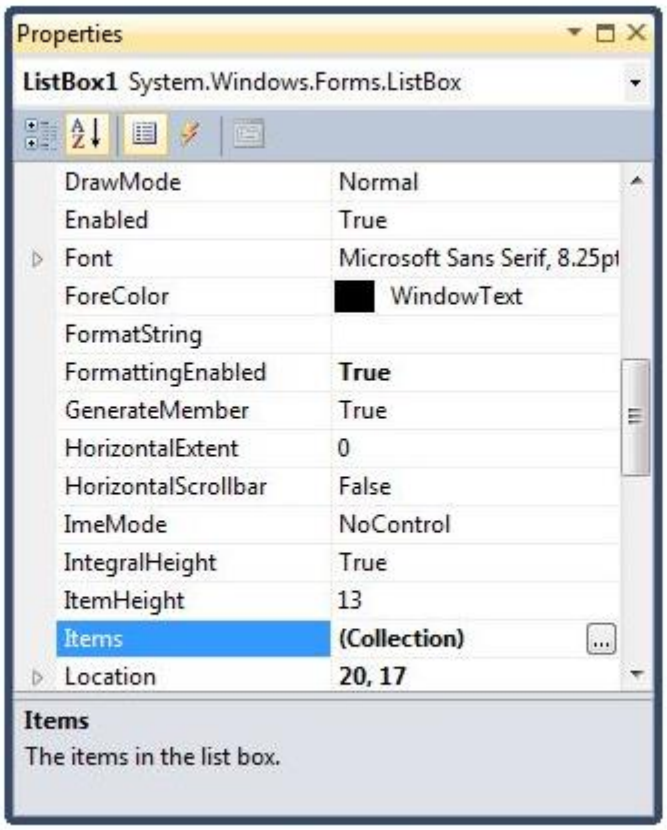
```
listBox1.Location = new System.Drawing.Point(12, 12);
listBox1.Name = "ListBox1";
listBox1.Size = new System.Drawing.Size(245, 200);
listBox1.BackColor = System.Drawing.Color.Orange;
listBox1.ForeColor = System.Drawing.Color.Black;
```

Once the ListBox control is ready with its properties, the next step is to add the ListBox to a Form. To do so, we use the Form.Controls.Add method that adds a ListBox control to the Form controls and displays it on the Form based on the location and size of the control. The following code snippet adds a ListBox control to the current Form:

```
Controls.Add(listBox1);
```

C# ListBox Properties

The easiest way to set properties is from the Properties Window. You can open the Properties window by pressing F4 or by right-clicking on a control and selecting the "Properties" menu item. The Properties window looks as in Figure 2.



DrawMode	Normal
Enabled	True
Font	Microsoft Sans Serif, 8.25pt
ForeColor	■ WindowText
FormatString	
FormattingEnabled	**True**
GenerateMember	True
HorizontalExtent	0
HorizontalScrollbar	False
ImeMode	NoControl
IntegralHeight	True
ItemHeight	13
Items	**(Collection)**
Location	**20, 17**

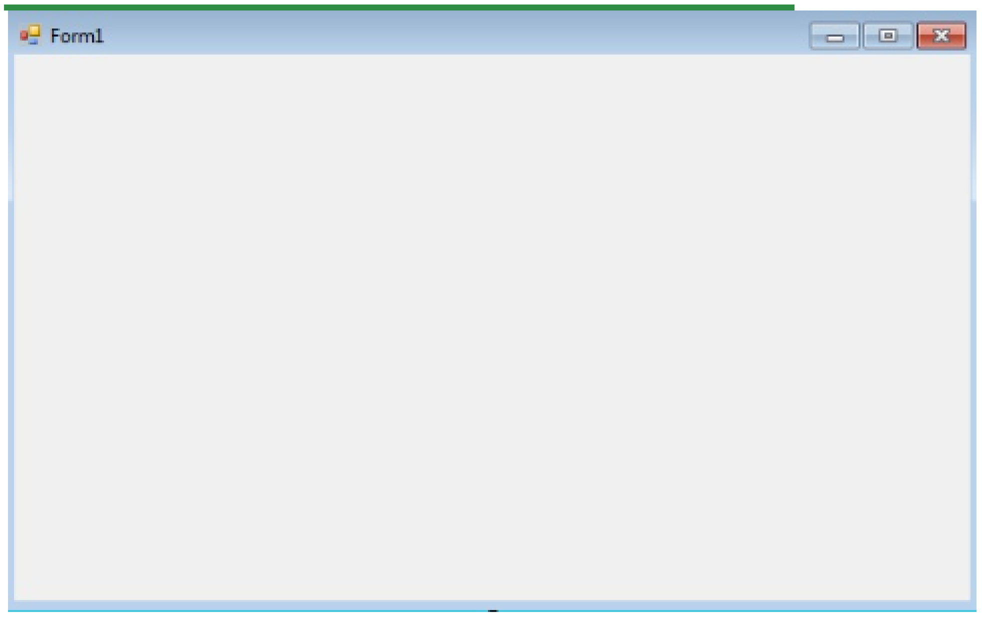
Items
The items in the list box.

Figure 2

C# | RichTextBox Class

In C#, RichTextBox control is a textbox which gives you rich text editing controls and advanced formatting features also includes a loading rich text format (RTF) files. Or in other words, RichTextBox controls allows you to display or edit flow content, including paragraphs, images, tables, etc. The RichTextBox class is used to represent the windows rich text box and also provide different types of properties, methods, and events. It is defined under **System.Windows.Forms** namespace.
It does not have the same 64K character capacity limit like TextBox control. It is used to provide text manipulation and display features similar to word processing applications like Microsoft Word. In C# you can create a RichTextBox in the windows form by using two different ways:
1. Design-Time: It is the easiest way to create a RichTextBox as shown in the following steps:

- **Step 1:** Create a windows form as shown in the below image:
  Visual Studio -> File -> New -> Project -> WindowsFormApp



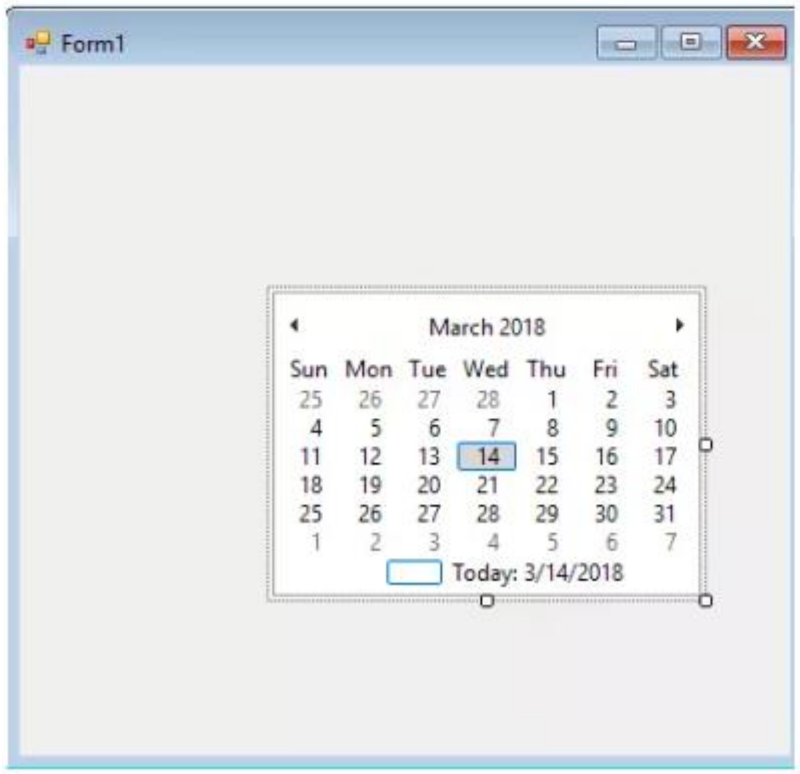
Step 2: Next, drag and drop the RichTextBox control from the toolbox to the form.

MonthCalendar Control (Windows Forms)

The Windows Forms `MonthCalendar` control presents a graphical interface for users to view and set date information. The control displays a grid containing the numbered days of the month, arranged in columns underneath the days of the week. You can select a different month by clicking the arrow buttons on either side of the month caption. Unlike the similar DateTimePicker control, you can select a range of dates with this control; however, the DateTimePicker control allows you to set times as well as dates.

- **C# MonthCalendar Control** is known as graphical interface that is used to modify and select range of date information for required application.
- In order to create C# MonthCalendar control, open the windows form application and go to Toolbar appearing on the left side.
- Find the MonthCalendar control, drag and drop it over the Form.

- You can play with it and move it around over the Form with the help of mouse.
- There is another way of creating the MonthCalendar control. Just double click on the MonthCalendar control, it will automatically place the MonthCalendar control on the Form.



MonthCalendar Control Properties

- In order to set the MonthCalendar control properties, just right click on the MonthCalendar control and go to properties.

- Properties window will appear on the right side of the windows form application.

  - MonthCalendar control comes with different useful properties including, name, location, font, forecolor, back color, margin, MaximumDate, MaximumSize etc. Let's discuss them one by one. Following window will appear as click on the properties.

Chapter 6: Container & Dialog

GroupBox Class

A GroupBox control is a container control that is used to place Windows Forms child controls in a group. The purpose of a GroupBox is to define user interfaces where we can categorize related controls in a group.

In this article, I will discuss how to create a GroupBox control in Windows Forms at design-time as well as run-time. After that, I will discuss various properties and methods available for the GroupBox control.

Creating a GroupBox

We can create a GroupBox control using a Forms designer at design-time or using the GroupBox class in code at run-time (also known as dynamically).

To create a GroupBox control at design-time, you simply drag and drop a GroupBox control from Toolbox to a Form in Visual Studio. After you drag and drop a GroupBox on a Form, the GroupBox looks like Figure 1. Once a GroupBox is on the Form, you can move it around and resize it using the mouse and set its properties and events.

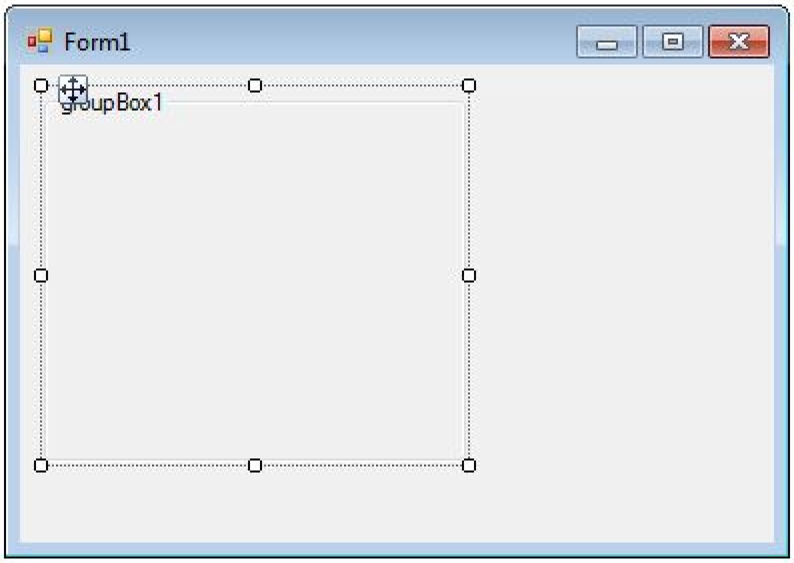


Figure 1

Creating a GroupBox control at run-time is merely a work of creating an instance of GroupBox class, setting its properties and adding GroupBox class to the Form controls.

First step to create a dynamic GroupBox is to create an instance of GroupBox class. The following code snippet creates a GroupBox control object.

```
GroupBox authorGroup = newGroupBox();
```

In the next step, you may set properties of a GroupBox control. The following code snippet sets background color, foreground color, Text, Name, and Font properties of a GroupBox.

```
authorGroup.Name = "GroupBox1";
authorGroup.BackColor = Color.LightBlue;
authorGroup.ForeColor = Color.Maroon;
authorGroup.Text = "Author Details";
authorGroup.Font = newFont("Georgia", 12);
```

Once a GroupBox control is ready with its properties, next step is to add the GroupBox control to the Form. To do so, we use Form.Controls.Add method. The following code snippet adds a GroupBox control to the current Form.

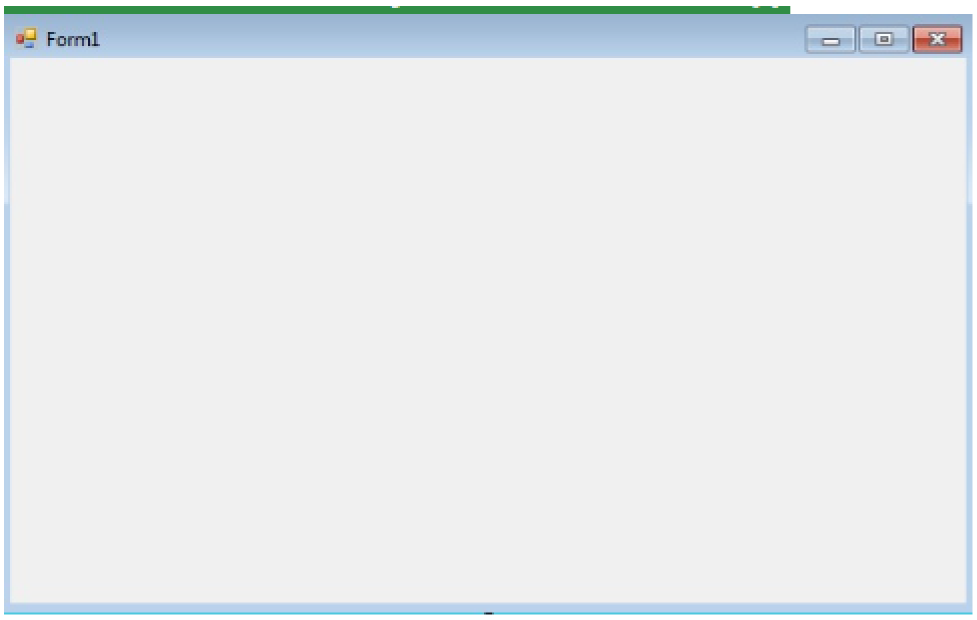
```
this.Controls.Add(authorGroup);
```

In Windows form, GroupBox is a container which contains multiple controls on it and the controls are related to each other. Or in other words, GroupBox is a

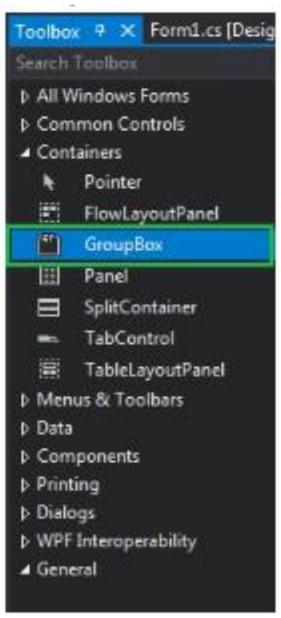
frame display around a group of controls with a suitable optional title. Or a GroupBox is used to categorize the related controls in a group. The GroupBox class is used to represent the windows group box and also provide different types of properties, methods, and events. It is defined under *System.Windows.Forms* namespace. The main use of a group box is to hold a logical group of RadioButton controls.
In C# you can create a GroupBox in the windows form by using two different ways:

1. Design-Time: It is the easiest way to create a GroupBox as shown in the following steps:

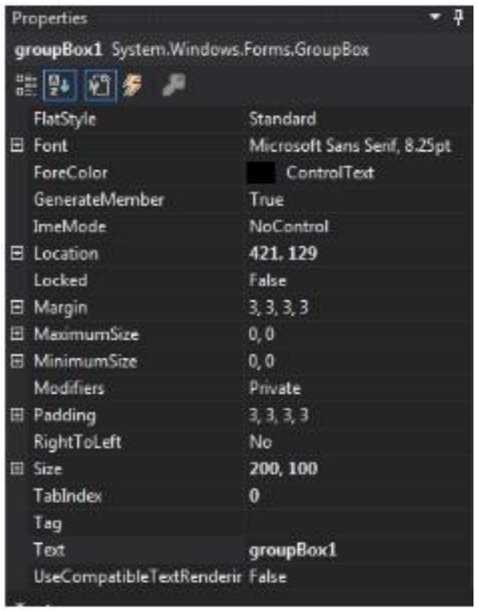
- **Step 1:** Create a windows form as shown in the below image:
  Visual Studio -> File -> New -> Project -> WindowsFormApp



Step 2: Next, drag and drop the GroupBox from the toolbox on the form.



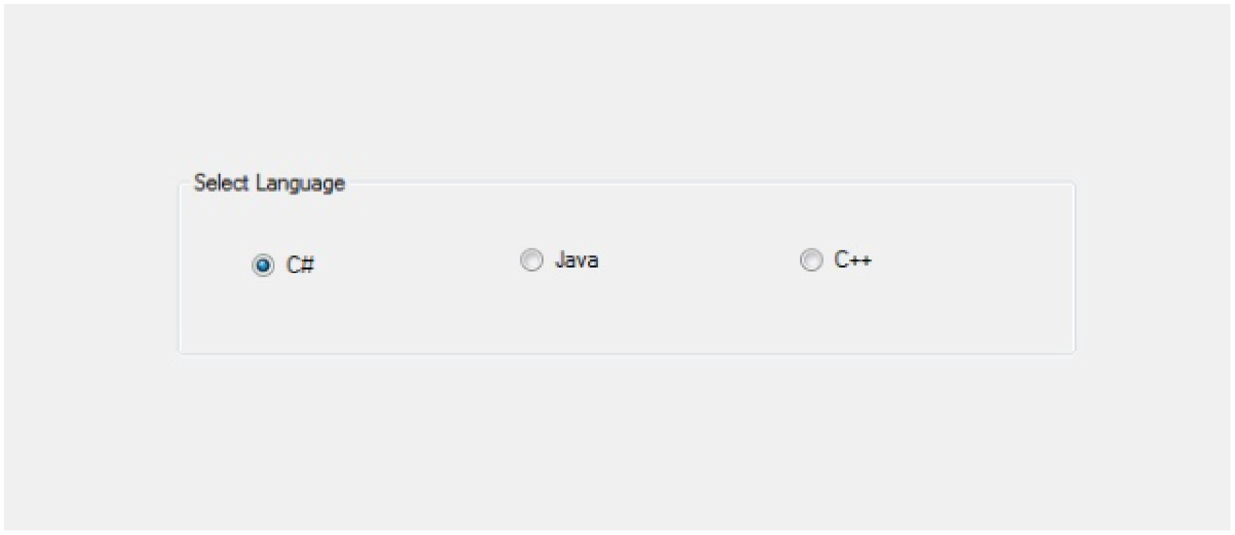
Step 3: After drag and drop you will go to the properties of the GroupBox to modify GroupBox according to your requirement.



groupBox1 System.Windows.Forms.GroupBox

FlatStyle	Standard
Font	Microsoft Sans Serif, 8.25pt
ForeColor	■ ControlText
GenerateMember	True
ImeMode	NoControl
Location	**421, 129**
Locked	False
Margin	3, 3, 3, 3
MaximumSize	0, 0
MinimumSize	0, 0
Modifiers	Private
Padding	3, 3, 3, 3
RightToLeft	No
Size	**200, 100**
TabIndex	**0**
Tag	
Text	**groupBox1**
UseCompatibleTextRenderir	False

Output:



2. Run-Time: It is a little bit trickier than the above method. In this method, you can create a GroupBox programmatically with the help of syntax provided by the GroupBox class. The following steps show how to set the create GroupBox dynamically:

- **Step 1:** Create a GroupBox using the *GroupBox()* constructor is provided by the GroupBox class.
- `// Creating a GroupBox`
- `GroupBox box = new GroupBox();`
- **Step 2:** After creating GroupBox, set the property of the GroupBox provided by the GroupBox class.

Panel:

The Panel Control is a container control to host a group of similar child controls. One of the major uses I have found for a Panel Control is when you need to show and hide a group of controls. Instead of show and hide individual controls, you can simply hide and show a single Panel and all child controls.

Windows Forms Panel controls are used to provide an identifiable grouping for other controls. Typically, you use panels to subdivide a form by function. The Panel control is similar to the GroupBox control; however, only the Panel control can have scroll bars, and only the GroupBox control displays a caption.

Creating a Panel

We can create a Panel Control using the Forms designer at design-time or using the Panel class in code at run-time.

Design-time

To create a Panel Control at design-time, you can drag and drop a Panel Control from the Toolbox to a Form in Visual Studio. After you dragging and dropping a Panel Control to the Form, the Panel looks like Figure 1.

Once a Panel is on the form, you can move it around and resize it using the mouse and set its properties and events.

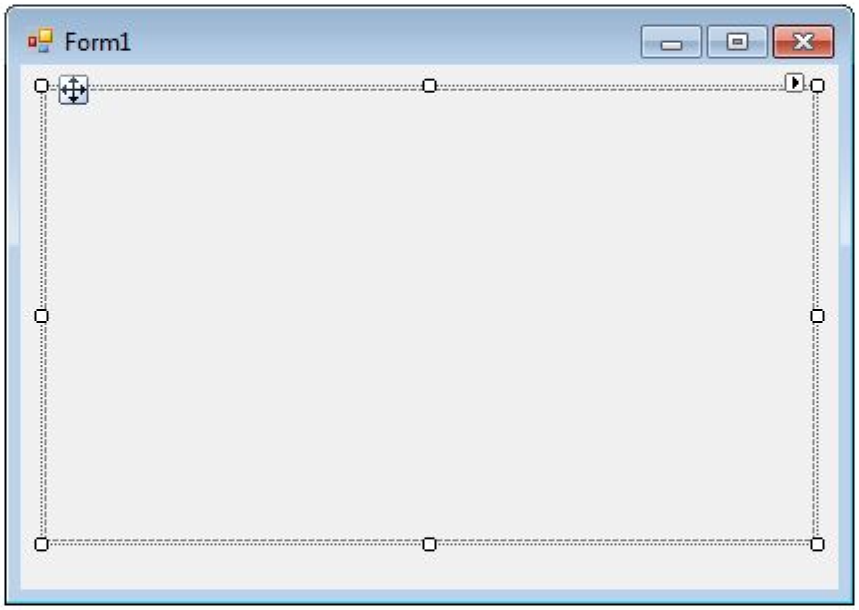


Figure 1

Run-time

Creating a Panel Control at run-time is merely a work of creating an instance of the Panel class, setting its properties and adding the Panel to the form controls.

The first step to create a dynamic Panel is to create an instance of the Panel class. The following code snippet creates a Panel Control object.

```
Panel dynamicPanel = newPanel();
```

In the next step, you may set the properties of a Panel Control.

The following code snippet sets the location, size and Name properties of a Panel.

```
dynamicPanel.Location = new System.Drawing.Point(26, 12);
dynamicPanel.Name = "Panel1";
dynamicPanel.Size = new System.Drawing.Size(228, 200);
dynamicPanel.TabIndex = 0;
```

Once the Panel Control is ready with its properties, the next step is to add the Panel to a form so it becomes a part of the form.

To do so, we use the "Form.Controls.Add" method that adds the Panel Control to the form's controls and displays it on the form based on the location and size of the control.

The following code snippet adds a Panel Control to the current form.

```
Controls.Add(dynamicPanel);
```

Common Dialog boxes:

What is a C# dialog box?

A dialog box in C# is a type of window, which is used to enable common communication or dialog between a computer and its user. A dialog box is most often used to provide the user with the means for specifying how to implement a command or to respond to a question. Windows.Form is a base class for a dialog box.

Sometimes, in a graphical user interface, a window is used to communicate with the user or establish a dialog between the user and the application. This additional window is called a dialog box. It may communicate information to the user; prompt the user for a response or both.

The simplest type of dialog box is the warning which displays a message and may require the user to acknowledge that the message has been read, usually by clicking “OK” or a decision as to whether or not an action should continue by clicking “OK” or “Cancel”.

Some dialog boxes are standard like the warning message or error message. Save the file and enter the password. These are called standard dialog boxes.

A dialog box can also be customized. Such a dialog box is called a custom dialog box.

Dialog boxes are special forms that are non-resizable. They are also used to display the messages to the user. The messages can be error messages, confirmation of the password, confirmation for the deletion of a particular record, Find-Replace utility of the word etc. There are standard dialog boxes to open and save a file, select a folder, print the documents, set the font or color for the text, etc.

MessageBox class is used to display messages to the user. The show() method is used to display a message box with the specified text, caption, buttons, and icon. There are other overloads also available.

Dialog boxes are of two types, which are given below.

1. Modal dialog box
2. Modeless dialog box

A dialog box that temporarily halts the application and the user cannot continue until the dialog has been closed is called modal dialog box. The application may require some additional information before it can continue or may simply wish to confirm that the user wants to proceed with a potentially dangerous course of action. The application continues to execute only after the dialog box is closed; until then the application halts. For example, when saving a file, the user gives a name of an existing file; a warning is shown that a file with the same name exists, whether it should be overwritten or be saved with different name. The file will not be saved unless the user selects “OK” or “Cancel”.

Another type of dialog box, which is used is a modeless dialog box. It is used when the requested information is not essential to continue, so the Window can be left open, while work continues somewhere else. For example, when working in a text editor, the user wants to find and replace a particular word. This can be done, using a dialog box, which asks for the word to be found and replaced. The user can continue to work, even if this box is open.

Dialog box can be configured to constant by setting FormBorderStyle property to FormBorderStyle.FixedDialog, setting MinimizeBox and MaximizeBox property to false. Framework Class Library(FCL) does not provide class as Dialog. The developer creates custom dialog classes by deriving type from System.Windows.Form base class.

Model dialog is displayed, using ShowDialog() method. Modeless dialog boxes are displayed, using Show() method.

Common Dialog Box

The dialog boxes that are used, which are common to all Windows Application. It performs common tasks like saving a file, choosing a font etc. This provides a standard way to the Application interface.

The examples are given below.

- FontDialog
- ColorDialog
- OpenDialog
- SaveDialog

These dialog boxes are implemented by an operating system, so they can be shared across all the Application that runs on that operating system (Windows).

Steps to use common dialog box

- Instantiate the required common dialog box.
- Set the properties of common dialog box, if required.
- Call its ShowDialog() method to invoke the dialog box.

ShowDialog() returns an enumerated type called DialogResult. It defines the identifiers, which indicates which button was clicked. For example, DialogResult.OK and DialogResult.Cancel are some values that indicates OK or Cancel button were clicked respectively.

Open File Dialog Box

The OpenFileDialog allows you to choose a file to be opened in an Application.

For example, the code is given below.

```
OpenFileDialog ofd = new OpenFileDialog();
ofd.Title = "Open a Text File";
ofd.Filter = "Text Files (*.txt) | *.txt | All Files(*.*) | *.*";
//Here you can filter which all files you wanted allow to open
DialogResult dr = ofd.ShowDialog();
if (dr == DialogResult.OK) {
    StreamReader sr = new StreamReader(ofd.FileName);
    txtEx.Text = sr.ReadToEnd();
    sr.Close();
}
```

Save File Dialog Box

The SaveFileDialog box is used to allow the user to select the destination and name of the file to be saved.

For example, the code is given below.

```
SaveFileDialog sfdlg = new SaveFileDialog();
sfdlg.Filter = "Text Files (*.txt) | *.txt"; //Here you can filter
which all files you wanted allow to open
if (sfdlg.ShowDialog() == DialogResult.OK) {
    // Code to write the stream goes here.
}
```

The content can be saved to the file, using appropriate class like *StreamWriter* class in the case of Text Editor Application.

Font and Color Dialog Boxes

FontDialogBox is used to allow the user to select font settings. The ColorDialogBox is used to allow the user to select a color.

For example, the code is given below.

```
//Font Dialog
FontDialog fdlg = new FontDialog();
fdlg.ShowDialog();
txtEx.Font = fdlg.Font;
//Color Dialog
ColorDialog cdlg = new ColorDialog();
cdlg.ShowDialog();
txtEx.ForeColor = cdlg.color;
```

Custom Dialog Box

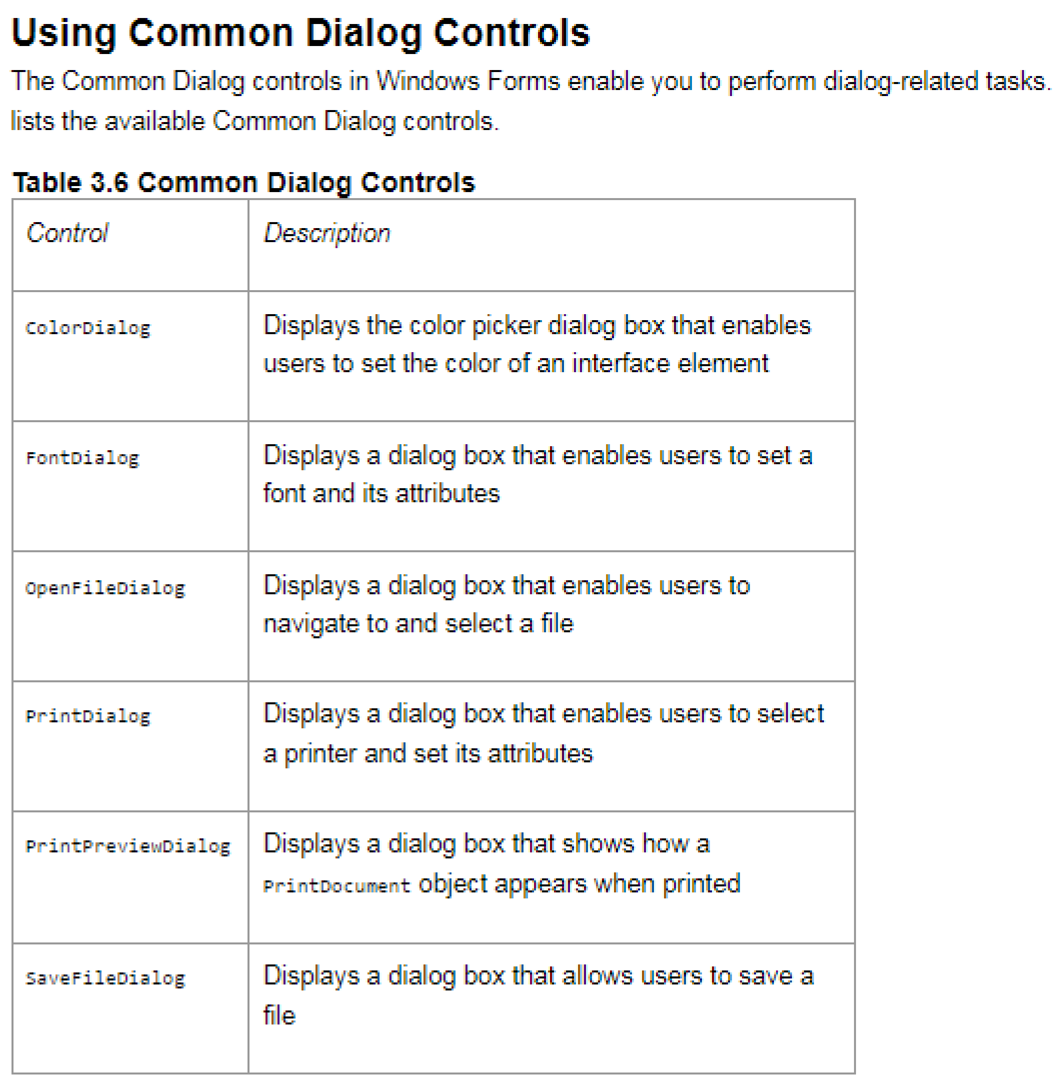
Even though common dialog boxes are useful, they do not support the requirements of domain-specific dialog boxes. Developer need to create their own dialog boxes.

Following steps represent the process of creating Custom Dialog Box

1. Add a form to your project by right clicking the project in Solution Explorer, point to Add and then click Windows Form.
2. In the properties Window, change the FormBorderStyle property to FixedDialog.
3. Customize the appearance of the form, as required.
4. Add controls into this form.

Note

Use ShowDialog() method to open the form as modal dialog box. This method can be used for the custom dialog boxes too. Show() method is used to open the model's dialog box.



Using Common Dialog Controls

The Common Dialog controls in Windows Forms enable you to perform dialog-related tasks. lists the available Common Dialog controls.

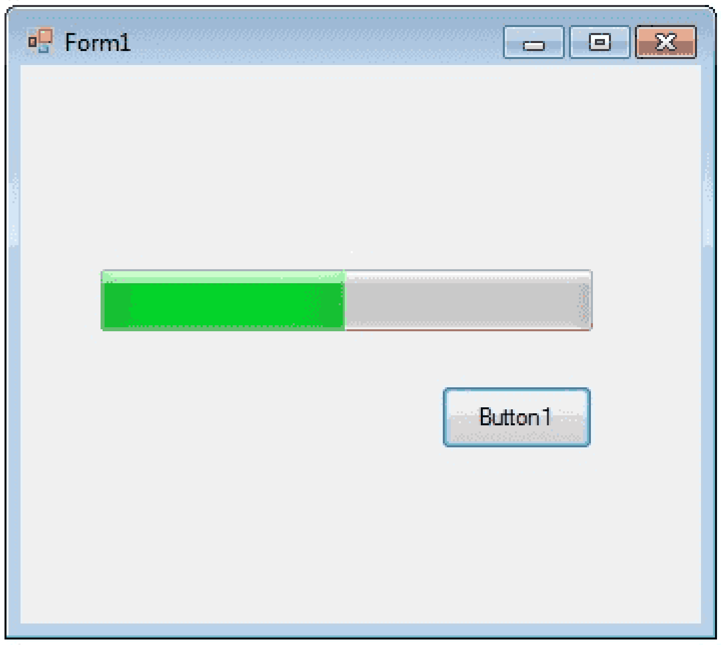
Table 3.6 Common Dialog Controls

Control	*Description*
`ColorDialog`	Displays the color picker dialog box that enables users to set the color of an interface element
`FontDialog`	Displays a dialog box that enables users to set a font and its attributes
`OpenFileDialog`	Displays a dialog box that enables users to navigate to and select a file
`PrintDialog`	Displays a dialog box that enables users to select a printer and set its attributes
`PrintPreviewDialog`	Displays a dialog box that shows how a `PrintDocument` object appears when printed
`SaveFileDialog`	Displays a dialog box that allows users to save a file

ProgressBar:

C# ProgressBar Control

A progress bar is a control that an application can use to indicate the progress of a lengthy operation such as calculating a complex result, downloading a large file from the Web etc.



ProgressBar controls are used whenever an operation takes more than a short period of time. The Maximum and Minimum properties define the range of values to represent the progress of a task.

- Minimum : Sets the lower value for the range of valid values for progress.
- Maximum : Sets the upper value for the range of valid values for progress.
- Value : This property obtains or sets the current level of progress.

By default, Minimum and Maximum are set to 0 and 100. As the task proceeds, the ProgressBar fills in from the left to the right. To delay the program briefly so that you can view changes in the progress bar clearly.

The following C# program shows a simple operation in a progressbar .

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

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            int i;

            progressBar1.Minimum = 0;
            progressBar1.Maximum = 200;

            for (i = 0; i <= 200; i++)
            {
                progressBar1.Value = i;
            }

        }

    }
}
```

A ProgressBar control to display the progress of a file copy operation. The example uses the Minimum and Maximum properties to specify a range for the ProgressBar that is equivalent to the number of files to copy. The code also uses the Step property with the PerformStep method to increment the value of the ProgressBar as a file is copied. This example requires that you have a ProgressBar control created called `pBar1` that is created within a Form and that there is a method created called `CopyFile` (that returns a Boolean value indicating the file copy operation was completed successfully) that performs the file copy operation. The code also requires that an array of strings containing the files to copy is created and passed to the `CopyWithProgress` method defined in the example and that the method is called from another method or event in the Form.

A ProgressBar control is used to represent the progress of a lengthy operation that takes time where a user must wait for the operation to be finished.

In this article, we will see how to create a ProgressBar control in a Windows Forms application using Visual Studio 2017. We will also discuss the properties and methods defined in the ProgressBar class.

Creating a ProgressBar

We can create a ProgressBar control using a Forms designer at design-time or using the ProgressBar class in code at run-time.

Design-time

To create a ProgressBar control at design-time, you simply drag a ProgressBar control from the Toolbox and drop onto a Form in Visual Studio. After you the drag and drop, a ProgressBar is created on the Form; for example the ProgressBar1 is added to the form and looks as in Figure 1.

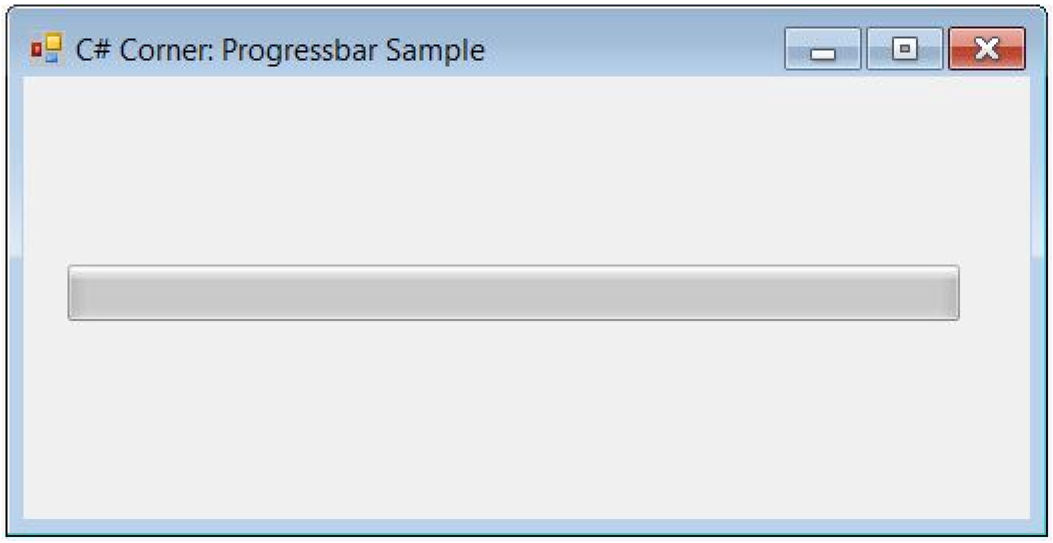
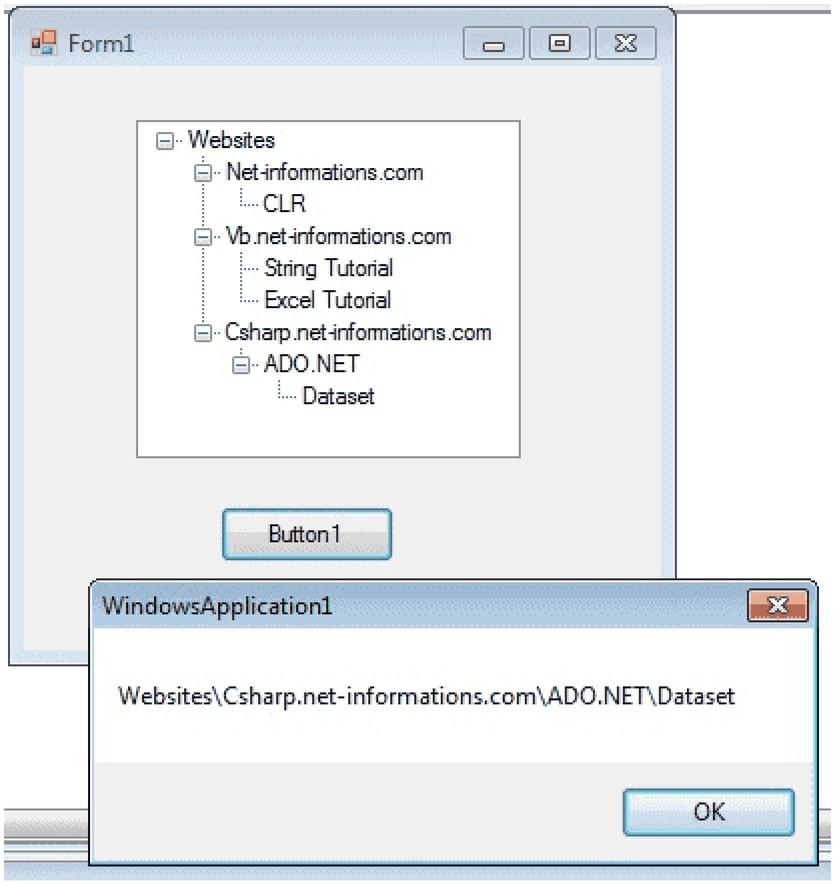


Figure 1

C# Treeview Control

The TreeView control contains a hierarchy of TreeViewItem controls. It provides a way to display information in a hierarchical structure by using collapsible nodes . The top level in a tree view are root nodes that can be expanded or collapsed if the nodes have child nodes.



```
Websites\Csharp.net-informations.com\ADO.NET\Dataset
```

You can explicitly define the TreeView content or a data source can provide the content. The user can expand the TreeNode by clicking the plus sign (+) button, if one is displayed next to the TreeNode, or you can expand the TreeNode by calling the TreeNode.Expand method. You can also navigate through tree views with various properties: FirstNode, LastNode, NextNode, PrevNode, NextVisibleNode, PrevVisibleNode.

The fullpath method of treeview control provides the path from root node to the selected node.

treeView1.SelectedNode.FullPath.ToString ();

Tree nodes can optionally display check boxes. To display the check boxes, set the CheckBoxes property of the TreeView to true.

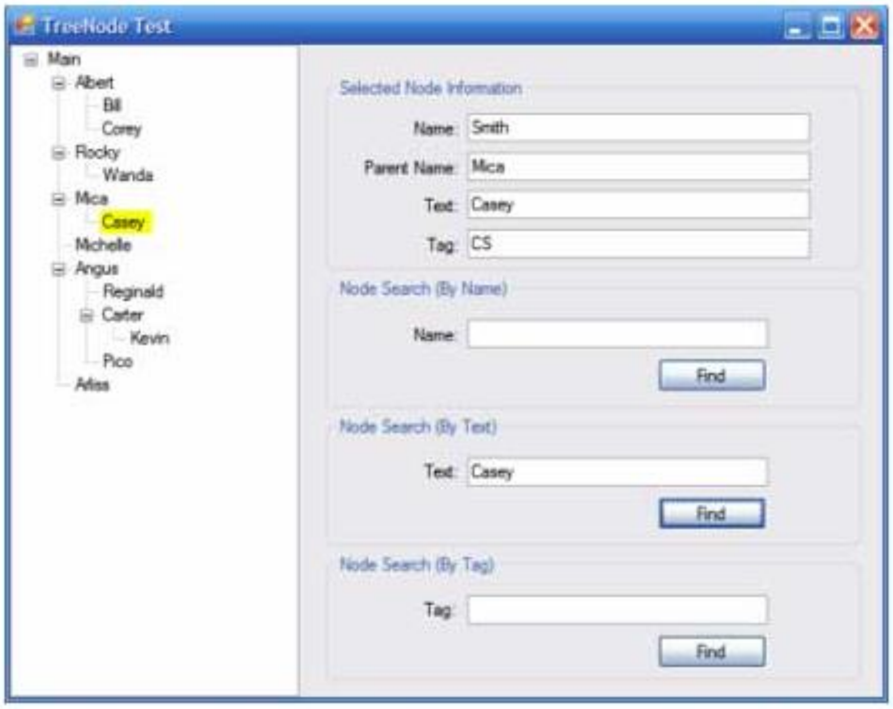
treeView1.CheckBoxes = true;

The following C# program shows a simple demonstration of treeview control.

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

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            TreeNode tNode ;
            tNode = treeView1.Nodes.Add("Websites");

            treeView1.Nodes[0].Nodes.Add("Net-informations.com");
            treeView1.Nodes[0].Nodes[0].Nodes.Add("CLR");

            treeView1.Nodes[0].Nodes.Add("Vb.net-informations.com");
            treeView1.Nodes[0].Nodes[1].Nodes.Add("String Tutorial");
            treeView1.Nodes[0].Nodes[1].Nodes.Add("Excel Tutorial");

            treeView1.Nodes[0].Nodes.Add("Csharp.net-informations.com");
            treeView1.Nodes[0].Nodes[2].Nodes.Add("ADO.NET");
            treeView1.Nodes[0].Nodes[2].Nodes[0].Nodes.Add("Dataset");
        }

        private void button1_Click(object sender, EventArgs e)
        {
            MessageBox.Show(treeView1.SelectedNode.FullPath.ToString ());
        }
    }
}
```

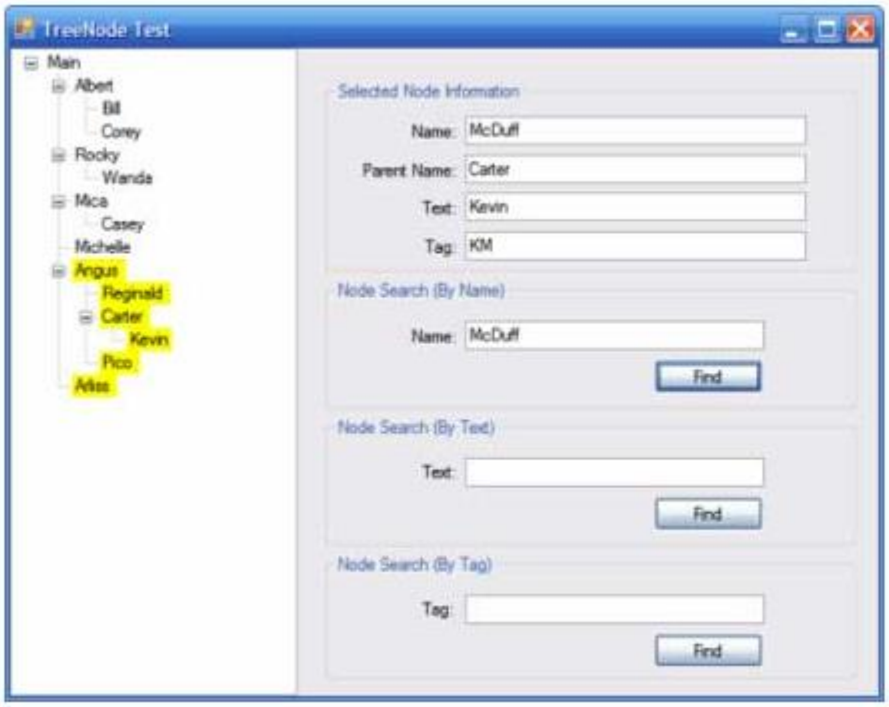
TreeView Introduction

This is a short article addressing some of the basics of working with a TreeView control; the article will address dynamically adding TreeNodes to a TreeView control, searching the nodes to find and highlight a single node or a collection of nodes matching a search term against the TreeNode's tag, text, or name properties, and manually or programmatically selecting nodes.



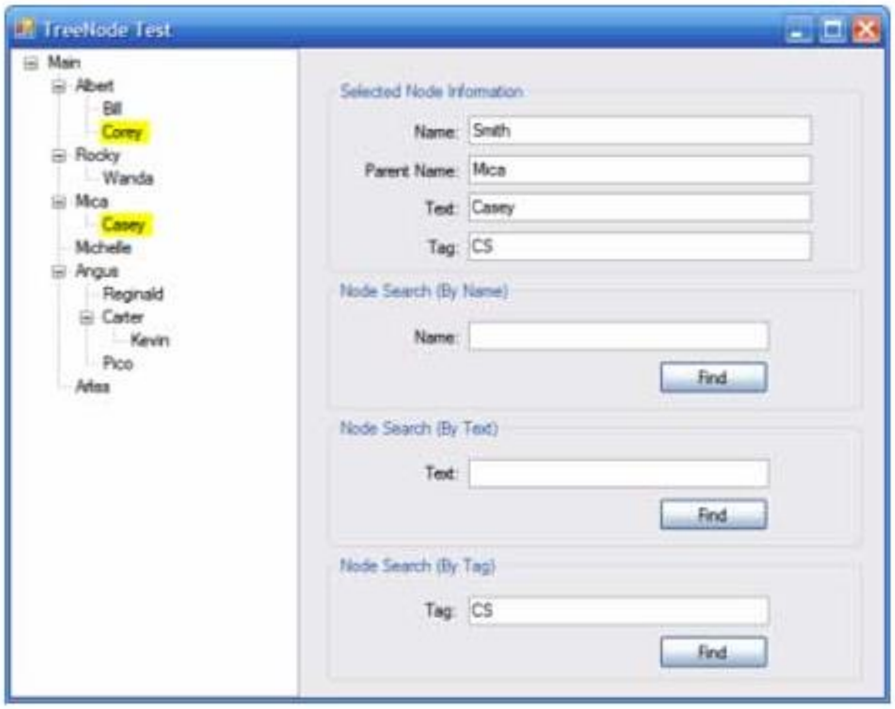
Name: Smith
Parent Name: Mica
Text: Casey
Tag: CS
Name:
Text: Casey
Tag:

Figure 1: Searching a TreeView Control by the Text Property.



Name: McDuff
Parent Name: Carter
Text: Kevin
Tag: KM
Name: McDuff
Text:
Tag:

igure 2: Searching a TreeView Control by the Name Property.



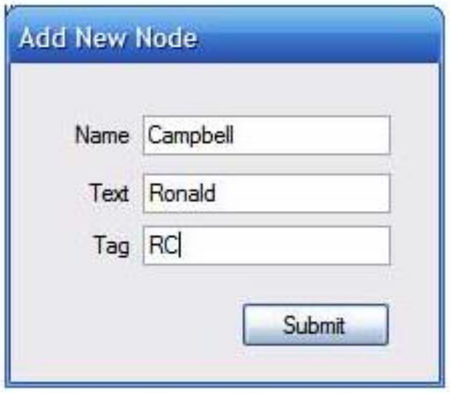
Name: Smith
Parent Name: Mica
Text: Casey
Tag: CS

Name:

Text:

Tag: CS

Figure 3: Searching a TreeView Control by the Name Property.



Name Campbell
Text Ronald
Tag RC

Figure 4: Creating a Node with Specific Properties.

TreeView Application

The application solution contains a single Windows Application project. All code supplied in support of this project is contained in two form classes. One is the main form containing the TreeView and a few controls used to display node information (Figures 1, 2, and 3) and to execute searches for a specific node or group of nodes based upon a user-supplied search term. The other form class (Figure 4) is used to create new nodes; within the application, this form is displayed by selecting a node from the TreeView and then selecting the "Add Node" option from the context menu.

There is nothing custom or fancy done with any of the TreeView related components in this application; it is merely a demonstration of how to work with a TreeView within the context of a Windows Forms application.

The Code: Form 1 - The Main Form

The main form class is a standard window form with a few controls added; the form contains a split container control; on the left-hand side of the control is a TreeView control, on the right-hand side of the splitter are four group boxes; the first group box contains a set of labels and text boxes used to display information about a selected node, the remaining group boxes contains labels, text boxes, and buttons used to conduct various searches of the TreeView's node collection based on the node's text, name, or tag values.

The functionality contained in the class is broken up into several regions; the class begins with the default imports, namespace declaration, and class declaration:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace EasyTreeView {
    public partial class Form1: Form {
        public Form1() {
            InitializeComponent();
            // start off by adding a base treeview node
            TreeNode mainNode = new TreeNode();
            mainNode.Name = "mainNode";
            mainNode.Text = "Main";
            this.treeView1.Nodes.Add(mainNode);
        }
    }
}
```

The form class constructor creates a main node in the TreeView control; at runtime, the user may select this node (or any child node originating form this node) to add additional nodes to the TreeView. The form class also contains a context menu; this context menu contains two options; one to add a new node, and one to delete an existing node. When a new node is requested, the application opens an instance of the New Node dialog; this dialog forces the user to set the name, text, and tag values for the new node. The tag value can be any object but in this example, the tag is limited to holding an additional string value. Once the values have been collected from the dialog, the new node is populated with the information and added to the selected node of the TreeView.

When a node is deleted, the selected node and all of its children are removed from the TreeView; one thing to note here is, if you are associating an object with a node through its tag; you will want to write the handler to destroy that object prior to deleting the selected node.

```
#region Add and Remove Nodes
/// <summary>
    /// Add a Treeview node using a dialog box
    /// forcing the user to set the name and text properties
    /// of the node
    /// </summary>
///
<param name="sender">
</param>
///
<param name="e">
</param>
private void cmnuAddNode_Click(object sender, EventArgs e)
{
    NewNode n = new NewNode();
    n.ShowDialog();
    TreeNode nod = new TreeNode();
    nod.Name = n.NewNodeName.ToString();
    nod.Text = n.NewNodeText.ToString();
    nod.Tag = n.NewNodeTag.ToString();
    n.Close();
    treeView1.SelectedNode.Nodes.Add(nod);
    treeView1.SelectedNode.ExpandAll();
}
/// <summary>
    /// Remove the selected node and it children
    /// </summary>
///
<param name="sender">
</param>
///
<param name="e">
</param>
private void cmnuRemoveNode_Click(object sender, EventArgs e)
{
    treeView1.SelectedNode.Remove();
}
#endregion
```

The next region of code is used to handle TreeView events; there are only two events handled in this section; the TreeView's After Select event and the TreeView's click event. The After Select event handler is used to populate the text boxes used to display information from the selected node (its Name, Text, Tag, and Parent text properties). The find functions described later highlight all found nodes in response to a search by setting the background color of each matching node to Yellow; the click event handler for the TreeView is used to remove all such highlighting.

```
#region Treeview Event Handlers
/// <summary>
/// Display information about the selected node
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void treeView1_AfterSelect(object sender, TreeViewEventArgs e)
{
    try
    {
        txtName.Text = "";
        txtParentName.Text = "";
        txtText.Text = "";
        txtTag.Text = "";
        txtName.Text = treeView1.SelectedNode.Name.ToString();
        txtText.Text = treeView1.SelectedNode.Text.ToString();
        txtTag.Text = treeView1.SelectedNode.Tag.ToString();
        txtParentName.Text = treeView1.SelectedNode.Parent.Text.ToString();
    }
    catch { }
}
/// <summary>
/// Clear nodes marked by the find functions
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void treeView1_Click(object sender, EventArgs e)
{
    ClearBackColor();
}
#endregion
```

The next region in the class is used to find a node by its name property. The method of finding a node by its name is the only find function directly supported by the TreeView; if you want to find a node by something other than its name, you will have to write your own methods to do so. This click event handler populates an array of nodes with matching names. The find method accepts two arguments; the first argument is the search term and the second is a Boolean value used to determine whether or not child nodes should also be included in the search; in this case, the search term is collected from a text box on the form and the option of searching child nodes is enabled by setting the second argument to true.

Once the collection of nodes is created, each matching node has its background color set to yellow so as to highlight the node in the TreeView. Prior to setting the background color of the matching nodes, all of the other nodes in the TreeView are set back to having white backgrounds by calling the Clear Back Color method.

```
#region Find By Name
/// <summary>
/// Use the treeview's built-in find function
/// to search for a node
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void btnFindNode_Click(object sender, EventArgs e)
{
    ClearBackColor();
    try
    {
        TreeNode[] tn = treeView1.Nodes[0].Nodes.Find(txtNodeSearch.Text, true);
        for (int i = 0; i < tn.Length; i++)
        {
            treeView1.SelectedNode = tn[i];
            treeView1.SelectedNode.BackColor = Color.Yellow;
        }
    }
    catch { }
}
```

MaskedTextBox:

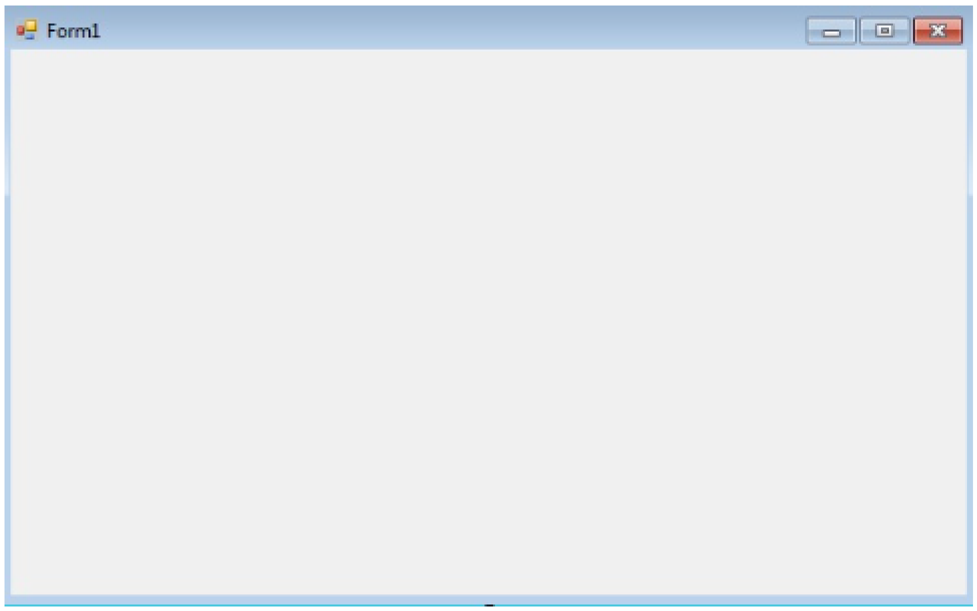
C# | MaskedTextBox Class

In C#, MaskedTextBox control gives a validation procedure for the user input on the form like date, phone numbers, etc. Or in other words, it is used to provide a mask which differentiates between proper and improper user input. The MaskedTextBox class is used to represent the windows masked text box and also provide different types of properties, methods, and events. It is defined under **System.Windows.Forms** namespace.
This class enhanced version of TextBox control, it supports a declarative syntax for receiving or rejecting the user input and when this control display at run

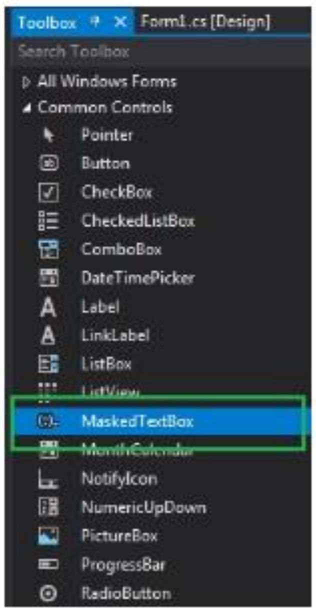
time, it represents the mask as a sequence of prompt characters and optional literal characters. In C# you can create a MaskedTextBox in the windows form by using two different ways:

1. Design-Time: It is the easiest way to create a MaskedTextBox as shown in the following steps:

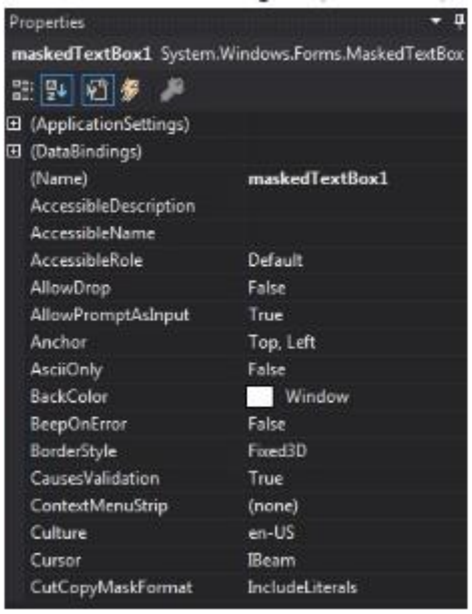
- **Step 1:** Create a windows form as shown in the below image:
  Visual Studio -> File -> New -> Project -> WindowsFormApp



Step 2: Next, drag and drop the MaskedTextBox control from the toolbox to the form.



step 3: After drag and drop you will go to the properties of the MaskedTextBox control to modify MaskedTextBox according to your requirement.

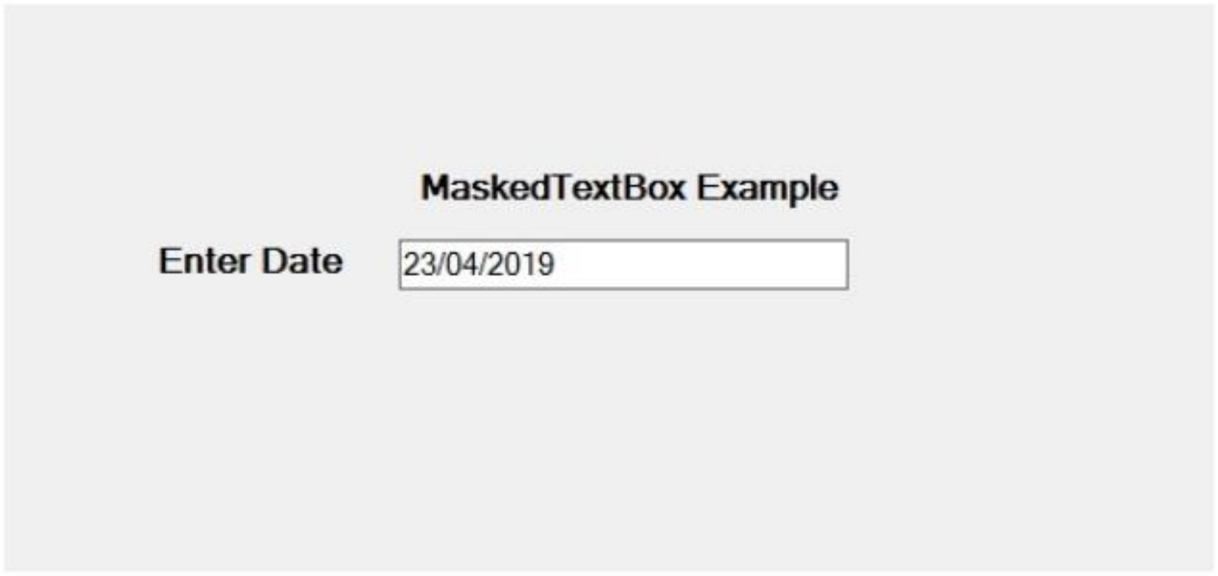


Properties

maskedTextBox1 System.Windows.Forms.MaskedTextBox

(ApplicationSettings)	
(DataBindings)	
(Name)	maskedTextBox1
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
AllowPromptAsInput	True
Anchor	Top, Left
AsciiOnly	False
BackColor	⬜ Window
BeepOnError	False
BorderStyle	Fixed3D
CausesValidation	True
ContextMenuStrip	(none)
Culture	en-US
Cursor	IBeam
CutCopyMaskFormat	IncludeLiterals

Output:



MaskedTextBox Example

Enter Date 23/04/2019