GOVERNMENT OF TAMILNADU
DIRECTORATE OF TECHNICAL EDUCATION
CHENNAI – 600 025
STATE PROJECT COORDINATION UNIT
Diploma in Computer Engineering
Course Code: 1052
M – Scheme

Learning Material of

COMPONENT BASED TECHNOLOGY

for

V Semester D-COMP

Convener
Thiru .D. Arulselvan
HOD/Computer Applications
Thiagarajar Polytechnic College
Salem–636005

Learning material development team:

1. **Mr. V.G.Ravindhren**
   Lecturer(SG) /Computer Engineering
   Seshayee Institute of Technology,
   Trichy- 620 010
2. **Mr A.N.GnanaJeevan**
   Lecturer /Computer Engineering
   Seshayee Institute of Technology,
   Trichy- 620 010
3. **Ms.G.Anushiya**
   Lecturer /Computer Engineering
   Seshayee Institute of Technology,
   Trichy- 620 010

Validated By

Mr.M.Suresh Babu
HOD/Computer Engineering
NPA Centenary Polytechnic College
Kotagiri-643217

STATE BOARD OF TECHNICAL EDUCATION & TRAINING, TAMILNADU.

DIPLOMA IN COMPUTER ENGINEERING

M- SCHEME

(to be implemented to the student Admitted from the Year 2015-2016 on wards)

Course Name : Diploma in Computer Engineering.

Subject Code : 35253

Semester : V

Subject title : COMPONENT BASED TECHNOLOGY

TEACHING & SCHEME OF EXAMINATION:

No. of weeks per Semester: 15 Weeks

Subject	Instruction		Examination			Duration
	Hours/ Week	Hours/ Semester	Internal Assessment	Board Examination	Total	
Component based Technology	4	60	25	75	100	3 Hours

UNITS AND ALLOCATION OF HOURS

UNIT No.	TOPIC	No. of Hours
I	INTRODUCTION TO .NETFRAMEWORK	10
II	INTRODUCTION TO C#	10
III	WINDOW APPLICATION USING WINDOW FORMS	10
IV	APPLICATION DEVELOPMENT USING ADO.NET	10
V	XML	10
	TEST AND REVISION	10
	TOTAL	60

RATIONALE

.NET Framework is changing the way developers write applications. .NET Framework provides a number of components to create many types of applications including those for consoles, Windows, mobile units and the web. Using .NET framework the data can be made available anytime, anywhere and on any device.

This subject introduces the basics of .NET Framework. Writing applications on C#.Net is covered in this course. Concepts of developing Window applications using C#.NET are discussed. This course helps to use ADO.NET to write the applications to connect with the back end database. The subject also enables the users to know the concepts of XML and the XML web services.

OBJECTIVES:

- On completion of the following units of syllabus contents, the students must be able to List the major elements of the .NET Framework and describe some of the major enhancements to the new version of C#.
- Describe the basic structure of a C#.NET project and use the main features of the integrated development environment (IDE).
- Use the new language features and syntax in C# .NET.
- Explain the basic concepts and terminology of object-oriented design specifically for C#.NET.
- Explain and use the basic concepts and terminology of object-oriented programming in C# .NET.
- Create applications by using Microsoft Windows Forms.
- Create applications that use ADO.NET.
- Create components in C# .NET.
- Set up and deploy various types of C# .NET-based applications.
- Develop Window applications using XML as back end database

Detailed Syllabus

UNIT–I INTRODUCTION TO .NET FRAMEWORK		**…. 10 Hours**
1.1	**Introduction to .NET framework:** Dot Net Architecture – Managed Code and the CLR –Intermediate Language, Metadata and JIT Compilation–Automatic Memory Management.	3 Hrs
1.2	**Introduction to.NET framework:** Common Type System(CTS) – Common Language Specification (CLS) – Assembly –Namespace .	3 Hrs
1.3	**Visual Studio .NET–** Using the .NET Framework. Exploring the Visual Studio Integrated Development Environment – System requirement – Versions	2 Hrs
1.4.	**The Framework Class Library-**.NETobjects – ASP.NET-.NET web services–Windows Forms	2 Hrs
UNIT–II INTRODUCTION TO C#		**…. 10 Hours**

2.1	**Elements:** Variables and constants–data types– declaration. Operators– types– precedence – Expressions – Program flow – Decision statements – if .. then, if..then..else, select..case	3 Hrs
2.2.	**Loop statements–** while..end while, do..loop, for..next, for..each..next.	2 Hrs
2.3.	**Types:** Value data types – Structures, Enumerations. Reference data types – Single dimensional– Multi-dimensional arrays–Jagged arrays– Dynamic arrays	2 Hrs
2.4	**Classes & objects** –Abstract & override methods – Creating and using your own classes – Data members and member methods – Instantiate an object –This keyword	3 Hrs
UNIT–III WINDOW APPLICATION USING WINDOW FORMS		**…. 10 Hours**
3.1	**Windows programming**–Creating windows Forms–Working with Toolbox Controls– Button, Check Box, Combo Box, Label, List Box, Radio Button, Text Box, Group Boxes, Picture Box	3 Hrs
3.2.	**Advanced Controls & Events** : Timer , Progress Bar, Month Calendar , ToolTips, Tab Controls, Panels -Events–Click, Close, Deactivate, Load, MouseMove, MouseDown, MouseUp, Keypress ,KeyDown, KeyUp.	2 Hrs
3.3	**Multiple Document Interface (MDI) Forms** – Creating MDI Applications – Creating MDI Child Windows –Arranging MDI Child Windows	2 Hrs
3.4	**Menus and Dialog Boxes** – Creating menus – Menu items – Creating Submenus , Menu Shortcuts, Context menu – Using dialog boxes – show Dialog() method.	3 Hrs
UNIT–IV APPLICATIONDEVELOPMENTUSINGADO.NET		**…. 10 Hours**
4.1	**FeaturesofADO.NET.** Architecture ofADO.NET– ADO.NET providers– connection – Command–Data Adapter–Dataset.	3 Hrs
4.2	**AccessingDatawithADO.NET: c**onnecting to Data Source, Accessing Data with Data set and Data Reader– Modifying Table data using Command Objects – Understanding Data Set and working with Data Column and DataRow – Data Tables - Working with Data GridView	4Hrs
4.3	**Create an ADO.NET application–** Using Stored Procedures.	3 Hrs
UNIT–VXML		**…. 10 Hours**
5.1	**Introduction:** Advantages –HTML Vs XML–Browsing and parsing XML– Creating a XML file–Datails and–Wellformed XML document–XML components-elements– Entities–Comments–Processing instructions–Attributes	3 Hrs

5.2.	**DTD:** Declarations in DTD: Element, Attribute, Entity and Notation– Construction of an XML document – XML Namespaces –Declaring name spaces –Default namespaces – XML schema–Need and use of Schema– Building blocks–Simple elements–Defining attributes–Complex elements	4 Hrs
5.3.	**XMLwith.NET:**XMLSerializationinthe.NETFramework–SOAPFundamentals– Using SOAP with the .NET Framework.	3 Hrs

TEXTBOOKS

Sl.No	TITLE	AUTHOR	PUBLISHER	Yearof Publishing/Edition
1.	Programming In C#, 3E	E. Balagurusamy	Tata McGraw-Hill Education,	2010
2.	Applications of .NET Technology	ISRD Group	TMGH Education Pvt Ltd.,New Delhi	2011

REFERENCES

S.No	Title	Author	Publisher	YearofPublishing/ Edition
1.	IntroducingMicrosoft.NET	David S.Platt	Microsoft Press	SaarcEdition,2001

UNIT I

Objective

1.1 To study about .NET Architecture, CLR and automatic memory management
1.2 To study about Common Type System and Common Language Specification.
1.3 To study about exploring the VS IDE and system requirement.
1.4 To learn about .Net objects and ASP .NET web services and windows Forms

1.1 Introduction to .NET framework

The .NET Framework is Microsoft's Managed Code programming model for building applications on Windows clients, servers, and mobile or embedded devices.

The Microsoft .Net Framework is a platform that provides tools and technologies you need to build Networked Applications as well as Distributed Web Services and Web Applications. The .Net Framework provides the necessary compile time and run-time foundation to build and run any language that conforms to the Common Language Specification (CLS).The main two components of .Net Framework are Common Language Runtime (CLR) and .Net Framework Class Library (FCL).

Figure 1.1 .NET Architecture

The Common Language Runtime (CLR) is the runtime environment of the .Net Framework , that executes and manages all running code like a Virtual Machine. The .Net Framework Class Library (FCL) is a huge collection of language-independent and type-safe reusable classes. The .Net Framework Class Libraries (FCL) are arranged into a logical grouping according to their functionality and usability is called Namespaces.

1.1.2 Managed codes and CLR

Managed Code in Microsoft .Net Framework, is the code that has executed by the Common Language Runtime (CLR) environment. On the other hand Unmanaged Code is directly executed by the computer's CPU. Data types, error-handling mechanisms, creation and destruction rules, and design guidelines vary between managed and unmanaged object models.

The benefits of Managed Code include programmers convenience and enhanced security. Managed code is designed to be more reliable and robust than unmanaged code, examples are Garbage Collection , Type Safety etc. The Managed Code running in a Common Language Runtime (CLR) cannot be accessed outside the runtime environment as well as cannot call directly from outside the runtime environment. This makes the programs more isolated and at the same time computers are more secure . Unmanaged Code can bypass the .NET Framework and make direct calls to the Operating System. Calling unmanaged code presents a major security risk.

CLR

The Common Language Runtime (CLR) is a an Execution Environment . Common Language Runtime (CLR)'s main tasks are to convert the .NET Managed Code to native code , manage running code like a Virtual Machine and also controls the interaction with the Operating System.

Common Language Runtime (CLR) manages Thread executions, Memory Management that is allocation of Objects and Buffers , Garbage Collection (GC) - Clean up the unused Objects and buffers , Exception Handling, Common Type System (CTS) that is all .NET language that conforms to the Common Language Specification (CLS) have the same primitive Data Types, Code safety verifications - code can be verified to ensure type safety, Language integration that is Common Language Runtime (CLR) follow a set of specification called Common Language Specification (CLS) , this will ensure the interoperability between languages, Integrated security and other system services.

1.1.3 Intermediate Languages

Intermediate language (IL) is an object-oriented programming language designed to be used by compilers for the .NET Framework before static or dynamic compilation to machine code. The IL is used by the .NET Framework to generate machine-independent code as the output of compilation of the source code written in any .NET programming language.

IL is a stack-based assembly language that gets converted to bytecode during execution of a virtual machine. It is defined by the common language infrastructure (CLI) specification. As IL is used for automatic generation of compiled code, there is no need to learn its syntax.

This term is also known as Microsoft intermediate language (MSIL) or common intermediate language (CIL).

1.1.4 Metadata and JIT Compilation

Metadata in .Net is binary information which describes the characteristics of a resource. This information include Description of the Assembly , Data Types and members with their declarations and implementations, references to other types and members , Security permissions etc. A module's metadata contains everything that needed to interact with another module.

During the compile time Metadata created with Microsoft Intermediate Language (MSIL) and stored in a file called a Manifest. Both Metadata and Microsoft Intermediate Language (MSIL) together wrapped in a Portable Executable (PE) file. During the runtime of a program Just In Time (JIT) compiler of the Common Language Runtime (CLR) uses the Metadata and converts Microsoft Intermediate Language (MSIL) into native code. When code is executed, the runtime loads metadata into memory and references it to discover information about your code's classes, members, inheritance, and so on. Moreover Metadata eliminating the need for Interface Definition Language (IDL) files, header files, or any external method of component reference.

1.1.5 Automatic Memory Management

The .NET framework has introduced a concept called Garbage Collector. This mechanism keeps track of the allocated memory references and releases the memory when it is not in reference. Since it is automatic it relieves the programmer to manage unused allocated memory. This concept is called Automatic Memory Management.

Free Memory
Object G
Object F
Object E
Object D
Object C
Object B
Object A

Figure 1.2 Managed Heap

1.2 Introduction to .NET framework

1.2.1 Common Type System (CTS)

Common Type System (CTS) describes a set of types that can be used in different .Net languages in common. That is, the Common Type System (CTS) ensure that objects written in different .Net languages can interact with each other. For Communicating between programs written in any .NET complaint language, the types have to be compatible on the basic level.

These types can be Value Types or Reference Types . The Value Types are passed by values and stored in the stack. The Reference Types are passed by references and stored in the heap. Common Type System (CTS) provides base set of Data Types which is responsible for cross language integration. The Common Language Runtime (CLR) can load and execute the source code written in any .Net language, only if the type is described in the Common Type System (CTS) .Most of the members defined by types in the .NET Framework Class Library (FCL) are Common Language Specification(CLS) compliant Types.

1.2.2 Common Language Specification (CLS)

Common Language Specification (CLS) is a set of basic language features that .Net Languages needed to develop Applications and Services, which are compatible with the .Net Framework. When there is a situation to communicate Objects written in different .Net Complaint languages, those objects must expose the features that are common to all the languages . Common Language Specification (CLS) ensures complete interoperability among applications, regardless of the language used to create the application.

Common Language Specification (CLS) defines a subset of Common Type System (CTS) . Common Type System (CTS) describes a set of types that can use different .Net languages have in common , which ensure that objects written in different languages can interact with each other. Most of the members defined by types in the .NET Framework Class Library (FCL) are Common Language Specification (CLS) compliant Types. Moreover Common Language Specification (CLS) standardized by ECMA.

1.2.3 Assembly

.Net Assembly is a logical unit of code, that contains code which the Common Language Runtime (CLR) executes. It is the smallest unit of deployment of a .net application and it can be a **.dll** or an .**exe**. Assembly is really a collection of types and resource information that are built to work together and form a logical unit of functionality. It include both executable application files that you can run directly from Windows without the need for any other programs (.exe files), and libraries (.dll files) for use by other applications.

Assemblies are the building blocks of .NET Framework applications. During the compile time Metadata is created with Microsoft Intermediate Language (MSIL) and stored in a file called Assembly Manifest . Both Metadata and Microsoft Intermediate Language (MSIL) together wrapped in a Portable Executable (PE) file. Assembly Manifest contains information about itself. This information is called Assembly Manifest, it contains information about the members, types, references and all the other data that the runtime needs for execution.

Every Assembly you create contains one or more program files and a Manifest. There are two types program files : Process Assemblies (EXE) and Library Assemblies (DLL). Each Assembly can have only one entry point (that is, DllMain, WinMain, or Main).

We can create two types of Assembly:

1. Private Assembly

2. Shared Assembly

A private Assembly is used only by a single application, and usually it is stored in that application's install directory. A shared Assembly is one that can be referenced by more than one application. If multiple applications need to access an Assembly, we should add the Assembly to the Global Assembly Cache (GAC). There is also a third and least known type of an assembly: Satellite Assembly . A Satellite Assembly contains only static objects like images and other non-executable files required by the application.

1.2.4 Namespace

NameSpace is the Logical group of types or we can say namespace is a container (e.g Class, Structures, Interfaces, Enumerations, Delegates etc.), example System.IO logically groups input output related features , System.Data.SqlClient is the logical group of ado.net Connectivity with Sql server related features. In Object Oriented world, many times it is possible that programmers will use the same class name, Qualifying NameSpace with class name can avoid this collision.

Namespaces allow you to create a system to organize your code. A good way to organize your *namespaces* is via a hierarchical system. You put the more general names at the top of the hierarchy and get more specific as you go down. This hierarchical system can be represented by nested *namespaces*. Below shows how to create a nested *namespace*. By placing code in different sub-namespaces, you can keep your code organized.

Example :-

```
// Namespace Declaration
using System;

// The Namespace
namespace MyNameSpace
{
   // Program start class
   class MyClass
   {
     //Functionality
   }
}
```

1.3 Visual Studio .NET

1.3.1 Using .NET Framework. Exploring the Visual Studio Integrated Development Environment

An Integrated Development Environment (IDE) is software that facilitates application development. In the context of .NET-based applications, Visual Studio is the most commonly used IDE. Some of the key features included are:

- Single IDE for all .NET applications. Therefore no switching required to other IDEs for developing .NET applications
- Single .NET solution for an application which has been built on code written in multiple languages
- Code editor supporting Intelligence and code refactoring
- Compilation from within the environment based on defined configuration options
- Integrated debugger that works at source and machine level
- Plug-in architecture that helps to add tools for domain specific languages
- Customizable environment to help the user to configure the IDE based on the required settings
- Browser that is built-in within the IDE helps to view content from internet such as help, source-code, etc. in online mode.

Visual Studio is integrated with .NET and includes the features of language specific environments, from one of its earlier versions (VS 6.0). It provides a single workspace consisting of a multiple-document interface in which activities related to code development such as editing, compiling, debugging, etc. is easily possible. The main facility that this IDE provides is form creation during design-time. By placing the controls in the layout the display of the application can be rendered at runtime. Hence, IDE provides simpler way of building applications in lesser time.

The latest version of Visual Studio 2010 IDE released with .NET 4.0 is designed to develop applications targeting Windows 7. It has additional features such as navigate to, incremental search, pascal case searching, view call hierarchy, multi-monitor support, code intellisense support (for classes and methods), HTML and Javascript snippet support in editor, tools to aid parallel programming and debugging improvements (Intellitrace, Pinned data tips, Breakpoint labels, etc.). The IDE can also be extended to customize the appearance and its behavior using macros and add-ins. Some features such as text size option and color customization in editor allow easy accessibility to people with disabilities.

Development of applications needs to consider the time necessary for lengthy learning process to work with the IDE due to the complicated integration of all the facilities included in IDE.

1.3.2 System requirement

Requirements	Supported Operating Systems
Visual Studio Express 2013 with Update 5 for Windows Desktop **Hardware Requirements** • 1.6 GHz or faster processor • 1 GB of RAM (1.5 GB if running on a virtual machine) • 5 GB of available hard disk space • 5400 RPM hard drive • DirectX 9-capable video card running at 1024 x 768 or higher display resolution **Additional Requirements** On Windows 8.1 and Windows Server 2012 R2, KB2883200 (available through Windows Update) is required.	Windows 7 SP1 (x86 and x64) Windows 8 (x86 and x64) Windows 8.1 (x86 and x64) Windows Server 2008 R2 SP1 (x64) Windows Server 2012 (x64) Windows Server 2012 R2 (x64)

Table 1.1

1.3.3 Versions

Overview of .NET Framework release history

Version number	CLR version	Release date	Development tool	Included in		Replaces
				Windows	Windows Server	
1.0	1.0	2002-02-13	Visual Studio .NET[4]	XP[a]	N/A	N/A
1.1	1.1	2003-04-24	Visual Studio .NET 2003[4]	N/A	2003	1.0[5]
2.0	2.0	2005-11-07	Visual Studio 2005[6]	N/A	2003, 2003 R2,[7] 2008 SP2, 2008 R2 SP1	N/A

3.0	2.0	2006-11-06	Expression Blend[8][b]	Vista	2008 SP2, 2008 R2 SP1	2.0
3.5	2.0	2007-11-19	Visual Studio 2008[9]	7, 8[c], 8.1[c], 10[c]	2008 R2 SP1	2.0, 3.0
4.0	4	2010-04-12	Visual Studio 2010[10]	N/A	N/A	N/A
4.5	4	2012-08-15	Visual Studio 2012[11]	8	2012	4.0
4.5.1	4	2013-10-17	Visual Studio 2013[12]	8.1	2012 R2	4.0, 4.5
4.5.2	4	2014-05-05	N/A	N/A	N/A	4.0–4.5.1
4.6	4	2015-07-20	Visual Studio 2015[13]	10	N/A	4.0–4.5.2
4.6.1	4	2015-11-30[14]	Visual Studio 2015 Update 1	10 v1511	N/A	4.0–4.6
4.6.2	4	2016-08-02[15]		10 v1607	N/A	4.0–4.6.1

Table 1.2

1.4 The framework Class library

The .NET Framework class library is a library of classes, interfaces, and value types that provide access to system functionality. It is the foundation on which .NET Framework applications, components, and controls are built. The namespaces and namespace categories in the class library are listed in the following table and documented in detail in this reference. The

namespaces and categories are listed by usage, with the most frequently used namespaces appearing first.

Namespace	Description
System	The System namespace contains fundamental classes and base classes that define commonly-used value and reference data types, events and event handlers, interfaces, attributes, and processing exceptions.
System.Activities	The System.Activities namespaces contain all the classes necessary to create and work with activities in Window Workflow Foundation.
System.AddIn	The System.AddIn namespaces contain types used to identify, register, activate, and control add-ins, and to allow add-ins to communicate with a host application.
System.CodeDom	The System.CodeDom namespaces contain classes that represent the elements of a source code document and that support the generation and compilation of source code in supported programming languages.
System.Collections	The System.Collections namespaces contain types that define various standard, specialized, and generic collection objects.
System.ComponentModel	The System.ComponentModel namespaces contain types that implement the run-time and design-time behavior of components and controls. Child namespaces support the Managed Extensibility Framework (MEF), provide attribute classes that define metadata for ASP.NET Dynamic Data controls, and contain types that let you define the design-time behavior of components and their user interfaces.

System.Configuration	The System.Configuration namespaces contain types for handling configuration data, such as data in machine or application configuration files. Child namespaces contain types that are used to configure an assembly, to write custom installers for components, and to support a pluggable model for adding functionality to, or removing functionality from, both client and server applications.
System.Data	The System.Data namespaces contain classes for accessing and managing data from diverse sources. The top-level namespace and a number of the child namespaces together form the ADO.NET architecture and ADO.NET data providers. For example, providers are available for SQL Server, Oracle, ODBC, and OleDB. Other child namespaces contain classes used by the ADO.NET Entity Data Model (EDM) and by WCF Data Services.
System.Deployment	The System.Deployment namespaces contain types that support deployment of ClickOnce applications.
System.Device.Location	The System.Device.Location namespace allows application developers to easily access the computer's location by using a single API. Location information may come from multiple providers, such as GPS, Wi-Fi triangulation, and cell phone tower triangulation. The System.Device.Location classes provide a single API to encapsulate the multiple location providers on a computer and support seamless prioritization and transitioning between them. As a result, application developers who use this API do not need to tailor applications to specific hardware configurations.
System.Diagnostics	The System.Diagnostics namespaces contain types that enable you to interact with system processes, event logs, and performance counters. Child namespaces contain types to interact with code analysis tools, to support contracts, to extend design-time support for application monitoring and instrumentation, to log event data using the Event Tracing for Windows (ETW) tracing subsystem, to read to and write from event logs and collect performance data,

	and to read and write debug symbol information.
System.DirectoryServices	The System.DirectoryServices namespaces contain types that provide access to Active Directory from managed code.
System.Drawing	The System.Drawing parent namespace contains types that support basic GDI+ graphics functionality. Child namespaces support advanced two-dimensional and vector graphics functionality, advanced imaging functionality, and print-related and typographical services. A child namespace also contains types that extend design-time user-interface logic and drawing.
System.Dynamic	The System.Dynamic namespace provides classes and interfaces that support Dynamic Language Runtime.
System.EnterpriseServices	The System.EnterpriseServices namespaces contain types that define the COM+ services architecture, which provides an infrastructure for enterprise applications. A child namespace supports Compensating Resource Manager (CRM), a COM+ service that enables non-transactional objects to be included in Microsoft Distributed Transaction Coordinator (DTC) transactions. Child namespaces are described briefly in the following table and documented in detail in this reference.
System.Globalization	The System.Globalization namespace contains classes that define culture-related information, including language, country/region, calendars in use, format patterns for dates, currency, and numbers, and sort order for strings. These classes are useful for writing globalized (internationalized) applications. Classes such as StringInfo and TextInfo provide advanced globalization functionalities, including surrogate support and text element processing.
System.IdentityModel	The System.IdentityModel namespaces contain types that are used to provide authentication and authorization for .NET applications.
System.IO	The System.IO namespaces contain types that support input and output, including the ability to read and write data to streams either synchronously or asynchronously, to compress data in streams, to create and use isolated stores, to map files to an application's

	logical address space, to store multiple data objects in a single container, to communicate using anonymous or named pipes, to implement custom logging, and to handle the flow of data to and from serial ports.
System.Linq	The System.Linq namespaces contain types that support queries that use Language-Integrated Query (LINQ). This includes types that represent queries as objects in expression trees.
System.Management	The System.Management namespaces contain types that provide access to management information and management events about the system, devices, and applications instrumented to the Windows Management Instrumentation (WMI) infrastructure. These namespaces also contain types necessary for instrumenting applications so that they expose their management information and events through WMI to potential customers.
System.Media	The System.Media namespace contains classes for playing sound files and accessing sounds provided by the system.
System.Messaging	The System.Messaging namespaces contain types that enable you to connect to, monitor, and administer message queues on the network and to send, receive, or peek messages. A child namespace contains classes that can be used to extend design-time support for messaging classes.

Table 1.3

1.4.1 .NET objects

The .NET Framework class library is a collection of reusable types that tightly integrate with the common language runtime. The class library is object oriented, providing types from which your own managed code can derive functionality. This not only makes the .NET Framework types easy to use, but also reduces the time associated with learning new features of the .NET Framework.

For example, the .NET Framework collection classes implement a set of interfaces that you can use to develop your own collection classes. Your collection classes will blend seamlessly with the classes in the .NET Framework.

1.4.2 ASP .NET

ASP.NET is a web application framework developed and marketed by Microsoft to allow programmers to build dynamic web sites. It allows you to use a full featured programming language such as C# or VB.NET to build web applications easily.

ASP.NET is a web development platform, which provides a programming model, a comprehensive software infrastructure and various services required to build up robust web applications for PC, as well as mobile devices.

ASP.NET works on top of the HTTP protocol, and uses the HTTP commands and policies to set a browser-to-server bilateral communication and cooperation.

ASP.NET is a part of Microsoft .Net platform. ASP.NET applications are compiled codes, written using the extensible and reusable components or objects present in .Net framework. These codes can use the entire hierarchy of classes in .Net framework.

The ASP.NET application codes can be written in any of the following languages:

- C#
- Visual Basic.Net
- Jscript
- J#

ASP.NET is used to produce interactive, data-driven web applications over the internet. It consists of a large number of controls such as text boxes, buttons, and labels for assembling, configuring, and manipulating code to create HTML pages.

1.4.3 .NET web services

A web service is a web-based functionality accessed using the protocols of the web to be used by the web applications. There are three aspects of web service development:

- Creating the web service
- Creating a proxy
- Consuming the web service

Creating a Web Service

A web service is a web application which is basically a class consisting of methods that could be used by other applications. It also follows a code-behind architecture such as the ASP.NET web pages, although it does not have a user interface.

Creating the Proxy

A proxy is a stand-in for the web service codes. Before using the web service, a proxy must be created. The proxy is registered with the client application. Then the client application makes the calls to the web service as it were using a local method.

The proxy takes the calls, wraps it in proper format and sends it as a SOAP request to the server. SOAP stands for Simple Object Access Protocol. This protocol is used for exchanging web service data.

When the server returns the SOAP package to the client, the proxy decodes everything and presents it to the client application

Consuming the Web Service

For using the web service, create a web site under the same solution. This could be done by right clicking on the Solution name in the Solution Explorer. The web page calling the web service should have a label control to display the returned results and two button controls one for post back and another for calling the service.

1.4.4 Windows Forms

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. 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.

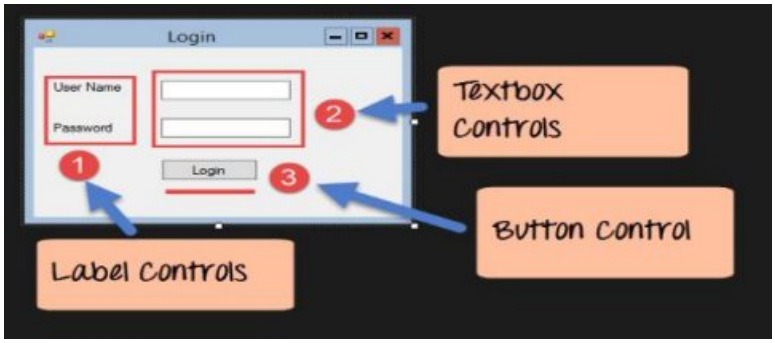


Figure 1.3

So an example of the controls available in the above application

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.

Review Questions
UNIT - I

Part A (2 marks)

1. What is the purpose of CLR?
2. What is the design principle of Common Intermediate Language (CIL)
3. What is meant by Common type system (CTS)?
4. What is a Window Form application?
5. List the steps of web service development.
6. What are the contents of System namespace?
7. What are the contents of System.Data namespace?
8. What are the contents of System.Drawing namespace?
9. What is meant by LINQ?

Part B (3 marks)

1. List the benefits of CLR.
2. What is the use of JIT compiler in .NET runtime?
3. Write notes on Metadata
4. Write notes on Assembly
5. List the uses of ASP.NET

Part C (5 marks)

1. What is a namespace? What are the uses of it? Explain with an example the creation of a namespace.
2. List and explain the features of Visual studio IDE.
3. List any five built-in namespaces and discuss the purpose of each.
4. Explain the three aspects of web service development in detail.
5. List and discuss the features of .NET framework.

*****

UNIT II
INTRODUCTION TO C#

Objectives:

- To understand the elements of C#
- To get familiarize with the control structures
- To know the different kinds of primary data types and derived data types
- To correlate the OOP concepts in the context of C#

C# is a simple, modern, general-purpose, object-oriented programming language developed by Microsoft within its .NET initiative led by Anders Hejlsberg C# programming is very much based on C and C++ programming languages. Having a basic understanding of C or C++ programming, will make it easier to learn C#.

C# is an elegant object-oriented language that enables developers to build a variety of secure and robust applications that run on the **.NET Framework.** You can use C# to create Windows applications, Web services, mobile applications, client-server applications, database applications, and much, much more.

The following reasons make C# a widely used professional language:

- It is a modern, general-purpose programming language
- It is object oriented.
- It is component oriented.
- It is easy to learn.
- It is a structured language.
- It produces efficient programs.
- It can be compiled on a variety of computer platforms.
- It is a part of .Net Framework.

By the help of C# programming language, we can develop different types of secured and robust applications:

- Window applications
- Web applications
- Distributed applications
- Web service applications
- Database applications etc.

Integrated Development Environment (IDE) for C#

Microsoft provides the following development tools for C# programming:

- Visual Studio 2010 (VS)
- Visual C# 2010 Express (VCE)
- Visual Web Developer

The last two are freely available from Microsoft official website. Using these tools, you can write all kinds of C# programs from simple command-line applications to more complex

applications. You can also write C# source code files using a basic text editor like Notepad, and compile the code into assemblies using the command-line compiler, which is again a part of the .NET Framework. Visual C# Express and Visual Web Developer Express edition are trimmed down versions of Visual Studio and has the same appearance. They retain most features of Visual Studio.

The .NET Framework

The .NET Framework consists of

- Common Language Runtime (CLR) and
- the .NET Framework class library.

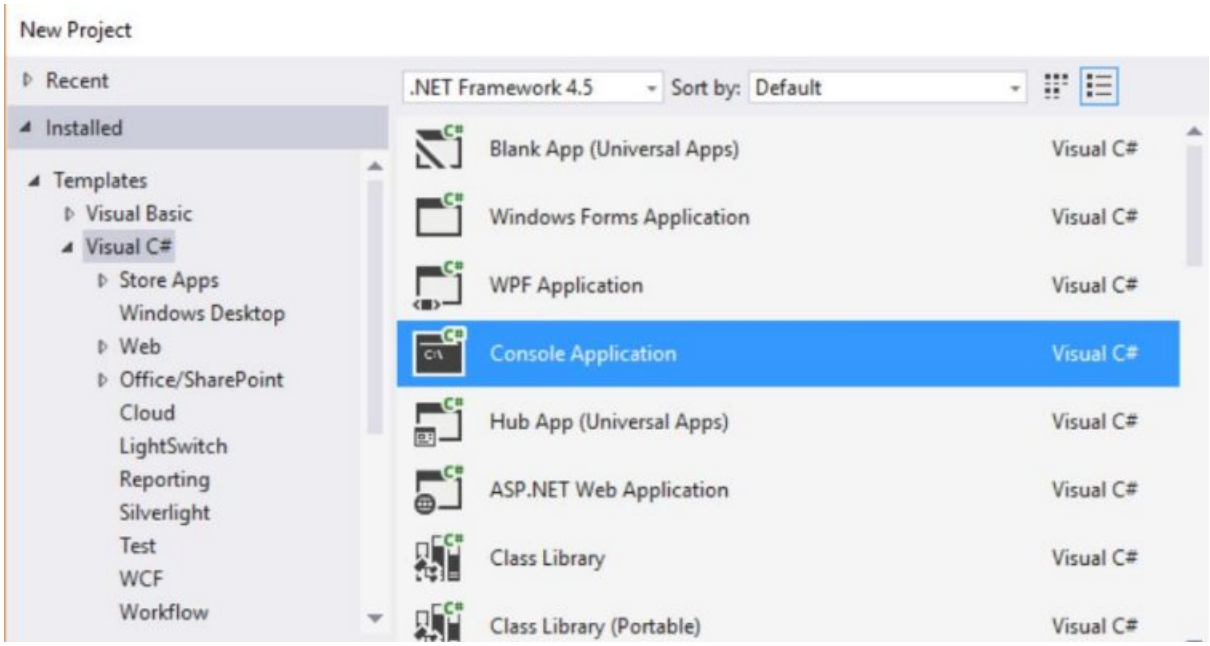
The CLR is the foundation of the .NET Framework. It manages code at execution time, providing core services such as memory management, code accuracy, and many other aspects of your code.

The class library is a collection of classes, interfaces, and value types that enable you to accomplish a range of common programming tasks, such as data collection, file access, and working with text.

C# programs use the .NET Framework class library extensively to do common tasks and provide various functionalities.

First C# Program

To create a C# program, you need to install an integrated development environment (IDE) with coding and debugging tools. We can use **Visual Studio Community edition**, which is available to download for free. After installing it, choose the default configuration. Next, click **File->New->Project** and then choose **Console Application** as shown below:



Enter a name for your Project and click OK.

Creating Hello World Program

A C# program consists of the following parts:

- Namespace declaration
- A class
- Class methods
- Class attributes
- A Main method
- Statements and Expressions
- Comments

Let us look at a simple code that prints the words "Hello World":

```
using System;
namespace HelloWorldApplication
{
class HelloWorld
{
static void Main(string[] args)
        {
        /* my first program in C# */
        Console.WriteLine("Hello World");
        Console.ReadKey();
        }
}
}
```


When this code is compiled and executed, it produces the following result:

```
Hello World
```

Let us look at the various parts of the given program:

1. The first line of the program **using System;** - the **using** keyword is used to include the **System** namespace in the program. A program generally has multiple **using** statements.
2. The next line has the **namespace** declaration. A **namespace** is a collection of classes. The *HelloWorldApplication* namespace contains the class *HelloWorld*.
3. The next line has a **class** declaration, the class *HelloWorld* contains the data and method definitions that your program uses. Classes generally contain multiple methods. Methods define the behavior of the class. However, the *HelloWorld* class has only one method **Main**.
4. The next line defines the **Main** method, which is the **entry point** for all C# programs. The **Main** method states what the class does when executed.
5. Anything written inside /*...*/ is considered as comment. Comments are ignored by the compiler for compilation.
6. The Main method specifies its behavior with the statement **Console.WriteLine("Hello World");**

*WriteLine i*s a method of the *Console c*lass defined in the *System n*amespace. This statement causes the message "Hello, World!" to be displayed on the screen.

7. The last line **Console.ReadKey()** makes the program wait for a key press and it prevents the screen from running andclosing quickly when the program is launched from Visual Studio .NET.

It is important to note the following points:

- C# is case sensitive.
- All statements and expression must end with a semicolon (;).
- The program execution starts at the Main method.
- Unlike Java, program file name could be different from the class name

The *using* Keyword
The first statement in any C# program is using System; The **using** keyword is used for including the namespaces in the program. A program can include multiple using statements.

The class Keyword
The **class** keyword is used for declaring a class.

Comments in C#
Comments are used for explaining code. Compilers ignore the comment entries. The multiline comments in C# programs start with /* and terminates with the characters */ as shown below:


```
/* This program demonstrates
The basic syntax of C# programming
Language */
```

Single-line comments are indicated by the '//' symbol. For example,

```
}//end class Rectangle
```

Member Variables
Variables are attributes or data members of a class, used for storing data.

Member Functions
Functions are set of statements that perform a specific task. The member functions of a class are declared within the class.

Identifiers
An identifier is a name used to identify a class, variable, function, or any other user defined item. The basic rules for naming classes in C# are as follows:

- A name must begin with a letter that could be followed by a sequence of letters, digits (0 - 9) or underscore. The first character in an identifier cannot be a digit.
- It must not contain any embedded space or symbol such as ? - +! @ # % ^ & * ( ) [ ] { } . ; : " ' / and \. However, an underscore (_) can be used.
- It should not be a C# keyword

C# Keywords

Keywords are reserved words predefined to the C# compiler. These keywords cannot be used as identifiers. However, if you want to use these keywords as identifiers, you may prefix the keyword with the @ character. In C#, some identifiers have special meaning in context of code, such as get and set are called contextual keywords.

The following table lists the reserved keywords and contextual keywords in C#:

Reserved words

abstract	as	base	bool	break	byte	case
catch	char	checked	class	const	continue	decimal
default	delegate	do	double	else	enum	event
explicit	extern	false	finally	fixed	float	for
foreach	goto	if	implicit	In	in (generic modifier)	int
interface	internal	is	lock	long	namespace	new
null	object	operator	out	out (generic modifier)	override	params
private	protected	public	readonly	ref	return	sbyte
sealed	short	sizeof	stackalloc	static	string	struct
switch	this	throw	true	try	typeof	uint
ulong	unchecked	unsafe	ushort	using	virtual	void
volatile	while					

Elements

Variables

A variable is nothing but a name given to a storage area that our programs can manipulate.

Each variable in C# has a specific type, which determines the size and layout of the variable's memory the range of values that can be stored within that memory and the set of operations that can be applied to the variable.

Creating a **variable** reserves a memory location, or a space in memory, for storing values. It is called **variable** because the information stored in that location can be changed when the program is running.

To use a variable,

1. it must first be declared by specifying the **name** and **data type**.
2. A variable name, also called an **identifier**, can contain letters, numbers and the underscore character (_) and must start with a letter or underscore.
3. Although the name of a variable can be any set of letters and numbers, the best identifier is **descriptive** of the data it will contain. This is very important in order to create clear, understandable and readable code!

DefiningVariables

Syntax for variable definition in C# is:
<data_type> <variable_list>;

Here, data_type must be a valid C# data type including char, int, float, double, or any user-defined data type, and variable_list may consist of one or more identifier names separated by commas.

Some valid variable definitions are shown here:
int i, j, k;
char c, ch;
float f, salary;

Variable Types

The variables in C#, are categorized into the following types:
- Value types
- Reference types
- Pointer types

Value Type

Value type variables can be assigned a value directly. They are derived from the class **System.ValueType**. The value types directly contain data. Some examples are **int, char, and float.**

Reference Type

The reference types do not contain the actual data stored in a variable, but they contain a reference to the variables. In other words, they refer to a memory location.

Object Type

The **Object Type** is the ultimate base class for all data types in C# Common Type System (CTS. The object types can be assigned values of any other types, value types, reference types, predefined or user defined types. However, before assigning values, it needs type conversion.

When a value type is converted to object type, it is called **boxing** and on the other hand, when an object type is converted to a value type, it is called **unboxing**.

```
object obj;
obj = 100; // this is boxing
```

Dynamic Type

You can store any type of value in the dynamic data type variable. Type checking for these types of variables takes place at run-time. Syntax for declaring a dynamic type is:

```
dynamic <variable_name> = value;
```

For example,
dynamic d = 20;

Dynamic types are similar to object types except that type checking for object type variables takes place at compile time, whereas that for the dynamic type variables takes place at run time.

String Type

The **String Type** allows you to assign any string values to a variable. It is derived from object type. The value for a string type can be assigned using string literals in two forms: quoted and @quoted.
For example,

String str = "Component based Technology";

As well as escaping quotes with backslashes we can use double-quoting in a @-prefixed string:

string msg = @"I want to learn ""c#""";

Pointer Type

Pointer type variables store the memory address of another type. Pointers in C# have the same capabilities as the pointers in C or C++. Syntax for declaring a pointer type is:

type* identifier;

For example,
char* cptr;
int* iptr;

A **data type** defines the information that can be stored in a variable, the size of needed memory and the operations that can be performed with the variable. For example, to store an integer value (a whole number) in a variable, use the **int** keyword:

int myAge;

The code above declares a variable named **myAge** of type **integer**.

A line of code that completes an action is called a statement. Each statement in C# must end with a **semicolon**.You can assign the value of a variable when you declare it:

int myAge = 18;

or later in your code:

int myAge;
myAge = 18;

Operators

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

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Bitwise Operators
5. Assignment Operators
6. Miscellaneous Operators

Arithmetic Operators

Following table shows all the arithmetic operators supported by C#. Assume variable **A** holds 10 and variable **B** holds 20 then:

Operator	Description	Example
+	Adds two operands	A + B = 30
-	Subtracts second operand from the first	A - B = -10
*	Multiplies both operands	A * B = 200
/	Divides numerator by de-numerator	B / A = 2
%	Modulus Operator and remainder of after an integer division	B % A = 0
++	Increment operator increases integer value by one	A++ = 11
--	Decrement operator decreases integer value by one	A-- = 9

Relational Operators

Following table shows all the relational operators supported by C#. Assume variable **A** =10 and variable **B** = 20, then:

Operator	Description	Example
== Equal to	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!= Not equal to	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
> Greater than	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
< Less than	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>= Greater than or equal to	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or	(A <= B) is true.

Less than or equal to	equal to the value of right operand, if yes then condition becomes true.	

Logical Operators

Following table shows all the logical operators supported by C#. Assume variable **A** holds Boolean value true and variable **B** holds Boolean value false, then:

Operator	Description	Example
&& (AND)	Called Logical AND operator. If both the operands are non zero then condition becomes true.	(A && B) is false.
|| (OR)	Called Logical OR Operator. If any of the two operands is non zero then condition becomes true.	(A || B) is true.
! (NOT)	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true

Bitwise Operators

Bitwise operator works on bits and perform bit by bit operation. The bit wise operators are:
& - bitwise AND, | - bitwise OR, ^ - bitwise EXOR

p	q	p & q (bitwise AND)	p | q (bitwise OR)	p ^ q (bitwise EXOR)
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Assume if A = 60; and B = 13, then in the binary format :
A = 0011 1100
B = 0000 1101

A&B = 0000 1100 (12 in decimal)
A|B = 0011 1101 (61 in decimal)
A^B = 0011 0001 (49 in decimal)
~A = 1100 0011 (195 in decimal)
The Bitwise operators supported by C# are listed in the following table. Assume variable A holds 60 and variable B holds 13, then:

The other bitwise operators are:

~ Compliment (NOT)
<< Left shift operator The operand value is moved left by the number of bits specified by the right operand
>> Right shift. The operand value is moved right by the number of bits specified by the right operand

If A =60, in binary it is 0011 1100.
A << 1 = 0111 1000 (120 in decimal) . Left shift is equivalent to multiplying a number by 2
A >> 2 = 0001 1110 (30 in decimal). Right shift is equivalent to dividing a number by 2

Assignment Operators
There are following assignment operators supported by C#:

Operator	Description
=	Simple assignment operator, Assigns values from right side operands to left side operand
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand
<<=	Left shift AND assignment operator
>>=	Right shift AND assignment operator
&=	Bitwise AND assignment operator
^=	bitwise exclusive OR and assignment operator
|=	bitwise inclusive OR and assignment operator

Miscellaneous Operators

Operator	Description	Example
sizeof()	Returns the size of a data type.	sizeof(int), returns 4.
typeof()	Returns the type of a class.	typeof(StreamReader);
&	Returns the address of an variable.	&a; returns actual address of the variable.
*	Pointer to a variable.	*a; creates pointer named ‘a’ to a variable.
? :	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y
is	Determines whether an object is of a certain type.	If(Ford is Car) // checks if Ford is an object of the Car class.
as	Cast without raising an exception if the cast fails.	Object obj = new StringReader("Hello"); StringReader r = obj as StringReader;

Operator precedence

Operator precedence determines the grouping of terms in an expression. This affects evaluation of an expression. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator.
For example x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has higher precedence than +, so the first evaluation takes place for 3*2 and then 7 is added into it.
Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators are evaluated first.

Highest to lowest

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to righ
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Example :

```
If  int a = 20;
  int b = 10;
  int c = 15;
  int d = 5;
```

```
Value of (a + b) * c / d is : 90
Value of ((a + b) * c) / d is : 90
Value of (a + b) * (c / d) is : 90
Value of a + (b * c) / d is : 50
```

Decision making

Decision making structures requires the programmer to specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false. Following is the general from of a typical decision making structure found in most of the programming languages:

C# provides following types of decision making statements.

Statement	Description
if statement	An if statement consists of a boolean expression followed by one or more statements.
if...else statement	An if statement can be followed by an optional else statement, which executes when the boolean expression is false.
nested if statements	You can use one if or else if statement inside another if or else if statement(s).
witch statement	A switch statement allows a variable to be tested for equality against a list of values.
nested switch statements	You can use one switch statement inside another switch statement(s)

if Statement

An **if** statement consists of a boolean expression followed by one or more statements.

Syntax

```
if(boolean_expression)
{
/* statement(s) will execute if the boolean expression is true */
}
```

If the boolean expression evaluates to **true**, then the block of code inside the if statement is executed. If boolean expression evaluates to **false**, then the first set of code after the end of the if statement (after the closing curly brace) is executed.

if...else Statement

An **if** statement can be followed by an optional **else** statement, which executes when the boolean expression is false.

Syntax

```
if(boolean_expression)
{
/* statement(s) will execute if the boolean expression is true */
}
else
{
/* statement(s) will execute if the boolean expression is false */
}
```

If the boolean expression evaluates to **true**, then the **if block** of code is executed, otherwise **else block** of code is executed

if...else if...else Statement

An **if** statement can be followed by an optional **else if...else** statement, which is very useful to test various conditions using single if...else if statement.

When using if, else if and else statements there are few points to keep in mind.

- An if can have zero or one else's and it must come after any else if's.
- An if can have zero to many else if's and they must come before the else.
- Once an else if succeeds, none of the remaining else if's or else's will be tested.

Syntax

```
if(boolean_expression 1)
{
/* Executes when the boolean expression 1 is true */
}
else if( boolean_expression 2)
{
/* Executes when the boolean expression 2 is true */
}
else if( boolean_expression 3)
{
/* Executes when the boolean expression 3 is true */
}
else
{
/* executes when the none of the above condition is true */
}
```


Nested if Statements

It is always legal in C# to **nest** if-else statements, which means you can use one if or else if statement inside another if or else if statement(s).

Syntax

```
if( boolean_expression 1)
{
/* Executes when the boolean expression 1 is true */
if(boolean_expression 2)
{
/* Executes when the boolean expression 2 is true */
}
}
```

Switch Statement

A **switch** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each **switch case**.

Syntax

```
switch(expression){
case constant-expression :
statement(s);
break; /* optional */
case constant-expression :
statement(s);
```

```
break; /* optional */
/* you can have any number of case statements */
default : /* Optional */
statement(s);
}
```

The following rules apply to a **switch** statement:

- The **expression** used in a **switch** statement must have an integral or enumerated type, or be of a class type in which the class has a single conversion function to an integral or enumerated type.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The **constant-expression** for a case must be the same data type as the variable in the switch, and it must be a constant or a literal. When the variable being switched on is equal to a case, the statements following that case will execute until a **break** statement is reached.
- When a **break** statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement. Not every case needs to contain a **break**. If no **break** appears, the flow of control will *fall through* to subsequent cases until a break is reached.
- A **switch** statement can have an optional **default** case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No **break** is needed in the default case.

The ? : Operator

The **conditional operator ? :** can be used to replace **if...else** statements. It has the following general form:

```
Exp1 ? Exp2 : Exp3;
```

Where Exp1, Exp2, and Exp3 are expressions. Notice the use and placement of the colon. The value of a ? expression is determined as follows: Exp1 is evaluated. If it is true, then Exp2 is evaluated and becomes the value of the entire ? expression. If Exp1 is false, then Exp3 is evaluated and its value becomes the value of the expression.

2.2 Loop Statements

There may be a situation, when you need to execute a block of code several number of times. In general, the statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths. A loop statement allows us to execute a statement or a group of statements multiple times and following is the general from of a loop statement in most of the programming languages:

C# provides following types of loop to handle looping requirements.

Loop Type	Description
while loop	It repeats a statement or a group of statements while a given condition is true. It tests the condition before executing the loop body.
for loop	It executes a sequence of statements multiple times and abbreviates the code that manages the loop
do...while loop	It is similar to a while statement, except that it tests the condition at the end of the loop body
nested loops	You can use one or more loops inside any another while, for or do..while loop.
For..each	

While Loop

A **while** loop statement in C# repeatedly executes a target statement as long as a given condition is true.

Syntax

```
while(condition)
{
statement(s);
}
```

Here, **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any non-zero value. The loop iterates while the condition is true. When the condition becomes false, program control passes to the line immediately following the loop. When the condition is tested and the result is false, the loop body is skipped and the first statement after the while loop is executed.

For Loop

A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

Syntax

```
for ( init; condition; increment )
{
statement(s);
}
```

Example:

```
using System;
```

```
namespace Loops
    {
    class Program
        {
        static void Main(string[] args)
        {

            for (int a = 10; a < 20; a = a + 1)
            {
            Console.WriteLine("value of a: {0}", a);
            }
            Console.ReadLine();
        }
        }
    }
```

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

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```


Do...While Loop

Unlike **for** and **while** loops, which test the loop condition at the start of the loop, the **do...while** loop checks its condition at the end of the loop. A **do...while** loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time. It is alos known as at least once loop.

Syntax

```
do
{
statement(s);
}while( condition );
```

Notice that the conditional expression appears at the end of the loop, so the statement(s) in the loop execute once before the condition is tested. If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop execute again. This process repeats until the given condition becomes false.

foreach

foreach loop is a different kind of looping constructs in C# programming that doesn't includes initialization, termination and increment/decrement characteristics. It uses collection to take value one by one and then processes them.

With a foreach loop, we evaluate each element individually. An index is not needed. With no indexes, loops are easier to write and programs are simpler.

syntax:

```
foreach (string name in arr)
 {

 }
```

Where, **name** is a string variable that takes value from collection as arr and then processes them in the body area.
foreach-loop is the easiest, least error-prone loop. It is preferred in many program contexts. But we lose some flexibility with it.

Index: Foreach uses no integer index. Instead, it is used on a collection—it returns each element in order.This is called enumeration. We eliminate errors caused by incorrect index handling.

Strings: We use foreach, on a string array, to loop through the elements in the array.

```
using System;

class Program

{

Staic void main()

{

string[] pets ={“dog”,”cat”,”bird”};

foreach (string p in pets)

      {

      Console.WriteLine(p);

      }

}

}
```


Output:
dog
cat
bird

Nested Loops
C# allows to use one loop inside another loop. Following section shows few examples to illustrate the concept.

Syntax
The syntax for a **nested for loop** statement in C# is as follows:

```
for ( init; condition; increment )
{
for ( init; condition; increment )
{
statement(s);
}
statement(s);
}
```

The syntax for a **nested while loop** statement in C# is as follows:

```
while(condition)
{
while(condition)
{
statement(s);
}
statement(s);
}
```

The syntax for a **nested do...while loop** statement in C# is as follows:


```
do
{
statement(s);
do
{
statement(s);
}while( condition );
}while( condition );
```

Note: You can put any type of loop inside of any other type of loop. For example a for loop can be inside a while loop or vice versa.

Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed. C# provides the following control statements.

Control Statement	Description
break	Terminates the loop or switch statement and transfers execution to the statement immediately following the loop or switch.
continue	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

Break Statement
The break statement in C# has following two usage:

1. When the break statement is encountered inside a loop, the loop is immediately terminated and program control resumes at the next statement following the loop.

2. It can be used to terminate a case in the switch statement. If you are using nested loops (i.e., one loop inside another loop), the break statement will stop the execution of the innermost loop and start executing the next line of code after the block.
Syntax:
break;

Continue Statement
The continue statement in C# works somewhat like the break statement. Instead of forcing termination, however, continue forces the next iteration of the loop to take place, skipping any code in between.

For the for loop, continue statement causes the conditional test and increment portions of the loop to execute. For the while and do...while loops, continue statement causes the program control passes to the conditional tests

2.3 Types:

Value Type

Value type variables can be assigned a value directly. They are derived from the class **System.ValueType**. The value types directly contain data. Some examples are **int, char, and float**, which stores numbers, alphabets, and floating point numbers, respectively. When you declare an **int** type, the system allocates memory to store the value.

The following table lists the available value types in C# 2010:

Type	**Represents**	**Range**	**Default Value**
bool	Boolean value	True or False	False
byte	8-bit unsigned integer	0 to 255	0
char	16-bit Unicode character	U +0000 to U +ffff	'\0'
decimal	128-bit precise decimal values with 28-29 significant digits	(-7.9 x 1028 to 7.9 x 1028) / 100 to 28	0.0M
double	64-bit double-precision floating point type	(+/-)5.0 x 10-324 to (+/-)1.7 x 10308	0.0D
float	32-bit single-precision floating point type	-3.4 x 1038 to + 3.4 x 1038	0.0F
int	32-bit signed integer type	-2,147,483,648 to 2,147,483,647	0
long	64-bit signed integer type	-923,372,036,854,775,808 to 9,223,372,036,854,775,807	0L
sbyte	8-bit signed integer type	-128 to 127	

To get the exact size of a type or a variable on a particular platform, you can use the **sizeof** method. The expression *sizeof(type)* yields the storage size of the object or type in bytes.

Built-in Data Types

There are a number of built-in data types in C#. The most common are:

Type	Description
int	integer
float	floating point number
double	double-precision version of float
char	A single character
bool	Boolean that can have only one of two values: True or False.
string	sequence of characters.

The statements below use C# data types:

```
int x = 42;
double pi = 3.14;
char y = 'Z';
bool isOnline = true;
string subName = "Component based ";
```

Note that **char** values are assigned using single quotes and **string** values require double quotes

Structures:

In C#, a structure is a value type data type. It helps you to make a single variable hold related data of various data types. The **struct** keyword is used for creating a structure. Structures are used to represent a record. Suppose you want to keep track of your books in a library. You might want to track the following attributes about each book:

- Title
- Author
- Subject
- Book ID

Defining a Structure

To define a structure, you must use the struct statement. The struct statement defines a new data type, with more than one member for your program. For example, here is the way you can declare the Book structure:

```
struct Books
{
  public string title;
  public string author;
  public string subject;
  public int book_id;
};
```

The following program shows the use of the structure:

```
using System;
struct Books
{
  private string title;
  private string author;
  private string subject;
  private int book_id;
  public void getValues(string t, string a, string s, int id)
  {
    title = t;
    author = a;
    subject = s;
    book_id = id;
  }
  public void display()
  {
    Console.WriteLine("Title : {0}", title);
    Console.WriteLine("Author : {0}", author);
    Console.WriteLine("Subject : {0}", subject);
    Console.WriteLine("Book_id :{0}", book_id);
  }

};

public class testStructure
{
  public static void Main(string[] args)
  {

    Books Book1 = new Books();   /* Declare Book1 of type Book */
    Books Book2 = new Books();   /* Declare Book2 of type Book */

    /* book 1 specification */
    Book1.getValues("C Programming",    "Nuha Ali", "C Programming Tutorial",6495407);

    /* book 2 specification */
    Book2.getValues("Telecom Billing", "Zara Ali", "Telecom Billing Tutorial", 6495700);

    /* print Book1 info */
    Book1.display();

    /* print Book2 info */
    Book2.display();

    Console.ReadKey();
  }
}
```

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

```
Title : C Programming
```

Author : Nuha Ali
Subject : C Programming Tutorial
Book_id : 6495407
Title : Telecom Billing
Author : Zara Ali
Subject : Telecom Billing Tutorial
Book_id : 6495700

Features of C# Structures

Structures in C# are quite different from that in traditional C or C++. The C# structures have the following features:

- Structures can have methods, fields, indexers, properties, operator methods, and events.
- Structures can have defined constructors, but not destructors. However, you cannot define a default constructor for a structure. The default constructor is automatically defined and cannot be changed.
- Unlike classes, structures cannot inherit other structures or classes.
- Structures cannot be used as a base for other structures or classes.
- A structure can implement one or more interfaces.
- Structure members cannot be specified as abstract, virtual, or protected.
- When you create a struct object using the **New** operator, it gets created and the appropriate constructor is called. Unlike classes, structs can be instantiated without using the New operator.
- If the New operator is not used, the fields remain unassigned and the object cannot be used until all the fields are initialized.

Class versus Structure

Classes and Structures have the following basic differences:

- classes are reference types and structs are value types
- structures do not support inheritance
- structures cannot have default constructor

Enumeration

An enumeration is a set of named integer constants. An enumerated type is declared using the **enum** keyword. C# enumerations are value data type. In other words, enumeration contains its own values and cannot inherit or cannot pass inheritance.

The general syntax for declaring an enumeration is:

```
enum <enum_name>
{
  enumeration list
};
```

Where,

- The *enum_name* specifies the enumeration type name.
- The *enumeration list* is a comma-separated list of identifiers.

Each of the symbols in the enumeration list stands for an integer value, one greater than the symbol that precedes it. By default, the value of the first enumeration symbol is 0. For example:

enum Days { Sun, Mon, tue, Wed, thu, Fri, Sat };

The following example demonstrates use of enum variable:

```
using System;
namespace EnumApplication
{
  class EnumProgram
  {
    enum Days { Sun, Mon, tue, Wed, thu, Fri, Sat };

    static void Main(string[] args)
    {
      int WeekdayStart = (int)Days.Mon;
      int WeekdayEnd = (int)Days.Fri;
      Console.WriteLine("Monday: {0}", WeekdayStart);
      Console.WriteLine("Friday: {0}", WeekdayEnd);
      Console.ReadKey();
    }
  }
}
```


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

Monday: 1
Friday: 5

Reference data types:

The reference types do not contain the actual data stored in a variable, but they contain a reference to the variables. In other words, they refer to a memory location. Using multiple variables, the reference types can refer to a memory location. If the data in the memory location is changed by one of the variables, the other variable automatically reflects this change in value. Example of **built-in** reference types are: **object**, **dynamic,** and **string**

The Reference type variable is such type of variable in C# that holds the reference of memory address instead of value.

class, interface, delegate, array are the reference type. When you create object of particular class with new keyword, a space is created in the managed **heap** that holds the reference of classes.

Arrays

An array stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type stored at contiguous memory locations.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index. All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

Declaring Arrays

To declare an array in C#, you can use the following syntax:

datatype[] arrayName;

where,

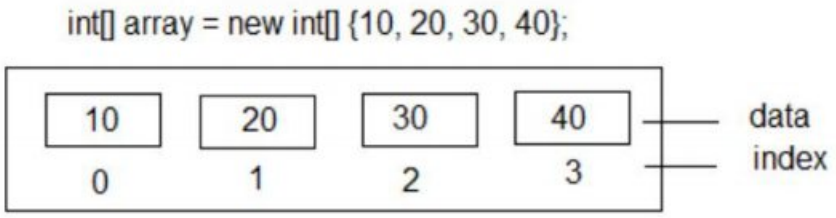
- *datatype* is used to specify the type of elements in the array.
- *[ ]* specifies the rank of the array. The rank specifies the size of the array.
- *arrayName* specifies the name of the array.

For example,

double[ ] balance;

Initializing an Array

Declaring an array does not initialize the array in the memory. When the array variable is initialized, you can assign values to the array. Array is a reference type, so you need to use the **new** keyword to create an instance of the array. For example,



In the above example array is the variable of type integer and it is initialized at the point of declaration.

An array can also be declared like:

double[] balance = new double[10];

Assigning Values to an Array

You can assign values to individual array elements, by using the index number, like:

```
double[] balance = new double[10];
balance[0] = 4500.0;
```

You can assign values to the array at the time of declaration as :

```
double[] balance = { 2340.0, 4523.69, 3421.0};
```

You can also create and initialize an array as :

```
int [] marks = new int[5] { 99, 98, 92, 97, 95};
```

You may also omit the size of the array as:

```
int [] marks = new int[] { 99, 98, 92, 97, 95};
```

You can copy an array variable into another target array variable. In such case, both the target and source point to the same memory location:

```
int [] marks = new int[] { 99, 98, 92, 97, 95};
int[] score = marks;
```

When you create an array, C# compiler implicitly initializes each array element to a default value depending on the array type. For example, for an int array all elements are initialized to 0.

Accessing Array Elements

An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example, double salary = balance[9];

The following example demonstrates the above-mentioned concepts declaration, assignment, and accessing arrays:

```
using System;
namespace ArrayApplication
{
class MyArray
{
static void Main(string[] args)
{
int [] n = new int[10];              /* n is an array of 10 integers */
int i,j;
/* initialize elements of array n */

for ( i = 0; i < 10; i++ )
{
n[ i ] = i + 100;
}
/* output each array element's value */
for (j = 0; j < 10; j++ )
```

```
{
Console.WriteLine("Element[{0}] = {1}", j, n[j]);
}
Console.ReadKey();
}
}
}
```

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

```
Element[0] = 100
Element[1] = 101
Element[2] = 102
Element[3] = 103
Element[4] = 104
Element[5] = 105
Element[6] = 106
Element[7] = 107
Element[8] = 108
Element[9] = 109
```

C# Arrays

There are following few important concepts related to array which should be clear to a C# programmer:

Concept	Description
Multi-dimensional arrays	C# supports multidimensional arrays. The simplest form of the multidimensional array is the two-dimensional array.
Jagged arrays	C# supports multidimensional arrays, which are arrays of arrays.
Passing arrays to functions	You can pass to the function a pointer to an array by specifying the array's name without an index.
Param arrays	This is used for passing unknown number of parameters to a function.
The Array Class	Defined in System namespace, it is the base class to all arrays, and provides various properties and methods for working with arrays.

Multidimensional Arrays

C# allows multidimensional arrays. Multi-dimensional arrays are also called rectangular array. You can declare a 2-dimensional array of strings as: string [,] names; or, a 3-dimensional array of int variables as: int [, ,] m;

Two-Dimensional Arrays

The simplest form of the multidimensional array is the 2-dimensional array. A 2-dimensional array is a list of one-dimensional arrays. A 2-dimensional array can be thought of as a table, which has x number of rows and y number of columns. Following is a 2-dimensional array, which contains 3 rows and 4 columns: Thus, every element in the array a is identified by an element name of the form a[i, j], where is the name of the array, and i and j are the subscripts that uniquely identify each element in array

Initializing Two-Dimensional Arrays:
Multidimensional arrays may be initialized by specifying bracketed values for each row. The following array is with 3 rows and each row has 4 columns.

int [,] a = int [3,4] = { {0, 1, 2, 3} , {4, 5, 6, 7} , {8, 9, 10, 11} };

Accessing Two-Dimensional Array Elements
An element in 2-dimensional array is accessed by using the subscripts. That is, row index and column index of the array. For example,

int val = a[2,3];

The above statement takes 4th element from the 3rd row of the array. You can verify it in the above diagram. Let us check the program to handle a two dimensional array:

```
using System;
namespace ArrayApplication
{
class MyArray
{
static void Main(string[] args)
{
/* an array with 5 rows and 2 columns*/
int[,] a = new int[5, 2] {{0,0}, {1,2}, {2,4}, {3,6}, {4,8} };
int i, j;
/* output each array element's value */
for (i = 0; i < 5; i++)
{
for (j = 0; j < 2; j++)
{
Console.WriteLine("a[{0},{1}] = {2}", i, j, a[i,j]);
}
}
Console.ReadKey();
}
}
}
```


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

```
a[0,0]: 0
a[0,1]: 0
a[1,0]: 1
a[1,1]: 2
a[2,0]: 2
a[2,1]: 4
a[3,0]: 3
a[3,1]: 6
a[4,0]: 4
a[4,1]: 8
```

Jagged Arrays
A Jagged array is an array of arrays.

You can declare a jagged array named *scores* of type **int** as:int [][] scores; Declaring an array, does not create the array in memory. To create the above array:

```
int[][] scores = new int[5][];
for (int i = 0; i < scores.Length; i++)
{
scores[i] = new int[4];
}
```

You can initialize a jagged array as:

```
int[][] scores = new int[2][]{new int[]{92,93,94},new int[]{85,66,87,88}};
```

Where, scores is an array of two arrays of integers - scores[0] is an array of 3 integers and scores[1] is an array of 4 integers.

Dynamic Array

A Dynamic Array defines a size of the array at runtime, but then makes room for new elements in the array during execution.

Static arrays have the disadvantage that if you have not used a full array then it will always use the same size as was defined during its declaration. We usually need to have an array that we would not know the values of or how many of them exist. For that we can use a dynamic array.

Declaration and Initialization

List<data type> name= new List<data type>();
e.g;
List<int> list=new List<int>();

Example

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace DynamicArray
{
  class Program
  {
     static void Main(string[] args)
    {
        List<int> list=new List<int>();
        list.Add(1);
        list.Add(10);
        list.Add(4);
        list.Add(0);
        int size = list.Count;
        for (int i = 0; i < list.Count; i++)
        Console.WriteLine(list[i]);
        list.Sort();
        Console.WriteLine("Sorted list values");
```

```
                    for (int i = 0; i < list.Count; i++)
                    Console.WriteLine(list[i]);
                    Console.ReadKey();
                }
            }
        }
```

Output:
1
10
4
0
Sorted list values
0
1
4
10

Array Class

The Array class is the base class for all the arrays in C#. It is defined in the System namespace. The Array class provides various properties and methods to work with arrays.

Properties of the Array Class

The following table describes some of the most commonly used properties of the Array class:

Sl. No.	Property
1	**IsFixedSize** Gets a value indicating whether the Array has a fixed size.
2	**IsReadOnly** Gets a value indicating whether the Array is read-only.
3	**Length** Gets a 32-bit integer that represents the total number of elements in all the dimensions of the Array.
4	**LongLength** Gets a 64-bit integer that represents the total number of elements in all the dimensions of the Array.
5	**Rank** Gets the rank (number of dimensions) of the Array.

Methods of the Array Class

The following table describes some of the most commonly used methods of the Array class:

Sl. No	Methods
1	**Clear()** Sets a range of elements in the Array to zero, to false, or to null, depending on the element type.
2	**Copy(Array, Array, length)** Copies a range of elements from an Array starting at the first element

	and pastes them into another Array starting at the first element. The length is specified as a 32-bit integer.
3	**CopyTo(Array, index)** Copies all the elements of the current one-dimensional Array to the specified one-dimensional Array starting at the specified destination Array index. The index is specified as a 32-bit integer.
4	**GetLength** Gets a 32-bit integer that represents the number of elements in the specified dimension of the Array.
5	**GetLongLength** Gets a 64-bit integer that represents the number of elements in the specified dimension of the Array.
6	**GetLowerBound** Gets the lower bound of the specified dimension in the Array.
7	**GetType** Gets the Type of the current instance. (Inherited from Object.)
8	**GetUpperBound** Gets the upper bound of the specified dimension in the Array.
9	**GetValue(Int32)** Gets the value at the specified position in the one-dimensional Array. The index is specified as a 32-bit integer.
10	**IndexOf(Array, Object)** Searches for the specified object and returns the index of the first occurrence within the entire one-dimensional Array.
11	**Reverse(Array)** Reverses the sequence of the elements in the entire one-dimensional Array.
12	**SetValue(Object, Index)** Sets a value to the element at the specified position in the one dimensional Array. The index is specified as a 32-bit integer
13	**Sort(Array)** Sorts the elements in an entire one-dimensional Array
14	**ToString()** Returns a string that represents the current object

2.4 Classes and objects

Class

A class is the fundamental building block of code when creating object-oriented software. A class describes in abstract, all of the characteristics and behaviour of a type of object.

When you define a class, you define a blueprint for a data type. This does not actually define any data, but it does define what the class name means. That is, what an object of the class consists of and what operations can be performed on that object.

Objects are instances of a class. The methods and variables that constitute a class are called members of the class.

You create an instance of this class, called as an object. On this object, you use the defined methods and variables. You can create as many instances of your class as you want.

A class should not be confused with an object. The class is the abstract concept for an object that is created at design-time by the programmer. The objects based upon the class are the concrete instances of the class that occur at run-time. For example, the *Car* class will define that all cars have a make, model and colour. Only at run-time will an object be instantiated as a Swift.

Defining a Class (Creating and using your own class)

A class definition starts with the keyword class followed by the class name; and the class body enclosed by a pair of curly braces. Following is the general form of a class definition:

```
<access specifier> class class_name
{
// member variables
<access specifier> <data type> variable1;
<access specifier> <data type> variable2;
...
<access specifier> <data type> variableN;

// member methods
<access specifier> <return type> method1(parameter_list)
{
// method body
}
<access specifier> <return type> method2(parameter_list)
{
// method body
}
...
<access specifier> <return type> methodN(parameter_list)
{
// method body
}

}
```


Note:

- Access specifiers specify the access rules for the members as well as the class itself. If not mentioned, then the default access specifier for a class type is **internal**. Default access for the members is **private**.
- Data type specifies the type of variable, and return type specifies the data type of the data the method returns, if any.

- To access the class members, you use the dot (.) operator.
- The dot operator links the name of an object with the name of a member.

The following example illustrates the concepts of object oriented programming:

```
using System;
namespace ConsoleApplication1
{
class Car
   {
      private string color;

      public Car(string color)  // constructor
      {
         this.color = color;
      }

      public string Describe() // method
      {
         return "This car is " + Color;
      }

      public string Color  // properties
      {
         get { return color; }
         set { color = value; }
      }
   }

   class Program
   {
      static void Main(string[] args)
      {
         Car car;

         car = new Car("Red"); // Instantiate an object
         Console.WriteLine(car.Describe());

         car = new Car("Green");
         Console.WriteLine(car.Describe());

         Console.ReadLine();

      }
   }


}
```

Member Functions and Encapsulation

A member function of a class is a function that has its definition or its prototype within the class definition similar to any other variable. It operates on any object of the class of which it is a member, and has access to all the members of a class for that object. Member variables are the attributes of an object and they are declared as private to implement encapsulation. These variables can only be accessed using the public member functions.

C# Constructors

A class **constructor** is a special member function of a class that is executed whenever we create new objects of that class. A constructor has exactly the same name as that of the class and it does not have any return type. Following example explains the concept of constructor:

A **default constructor** does not have any parameter but if you need, a constructor can have parameters. Such constructors are called **parameterized constructors**. This technique helps you to assign initial value to an object at the time of its creation

C# Destructors

A **destructor** is a special member function of a class that is executed whenever an object of its class goes out of scope. A **destructor** has exactly the same name as that of the class with a prefixed tilde (~) and it can neither return a value nor can it take any parameters. Destructor can be very useful for releasing memory resources before exiting the program. Destructors cannot be inherited or overloaded.

Abstract Classes and Abstract Methods

The purpose of abstract class is to provide default functionality to its sub classes. When a method is declared as abstract in the base class then every derived class of that class must provide its own definition for that method.

An abstract class can also contain methods with complete implementation, besides abstract methods. When a class contains at least one abstract method, then the class must be declared as **abstract class**.

It is **mandatory** to override abstract method in the derived class.When a class is declared as abstract class, then it is not possible to create an instance for that class. But it can be used as a parameter in a method.

Example:
The following example creates three classes shape, circle and rectangle where circle and rectangle are inherited from the class shape and overrides the methods Area() and Circumference() that are declared as abstract in Shape class and as Shape class contains abstract methods it is declared as abstract class.

```
using System;
namespace Demo
{
   //Abstract class
   abstract class Shape1
   {
      protected float R, L, B;
      //Abstract methods can have only declarations
      public abstract float Area();
      public abstract float Circumference();
   }

   class Rectangle1 : Shape1   // inheritance
   {
      public void GetLB()
      {
         Console.Write("Enter  Length  :  ");
         L = float.Parse(Console.ReadLine());
         Console.Write("Enter Breadth : ");
         B = float.Parse(Console.ReadLine());
      }
      public override float Area()
      {
         return L * B;
      }
      public override float Circumference()
      {
         return 2 * (L + B);
      }
   }


   class Circle1 : Shape1        //inheritance
   {
      public void GetRadius()
      {
         Console.Write("Enter  Radius  :  ");
         R = float.Parse(Console.ReadLine());
      }
      public override float Area()
      {
         return 3.14F * R * R;
      }
      public override float Circumference()
```

```
        {
            return 2 * 3.14F * R;
        }
    }
    class MainClass
    {
        public static void Calculate(Shape1 S)
        {

            Console.WriteLine("Area : {0}", S.Area());
            Console.WriteLine("Circumference : {0}", S.Circumference());

        }
        static void Main()
        {

            Rectangle1 R = new Rectangle1();
            R.GetLB();
            Calculate(R); // Calculate is the method of Main class
            Console.WriteLine();
            Circle1 C = new Circle1();
            C.GetRadius();
            Calculate(C);
            Console.Read();

        }
    }
}
```


Output:

```
Enter  Length  :  10
Enter Breadth : 12
Area : 120
Circumference : 44

Enter  Radius  :  5
Area : 78.5
Circumference : 31.4
```

Note

In the above example, method Calculate takes a parameter of type Shape1 from which rectangle1 and circle1 classes are inherited.

A base class type parameter can take derived class object as an argument. Hence the

calculate method can take either rectangle1 or circle1 object as argument and the actual argument in the parameter S will be determined only at runtime and hence this example is an example for runtime polymorphism.

"this" keyword:

The "this" keyword is a special type of reference variable, that is implicitly defined within each constructor and non-static method as a first parameter of the type class in which it is defined.

"this" in C# is used to access the field variables. All the field variables must be accessed using "this" keyword.

It is useful when the field variable and the local variables of a method have same name**.** In the above case local variables overrides field variables and field variables are no more visible inside the method. So this can be used to refer field variables to make visible inside method.

For example, consider the following class written in C#.

```
class Demo
{
 int a=2,b=10;
 public void Get()
  {
  int a=23,b=34;
  Console.WriteLine("a={0} b={1}",this.a,this.b);
  Console.WriteLine("It is the class variable");
  Console.WriteLine("Now the local variables are:");
  Console.WriteLine("a={0} b={1}",a,b);
  }
}
class MainClass
{
 static void Main(string args[])
  {
  Demo d= new Demo();
  d.Get();
  }
}
```


Output:

```
a=2 b=10
It is the class variable
Now the local variables are:
a=23 b=34
```

When we write this.a and this.b, it will refer to the field variables. But when we write only a and b it refers to the local variables of the method Get(). As the field variable and local variables have the same name we have to use the this keyword. We can't access a field variable without this keyword in C#. this always refer to the members of same class.

Review Questions
UNIT - II

Part A (2 marks)

1. List any two developing environment for C# applications.
2. What is the use of using keyword?
3. List the variable types in C#
4. List the types of operators in C#
5. What is the use of is and as operators?
6. List the decision making statements in C#.
7. What is the importance of for..each statement.
8. List the two loop control statements and describe them.
9. What is meant by jagged arrays?
10. What is meant by Param arrays?
11. What is a class?
12. What is an object?
13. What is a Destructor?
14. What is an abstract class?

Part B (3 marks)

1. Justify with any three reasons why C# is a professional language.
2. List the types of applications that can be developed using C#
3. List the important parts of a C# Program and discuss.
4. List any three C# keywords and describe the purpose of each.
5. Write notes on C# variables.
6. What is a pointer type variable? Give an example.
7. Write the syntax of else..if ladder.
8. List the types of loops and briefly describe them.
9. Compare Structure with class.
10. Write notes on: Reference data types.
11. Discuss the importance of dynamic array with an example.
12. List any three properties of Array class and describe.
13. What is an access specifier? Give an example.
14. What is a constructor? List the difference between default and parameterized constructor.
15. Write notes on : this keyword.

Part C (5 marks)

1. Write a simple console application using C# and explain the program line by line.
2. What is meant by operator precedence? Tabulate and explain the category, operators and the order of execution .
3. Write the syntax of switch..case statement and list the rules apply to it.
4. List the data types supported by C# and tabulate their properties.
5. List and explain the features of structure in C#.
6. List the different types of arrays in C# describe them with example.

7. Write a program that implements Dynamic array using List class.
8. List any 10 methods of Array class and discuss their purpose.
9. Describe the procedure of defining a class with the specification o general form.
10. With a suitable example explain the concept of Inheritance in C#

******

UNIT III
WINDOWS APPLICATION USING WINDOW FORMS

Objectives:

- To recognize the elements of a window application
- To know the different advanced elements of GUI applications
- To understand the events and event handling
- To get the basics of MDI applications, menus and dialogs

3.1 Windows Programming

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.

Windows Forms is the name given to a graphical (GUI) class library included as a part of Microsoft .NET Framework, providing a platform to write rich client applications for desktop, laptop, and tablet PCs.

Windows Forms is a framework located in the System.Windows.Forms.dll assembly for building Windows applications in .NET based on a graphical user interface (GUI). Any language that supports the common language runtime (CLR) can use Windows Forms.

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.

Why Windows Forms?

In Microsoft .NET Framework is designed so that all windows are forms, including dialog boxes. Microsoft coined the term Windows form. Now developers using any .NET-supported language have access to the same windowing classes, whether they work with C#, VB, C++, or any other .NET-compliant language. This language independence has been extended to support many more languages, including COBOL.

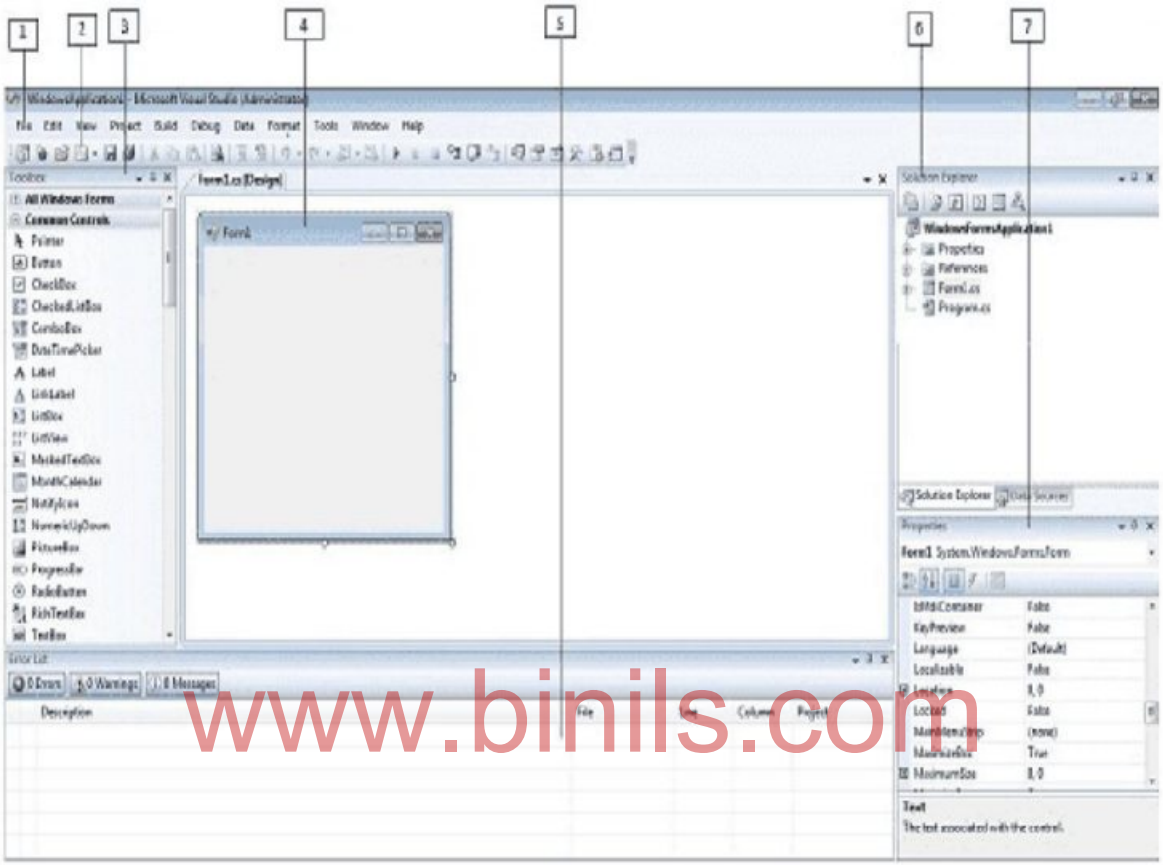
In addition to the preceding, the main benefits of Windows Forms are its ease of use, the standardization of the control hierarchy, and that it allows for rapid application development (RAD). Changing the colors and fonts of controls using MFC or Win32 can be a real headache. The .NET Framework has taken care of most such problems and inconveniences.

In addition, Windows Forms applications provide the following:

- Simple and flexible property support, modeled after
- Common control support, including support for font and color dialogs
- Support for Web Services
- Data-aware controls using ADO.NET
- ActiveX support

- GDI+ (Graphical Device Interface +), a better and richer graphics library, which supports alpha blending, texture brushes, advanced transformations, and rich text
- Metadata support

Fig. 3.1 Visual Studio IDE



1. Menu Bar
2. Standard Toolbar
3. ToolBox
4. Forms Designer
5. Output Window
6. Solution Explorer
7. Properties Window

Writing Your First Windows Application

The most commonly used tool to develop Windows application is Visual Studio .NET (VS.NET) environment, but VS.NET is not a mandatory tool. A Windows Forms application, may be developed without using Visual Studio's integrated development environment (IDE). Simply use any text editor to write your code and save the file with a .cs extension.

Our first Windows application is a simple one that creates a window. To create a Windows-based application, you derive a class from System.Windows.Forms.Form and call the default constructor. The Form class acts as a container for other controls.

First Console Windows Application

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

// Derive your class from the System.Windows.Forms.Form class
public class WinForm : Form
{
   public WinForm()
   { }

   static void Main(string[] args)
   {
      // Create a Form object
      WinForm myFrm = new WinForm();

      // Set the window title
      myFrm.Text = "My First Windows Application";
      // Pass form object
      Application.Run(myFrm);
   }
}
```

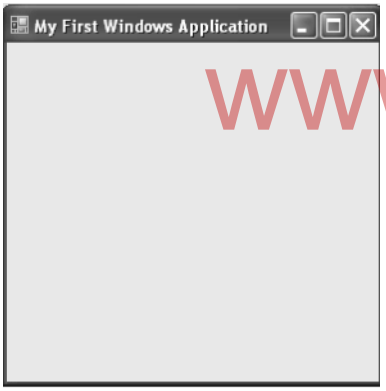


Fig. 3.2 First Windows Application

The window's title is set by the form's Text property. The static method Application.Run creates a standard message loop on the current thread. The program, as Figure illustrates, creates an empty form with the title "My First Windows Application" in the caption bar.

How to create a new project in C# ?

Open your Visual Studio Environment and Click **File->New Project**

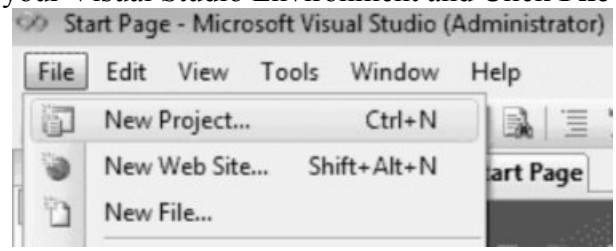


Fig. 3.3 Create new application

Then you will get a **New Project Dialogue Box** asking in which language you want to create a new project.

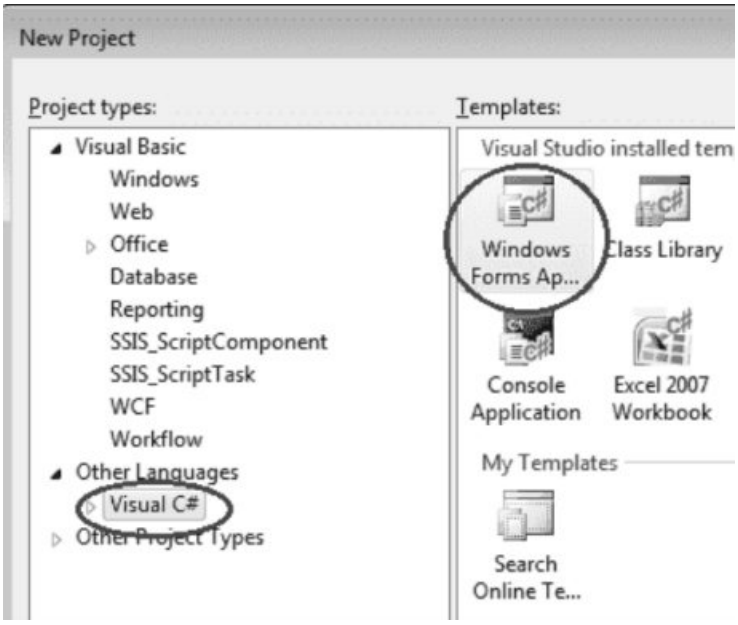


Fig. 3.4 Application type and Language selection

Select Visual C# from the list, then you will get the following screen.

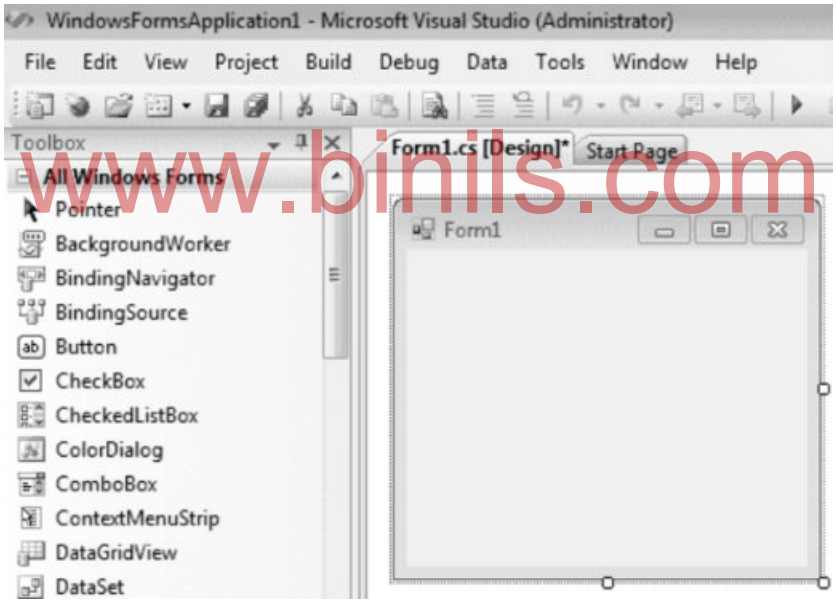


Fig. 3.5 Blank Window application

Now you can add controls in your Form Control.

How to add controls to Form ?

In the left side of the Visual Studio Environment you can see the **Tool Box**. There are lots of controls grouping there in the Tool Box according to their functionalities. Just click the + sign before each group then you can see the controls inside the group. You can select basic controls from **Common Controls** group. You can place the control in your Form by drag and drop the control from your toolbox to Form control.

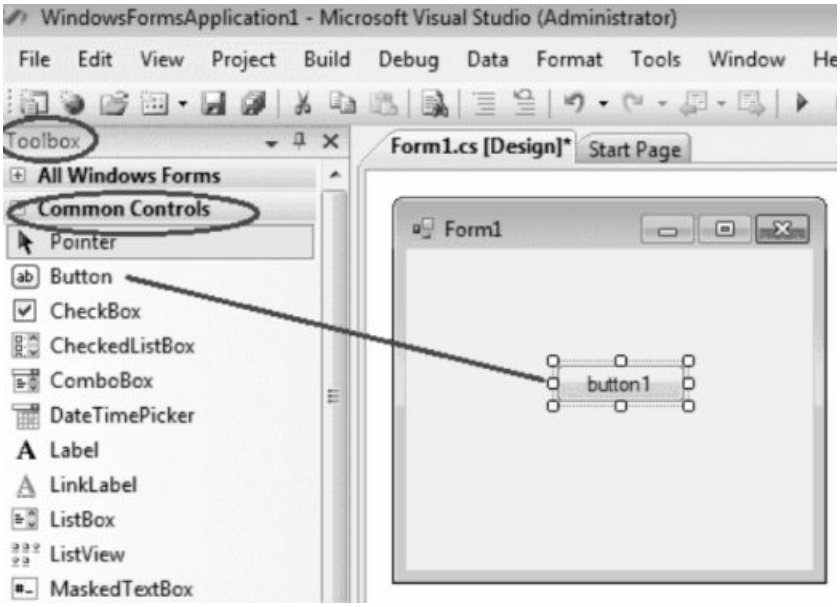


Fig. 3.6 Placing controls on Forms

How to drag and drop controls ?

In the above picture we drag and drop the Button control from Toolbox - Common control to Form control.

Windows Common controls (Toolbox controls)

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.

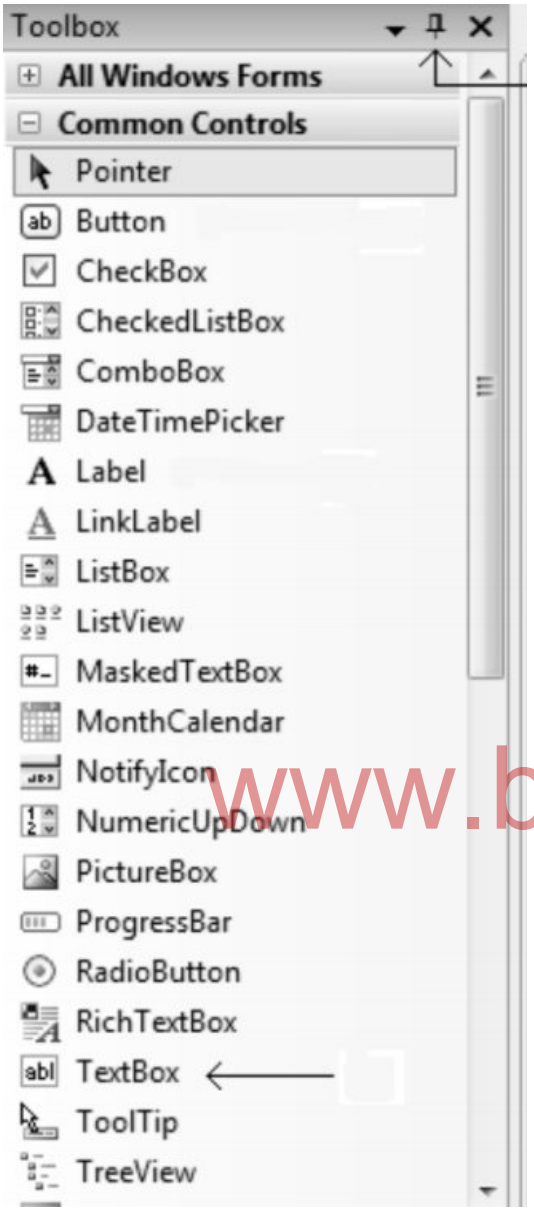
Fig. 3.7 Label

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";
```

Fig. 3.8 Common controls



Toolbox – Common controls

Button
Checkbox
CheckedListBox
ComboBox
DateTimePicker
Label
LinkLabel
ListBox
ListView
MaxkedTextBox
MonthCalendar
NotifyIcon
NumericUpDown
PictureBox
ProgressBar
RafioButton
RichTextBox
TextBox
ToolTip

Button

Windows Forms controls are reusable components that encapsulate user interface functionality and are used in client side Windows applications. A button is a control, which is an interactive component that enables users to communicate with an application. The Button class inherits directly from the ButtonBase class. A Button can be clicked by using the mouse, ENTER key, or SPACEBAR if the button has focus.When you want to change display text of the Button , you can change the Text property of the button.

button1.Text = "Click Here";

TextBox

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.

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 = "C# programming";
```

You can also collect the input value from a TextBox control to a variable like this way

```
string var;
var = textBox1.Text;
```

TextBox Properties

You can set TextBox properties through Property window or through program. You can open Properties window by pressing F4 or right click on a control and select Properties menu item

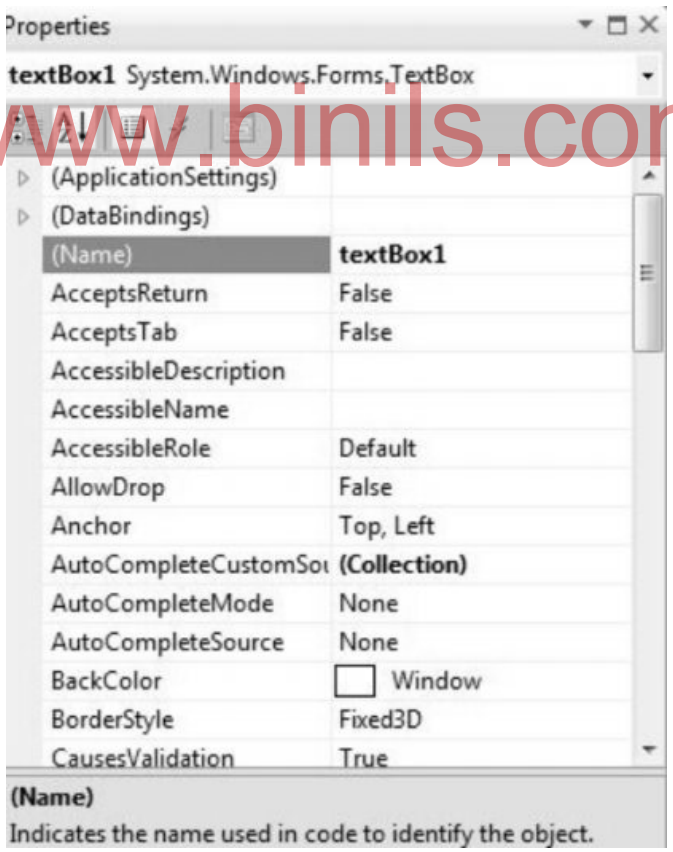


Fig. 3.9 Properties Window

The below code set a textbox width as 250 and height as 50 through source code.

```
textBox1.Width = 250;
textBox1.Height = 50;
```

Background Color and Foreground Color

You can set background color and foreground color through property window and programmatically.

```
textBox1.BackColor = Color.Blue;
textBox1.ForeColor = Color.White;
```

Textbox Maximum Length

Sets the maximum number of characters or words the user can input into the text box control.

```
textBox1.MaxLength = 40;
```

Textbox ReadOnly

When a program wants to prevent a user from changing the text that appears in a text box, the program can set the controls Read-only property is to True.

```
textBox1.ReadOnly = true;
```

Multiline TextBox

You can use the Multiline and ScrollBars properties to enable multiple lines of text to be displayed or entered.

```
textBox1.Multiline = true;
```


Textbox password character

TextBox controls can also be used to accept passwords and other sensitive information. You can use the PasswordChar property to mask characters entered in a single line version of the control

```
textBox1.PasswordChar = '*';
```

The above code set the PasswordChar to * , so when the user enter password then it display only * instead of typed characters.

How to retrieve integer values from textbox ?

```
int i;
i = int.Parse (textBox1.Text);
```

Parse method Converts the string representation of a number to its integer equivalent.

TextBox Events

Keydown event

You can capture which key is pressed by the user using KeyDown event

Ex:

```
private void textBox1_KeyDown(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.Enter)
    {
        MessageBox.Show("You press Enter Key");

    }
    if (e.KeyCode == Keys.CapsLock)
    {
        MessageBox.Show("You press Caps Lock Key");
    }
}
```

TextChanged Event

When user input or setting the Text property to a new value raises the TextChanged event

```
private void textBox1_TextChanged(object sender, EventArgs e)
{
    label1.Text = textBox1.Text;
}
```

ComboBox Control

C# controls are located in the Toolbox of the development environment, and you use them to create objects on a form with a simple series of mouse clicks and dragging motions. A ComboBox displays a text box combined with a ListBox, which enables the user to select items from the list or enter a new value.

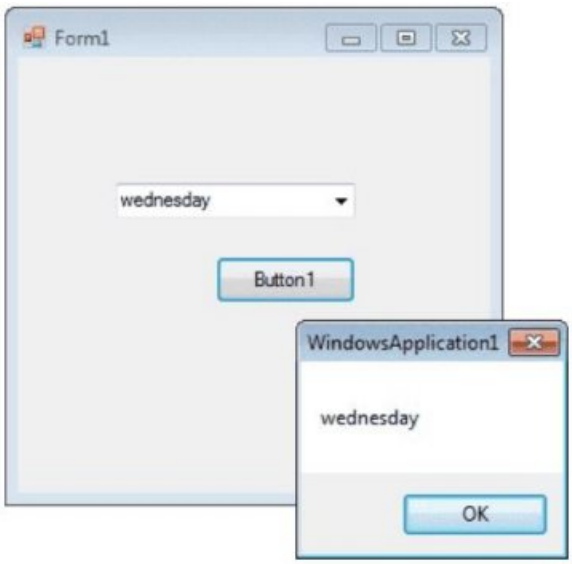


Fig. 3.10 Combo box

The user can type a value in the text field or click the button to display a drop down list. You can add individual objects with the Add method. You can delete items with the Remove method or clear the entire list with the Clear method.

How add a item to combobox

```
comboBox1.Items.Add("Sunday");
comboBox1.Items.Add("Monday");
comboBox1.Items.Add("Tuesday");
```

How to retrieve value from ComboBox

If you want to retrieve the displayed item to a string variable , you can code like this

```
string var;
var = comboBox1.Text;
                                Or
var item = this.comboBox1.GetItemText(this.comboBox1.SelectedItem);
MessageBox.Show(item);
```

How to remove an item from ComboBox

You can remove items from a ComboBox in two ways. You can remove item at a the specified index or giving a specified item by name.

```
comboBox1.Items.RemoveAt(1);
```

The above code will remove the second item from the combobox.

```
comboBox1.Items.Remove("Friday");
```

The above code will remove the item "Friday" from the combobox.

ListBox Control

The ListBox control enables you to display a list of items to the user that the user can select by clicking.

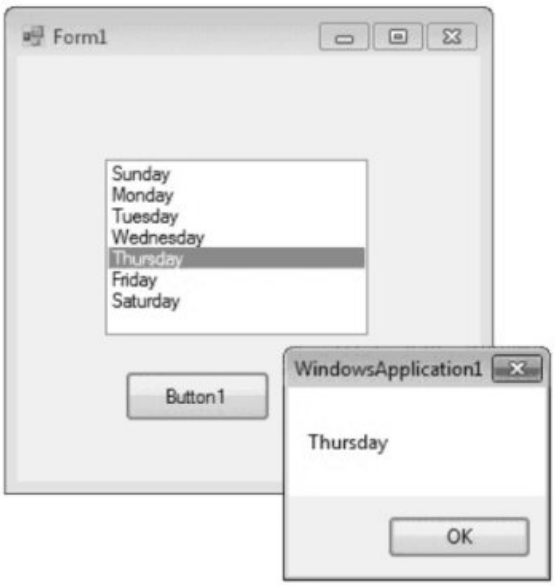


Fig. 3.11 List Box

In addition to display and selection functionality, the ListBox also provides features that enable you to efficiently add items to the ListBox and to find text within the items of the list. You can use the Add or Insert method to add items to a list box. The Add method adds new items at the end of an unsorted list box.

```
listBox1.Items.Add("Sunday");
```

If you want to retrieve a single selected item to a variable , you can code like this

```
string var;
var = listBox1.Text;
```

How to bind a ListBox to a List ?

First you should create a fresh List Object and add items to the List.

```
List<string> nList = new List<string>();
nList.Add("January");
nList.Add("February");
nList.Add("March");
nList.Add("April");
```

The next step is to bind this List to the Listbox. In order to do that you should set datasource of the Listbox.

```
listBox1.DataSource = nList;
```


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

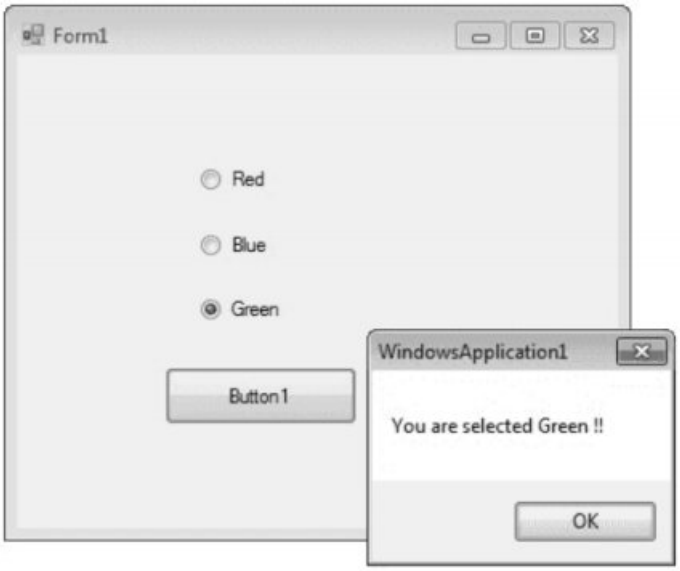


Fig. 3.12 Radio Button

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

```
radioButton1.Checked = true;
```

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 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.

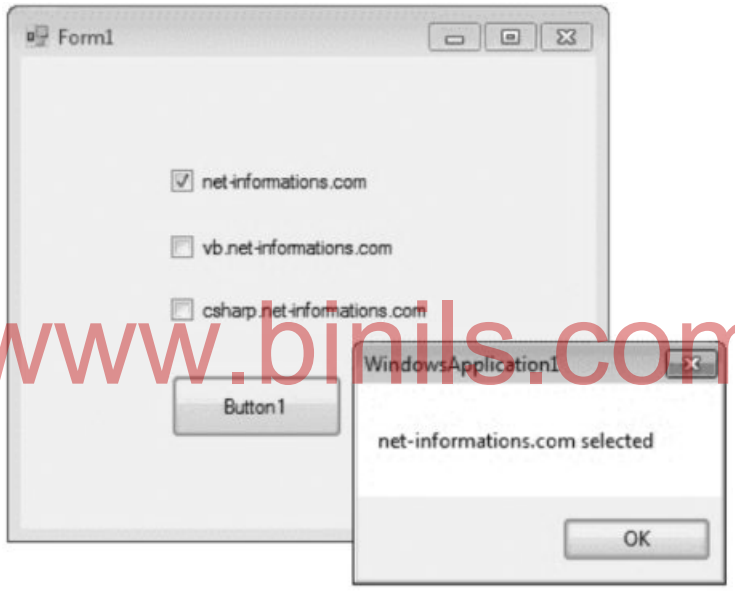


Fig. 3.13 Checkbox

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.

```
checkBox1.Text = "Net-informations.com";
```

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.

```
checkBox1.ThreeState = true;
```

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.

PictureBox Control

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

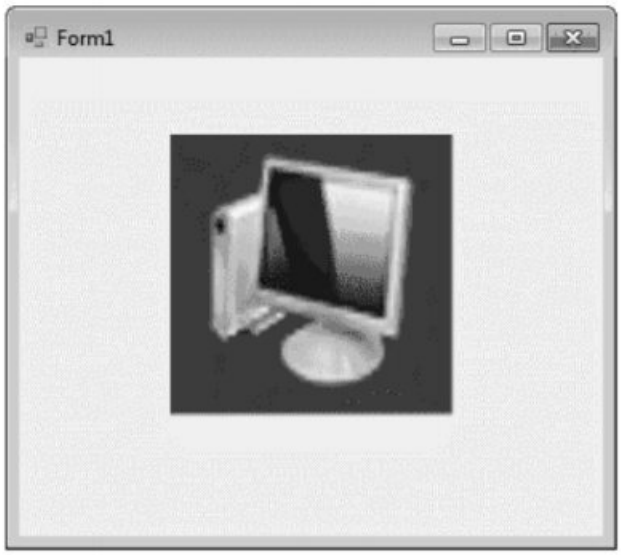


Fig. 3.14 Picture Box

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.

GroupBox

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 categories related controls in a group.

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 mouse and set its properties and events.

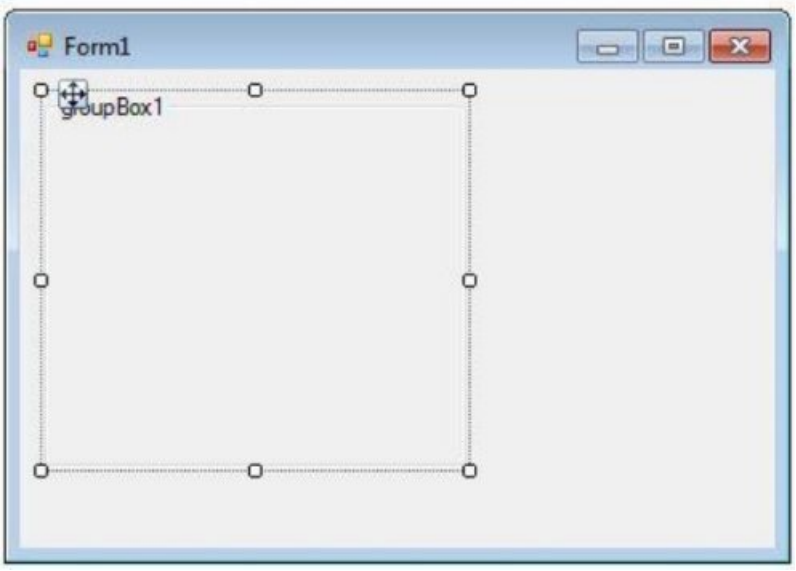


Fig. 3.15 GroupBox

Setting GroupBox Properties

After you place a GroupBox 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 click on a control and select Properties menu item. The Properties window looks like Figure 2.

Location, Height, Width, and Size

The Location property takes a Point that specifies the starting position of the GroupBox on a Form. The Size property specifies the size of the control. We can also use Width and Height property instead of Size property. The following code snippet sets Location, Width, and Height properties of a GroupBox control.

```
authorGroup.Location = new Point(10, 10);
authorGroup.Width = 250;
authorGroup.Height = 200;
```

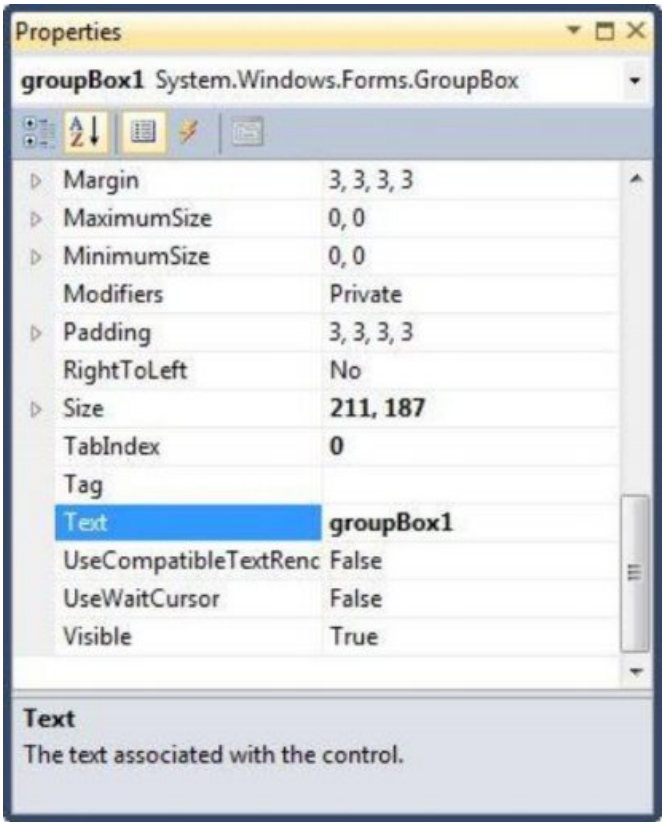


Fig. 3.16 GroupBox Properties

Background and Foreground

BackColor and ForeColor properties are used to set background and foreground color of a GroupBox respectively. If you click on these properties in Properties window, the Color Dialog pops up.

Name

Name property represents a unique name of a GroupBox control. It is used to access the control in the code. The following code snippet sets and gets the name and text of a GroupBox control.

```
authorGroup.Name = "GroupBox1";
```

Text

Text property of a GroupBox represents the header text of a GroupBox control. The following code snippet sets the header of a GroupBox control.

```
authorGroup.Text = "Author Details";
```

3.2 Advanced controls & Events

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.

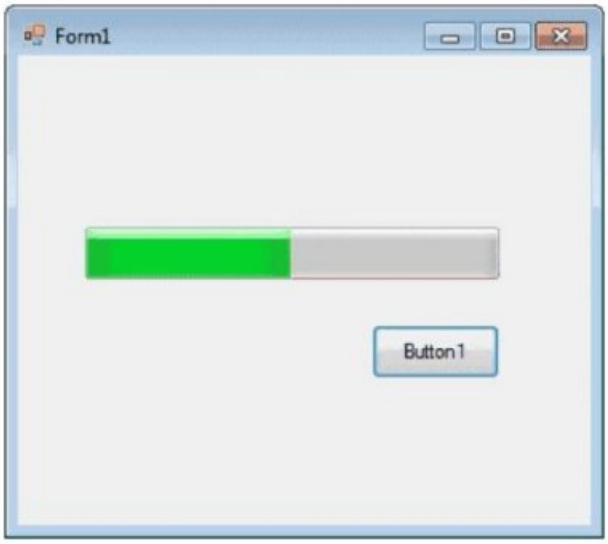


Fig. 3.17 Progress Bar

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;
            }
        }
    }
}
```

Timer Control

What is 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.

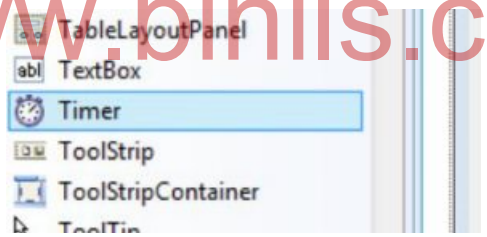


Fig. 3.18 Timer control in ToolBox

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.

Fig. 3.19 Timer control on Form

How to Timer Control ?

We can control programs with Timer Control in millisecond, seconds, minutes and even in hours. The Timer Control allows us to set Intervel 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.

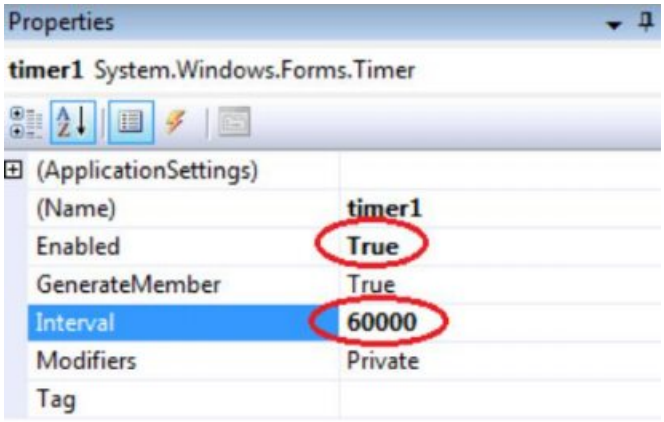


Fig. 3.20 Timer Properties

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.

```
using System;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }
      private void timer1_Tick(object sender, EventArgs e)
      {
         label1.Text = DateTime.Now.ToString();
      }
   }
}
```

Start and Stop Timer Control

The Timer control have included the Start and Stop methods for start and stop the Timer control functions.

```
Timer1.Start();

Timer1.Stop();
```

MonthCalendar control

MonthCalendar is a selectable calendar widget. On the MonthCalendar, a user can select a day, or a range of days. The user can also scroll through the months. This control provides many useful options. It is ideal for instant calendars.

Fig. 3.21 MonthCalendar

MaxDate, MinDate. The MonthControl provides two important properties of the calendar called MaxDate and MinDate. These indicate the maximum and minimum selectable dates. These dates give you a lot of range to select dates.

Date Properties:
MinDate: 1/1/1753
MaxDate: 12/31/9998

ShowToday. There are two properties that allow you to change whether and how the "Today" text at the bottom of the calendar appears. ShowToday is by default set to true. If you set it to false, it will not be present at the bottom of the calendar.

ShowTodayCircle: The ShowTodayCircle property adjusts the visibility of the box on the left of the "Today" display.

DateChanged. The MonthCalendar provides an event-driven user interface and you can provide and hook up event handlers to execute code on user actions. The DateChanged event allows you to detect whenever the user changes the date to something else.

Tooltip

A tooltip is a small pop-up window that displays some information when you rollover on a control.

Creating a Tooltip

Tooltip class represents a tooltip control. Once a Tooltip object is created, we need to call SetToolTip method and pass a control and text. The following code snippet creates a Tooltip and attach to a Button control using SetToolTip method.

```
ToolTip toolTip1 = new ToolTip();
toolTip1.ShowAlways = true;
toolTip1.SetToolTip(button1, "Click me to execute.");
```

If you rollover on Button control, you will see the following output.

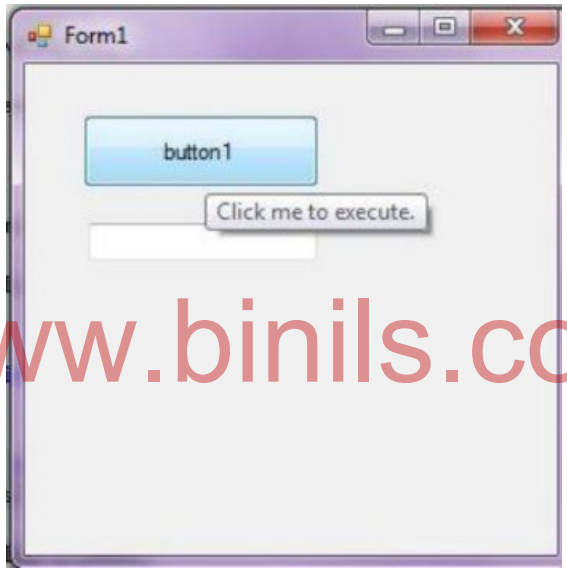


Fig. 3.22 Tooltip

Tooltip Properties

- Active - A tooltip is currently active.
- AutomaticDelay - Automatic delay for the tooltip.
- AutoPopDelay - The period of time the ToolTip remains visible if the pointer is stationary on a control with specified ToolTip text.
- InitialDelay - Gets or sets the time that passes before the ToolTip appears.
- IsBaloon - Gets or sets a value indicating whether the ToolTip should use a balloon window.
- ReshowDelay - Gets or sets the length of time that must transpire before subsequent ToolTip windows appear as the pointer moves from one control to another.
- ShowAlways - Displays if tooltip is displayed even the parent control is not active.
- ToolTipIcon - Icon of tooltip window.
- ToolTipTitle - Title of tooltip window.

- UseAnimation - Represents weather an animation effect should be used when displaying the tooltip.
- UseFading - Represents weather a fade effect should be used when displaying the tooltip.

TabControl

The TabControl manages tab pages where each page may host different child controls.

Creating a TabControl

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

Design-time

To create a TabControl control at design-time, you simply drag and drop a TabControl control from Toolbox onto a Form in Visual Studio. After you drag and drop a TabControl on a Form, the TabControl1 is added to the Form and looks like Figure 3.23.

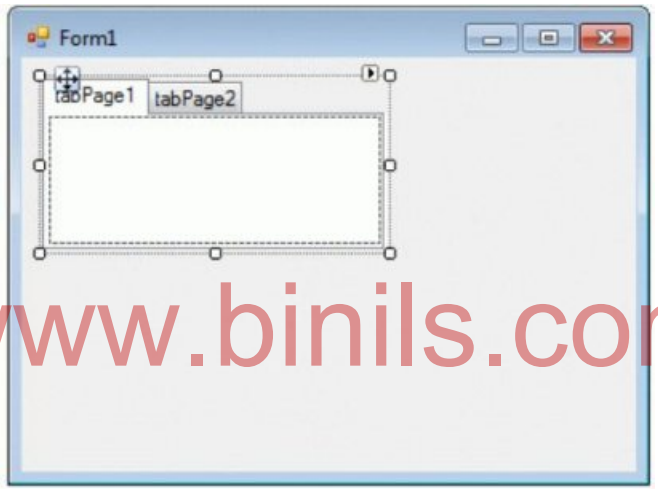


Fig. 3.23 Tab control

A TabControl is just a container and has no value without tab pages. As you can see from Figure 3.23, by default two Tab Pages are added to the TabControl. We can add and remove tab pages by clicking on the Tasks handle and selecting Add and Remove Tab links as you see in Figure 3.24.

Fig. 3.24 Adding tabs in Tab control

Add Tab link adds next tab page and Remove Tab removes the current tab page from a Tab Control. We will discuss tab pages in more details later in this tutorial.

Run-time

TabControl class represents a tab control. The following code snippet creates a TabControl and sets its Name, BackColor, ForeColor, Font, Width, and Height properties.

C# Code:

```
// Create a TabControl and set its properties
TabControl dynamicTabControl = new TabControl();
dynamicTabControl.Name = "DynamicTabControl";
dynamicTabControl.BackColor = Color.White;
dynamicTabControl.ForeColor = Color.Black;
dynamicTabControl.Font = new Font("Georgia", 16);
dynamicTabControl.Width = 300;
dynamicTabControl.Height = 200;
```

Once the TabControl control is ready with its properties, we need to add it to a Form by calling Form.Controls.Add method.

```
Controls.Add(dynamicTabControl);
```

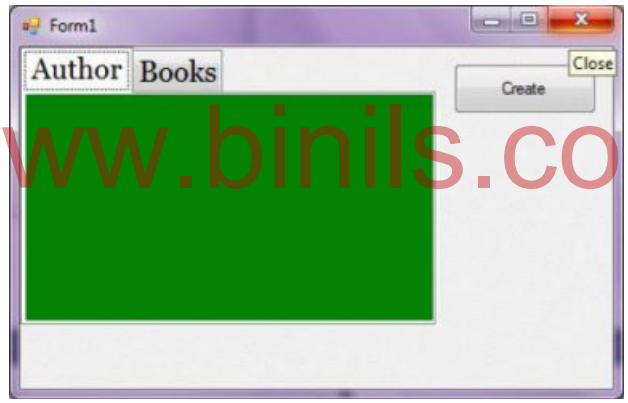


Fig. 3.25 Tab control with two tabs

Panel

The Panel Control is a container control to host a group of similar child controls. One of the major use of 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.

Panel elements are components that control the rendering of elements—their size and dimensions, their position, and the arrangement of their child content

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 3.26.

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

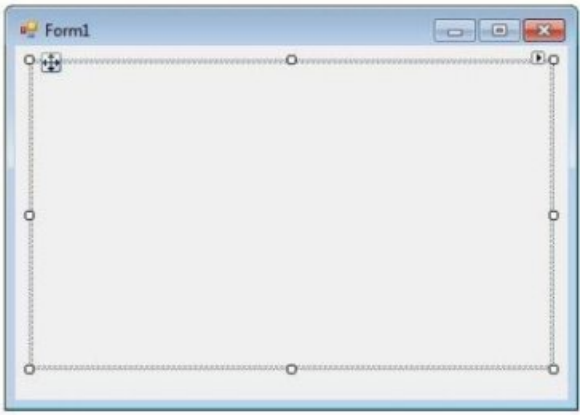


Fig. 3.26 Panel control placed on Form

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 = new Panel();
```

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);
```

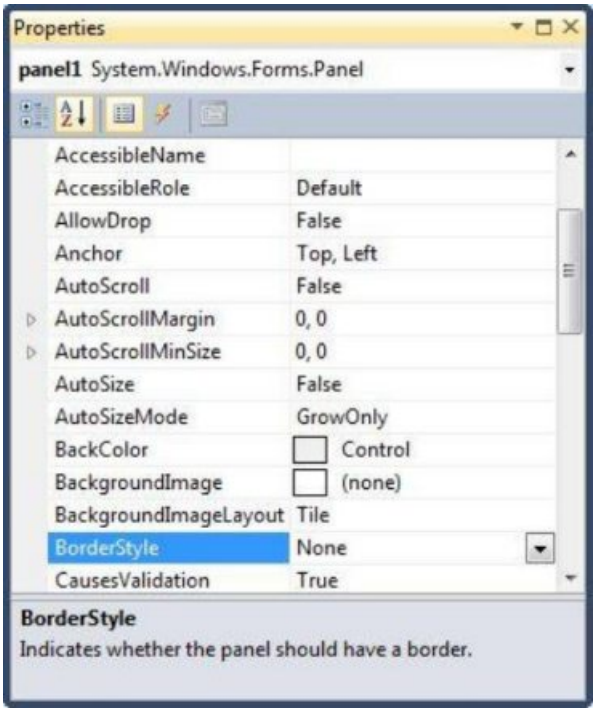


Fig. 3.27 Panel properties

Setting Panel Properties

After you place a Panel Control on a form, the next step is to set its properties.

The easiest way to set properties is from the Properties Window. You can open the Properties window by pressing F4 or right-clicking on a control and selecting the Properties menu item. The Properties window looks as in Figure 3.27.

The Panel has most of the common control properties. Here I will discuss the main purpose of a Panel.

Adding Controls to a Panel

You can add controls to a Panel by dragging and dropping a control to the Panel. We can add controls to a Panel at run-time by using its Add method. The following code snippet creates a Panel, creates a TextBox and a CheckBox and adds these two controls to a Panel.

```
private void CreateButton_Click(object sender, EventArgs e)
{
   Panel dynamicPanel = new Panel();
   dynamicPanel.Location = new System.Drawing.Point(26, 12);
   dynamicPanel.Name = "Panel1";
   dynamicPanel.Size = new System.Drawing.Size(228, 200);
   dynamicPanel.BackColor = Color.LightBlue;
   TextBox textBox1 = new TextBox();
   textBox1.Location = new Point(10, 10);
   textBox1.Text = "I am a TextBox5";
   textBox1.Size = new Size(200, 30);
```

```
    CheckBox checkBox1 = new CheckBox();
   checkBox1.Location = new Point(10, 50);
   checkBox1.Text = "Check Me";
   checkBox1.Size = new Size(200, 30);
    dynamicPanel.Controls.Add(textBox1);
   dynamicPanel.Controls.Add(checkBox1);
    Controls.Add(dynamicPanel);
}
```

The output looks as in Figure 3.28

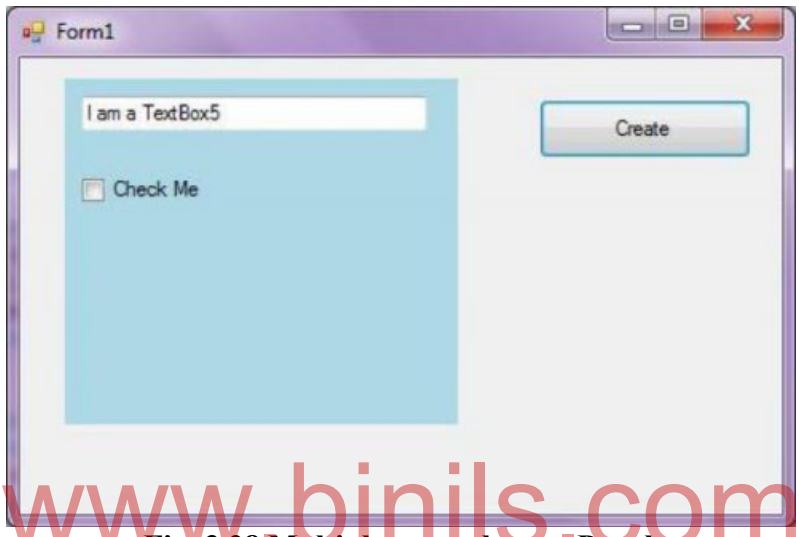


Fig. 3.28 Multiple controls on a Panel

Show and Hide a Panel

Instead of showing and hidding individual controls, we can group controls that we want to show and hide and place them on two different Panels and show and hide the Panels. To show and hide a Panel, we use the Visible property.

```
dynamicPanel.Visible = false;
```

Events

An *event* in C# is a way for a class to provide notifications to clients of that class when some interesting thing happens to an object. The most familiar use for events is in graphical user interfaces; typically, the classes that represent controls in the interface have events that are notified when the user does something to the control (for example, click a button).

Event handling is important to develop any graphical user interfaces (GUI) application. When a user interacts with a GUI control (e.g., clicking a button on a form), one or more methods are executed in response to the above event. Events can also be generated without user interactions. Event handlers are methods in an object that are executed in response to some events occurring in the application. To understand the event handling model of .Net framework, we need to understand the concept of delegate.

EVENTS

Windows Forms programs are event-based.

Event: Click

Forms are idle until the user takes an action upon a control. When you add a Click event handler to a Button, the event handler method will be invoked when the user clicks on the button.

Button. A button accepts clicks. In Windows Forms we use a Button control that accepts click events and performs other actions in the user interface. This control provides a way to accept input—and invoke logic based on that input.

Fig.3,29 Button click event

Action: You can use a Button control to perform an action when the user clicks or presses a key to activate the button.

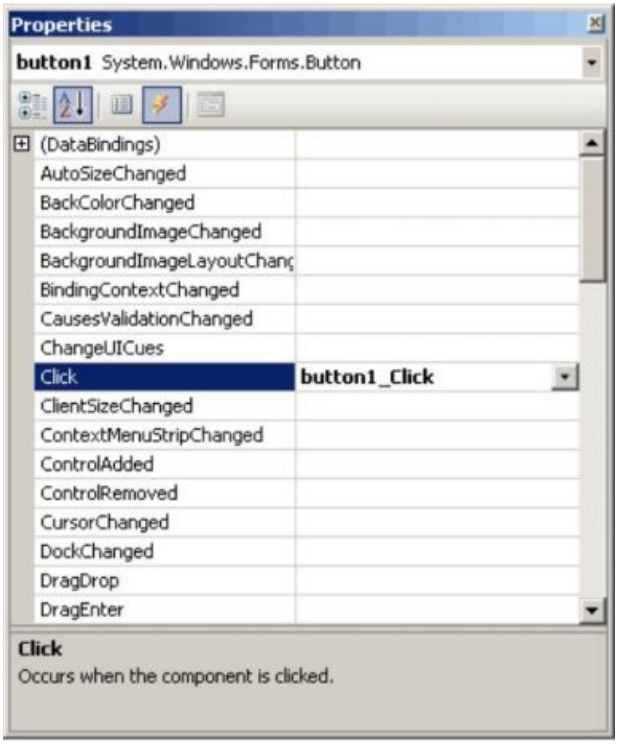


Fig.3.30 Button Events

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

namespace WindowsFormsApplication21
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      private void button1_Click(object sender, EventArgs e)
      {
         MessageBox.Show("This is a test message");
      }
   }
}
```

Event : Close

The Closing event occurs as the form is being closed. When a form is closed, all resources created within the object are released and the form is disposed. If you cancel this event, the form remains opened. To cancel the closure of a form, set the Cancel property of the CancelEventArgs passed to your event handler to **true**.

When a form is displayed as a modal dialog box, clicking the **Close** button (the button with an X at the upper-right corner of the form) causes the form to be hidden and the DialogResult property to be set to DialogResult.Cancel**.** You can override the value assigned to the DialogResult property when the user clicks the **Close** button by setting the DialogResult property in an event handler for the Closing event of the form.

Note

When the Close method is called on a Form displayed as a modeless window, you cannot call the Show method to make the form visible, because the form's resources have already been released. To hide a form and then make it visible, use the Control.Hide method.

The Form.Closed and Form.Closing events are not raised when the Application.Exit method is called to exit your application. If you have validation code in either of these events that must be executed, you should call the Form.Close method for each open form individually before calling the Exit method.

Event: Deactivate

Deactivate occurs when an object is no longer the active window.

An object can become active by user action, such as clicking it, or by using the **Show** or **SetFocus** methods in code.

The Activate event can occur only when an object is visible. For example, a form loaded with the **Load** statement isn't visible unless you use the **Show** method or set the form's **Visible** property to **True**.

The Activate and Deactivate events occur only when moving the focus within an application. Moving the focus to or from an object in another application doesn't trigger either event. The Deactivate event doesn't occur when unloading an object.

The Activate event occurs before the GotFocus event; the LostFocus event occurs before the Deactivate event.

Remarks

A window is deactivated (becomes a background window) when:

- A user switches to another window in the current application.
- A user switches to the window in another application by using ALT+TAB or by using Task Manager.
- A user clicks the taskbar button for a window in another application.

After a window is first deactivated, it may be reactivated and deactivated many times during its lifetime. If an application's behavior or state depends on its activation state, it can inspect IsActive to determine which activation state it's in.

Event: Load

Every Windows Forms program will use the Form class. In many programs, the Load, FormClosing and FormClosed event handlers provide needed functionality. We look closer at Form. We demonstrate several event handlers on the Form class.

1. To start, create a new Windows Forms program.
2. Next, we add the Load event handler. In Visual Studio, double-click somewhere on the visible Form.
3. In Form1_Load, you can set things like the title text (Text property) or the Window position.

```
using System;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
  public partial class Form1 : Form
  {
    public Form1()
    {
      InitializeComponent();
    }
    private void Form1_Load(object sender, EventArgs e)
    {
      // You can set properties in the Load event handler.
      this.Text = DateTime.Now.DayOfWeek.ToString();
      this.Top = 60;
      this.Left = 60;
    }
```

```
    private void Form1_FormClosing(object sender, FormClosingEventArgs e)
    {
        // You can cancel the Form from closing.
        if ((DateTime.Now.Minute % 2) == 1)
        {
            this.Text = "Can't close on odd minute";
            e.Cancel = true;
        }
    }
    private void Form1_FormClosed(object sender, FormClosedEventArgs e)
    {
        // You can write to a file to store settings here.
    }
  }
}
```

Event: MouseMove

This event occurs when the mouse pointer is moved over the control.
Namespace: System.Windows.Forms
Assembly: System.Windows.Forms (in System.Windows.Forms.dll)
Syntax
public event MouseEventHandler MouseMove
Handling Mouse Events

The window in Figure 3.31 lists multiple mouse events. Event handlers can be generated simply by double-clicking the desired event.

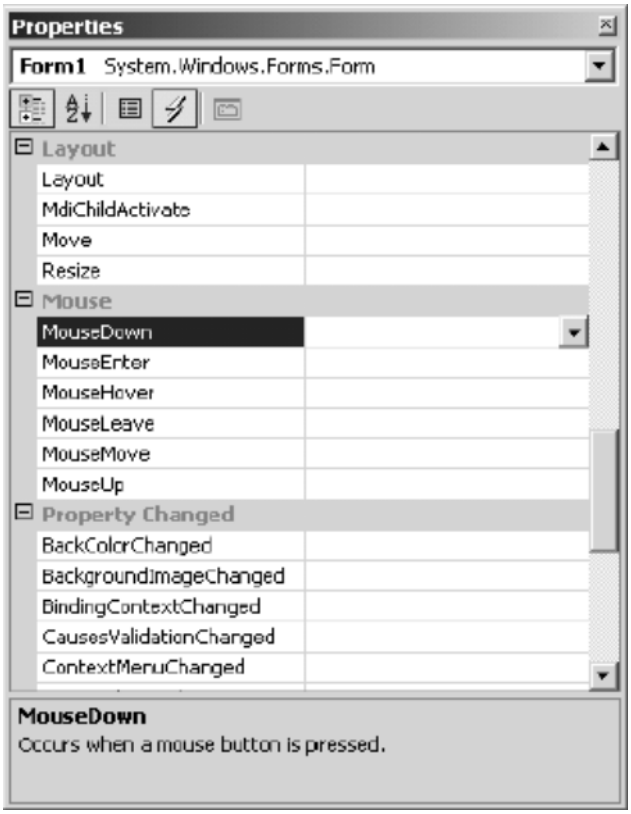


Figure 3.31: Mouse Events

To carry out some action after the mouse event occurs, we need to write the event handlers. The second parameter of the event handler method is a System.WinForms.MouseEventArgs object, which details the mouse's state. The MouseEventArgs' members are listed in Table 3.1.

Member	Description
Button	Indicates which mouse button was pressed: left, right, middle, or none
Clicks	Indicates the number of times the mouse button was pressed and released
Delta	Indicates a signed count of the number of detents the mouse wheel has rotated
X	The *x* coordinate of mouse click
Y	The *y* coordinate of mouse click

Table 3.1: MouseEventArgs Members

Listing 3.1 shows the event handler code for the MouseDown and MouseMove events.

Listing 3.1: Mouse Event Handlers

```
public void OnMouseDown(object sender, System.Windows.Forms.MouseEventArgs e)
{
   switch (e.Button)
   {
      case MouseButtons.Left:
         MessageBox.Show(this, "Left Button Click");
         break;
      case MouseButtons.Right:
         MessageBox.Show(this, "Right Button Click");
         break;
      case MouseButtons.Middle:
         break;
      default:
         break;
   }
}

private void OnMouseMove(object sender, System.Windows.Forms.MouseEventArgs e)
{
   this.Text = "Mouse Position:" + e.X.ToString() + "," + e.Y.ToString();
}
```


Figure 3.32 shows the output of Listing 3.1. A mouse click displays, in a message box, the mouse button clicked, while a mouse move shows the mouse's coordinates as the title of the form.

Fig. 3.32 Mouse Down

All mouse events are placed in

Namespace: System.Windows.Forms
Assembly: System.Windows.Forms (in System.Windows.Forms.dll)

Event: MouseDown

Occurs when the mouse pointer is over the control and a mouse button is pressed.
Syntax

public event MouseEventHandler MouseDown

Event: MouseUp

Occurs when the mouse pointer is over the control and a mouse button is released.
Syntax

public event MouseEventHandler MouseUp

Handle Keyboard Input at the Form Level in C#

Windows Forms processes keyboard input by raising keyboard events in response to Windows messages. Most Windows Forms applications process keyboard input exclusively by handling the keyboard events.

How do I detect keys pressed in C#

You can detect most physical key presses by handling the KeyDown or KeyUp events. Key events occur in the following order:

1. KeyDown
2. KeyPress
3. KeyUp

How to detect when the Enter Key Pressed in C#

The following C# code behind creates the KeyDown event handler. If the key that is pressed is the Enter key, a MessegeBox will displayed .

```
if (e.KeyCode == Keys.Enter)
{
        MessageBox.Show("Enter Key Pressed ");
}
```

How to get TextBox1_KeyDown event in your C# source file ?

Select your TextBox control on your Form and go to Properties window. Select Event icon on the properties window and scroll down and find the KeyDown event from the list and double click the Keydown Event. The you will get the KeyDown event in your source code editor.

```
private void textBox1_KeyDown(.....)
{
}
```

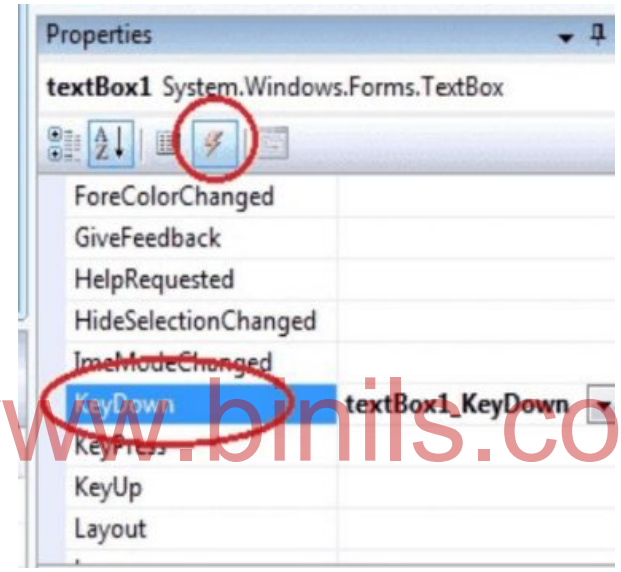


Fig. 3.33 Handling Key events

Difference between the KeyDown Event, KeyPress Event and KeyUp Event

KeyDown Event : This event raised as soon as the user presses a key on the keyboard, it repeats while the user keeps the key depressed.

KeyPress Event : This event is raised for character keys while the key is pressed and then released. This event is not raised by noncharacter keys, unlike KeyDown and KeyUp, which are also raised for noncharacter keys

KeyUp Event : This event is raised after the user releases a key on the keyboard.

Event: Keypress

Occurs when a character. space or backspace key is pressed while the control has focus.

Syntax

public event KeyPressEventHandler KeyPress

Use the KeyChar property to sample keystrokes at run time and to consume or modify a subset of common keystrokes.

To handle keyboard events only at the form level and not enable other controls to receive keyboard events, set the KeyPressEventArgs.Handled property in your form's KeyPress event-handling method to **true**.

Event: KeyDown. You can read key down events in the TextBox control in Windows Forms. The Windows Forms system provides several key-based events. This tutorial uses the KeyDown event handler which is called before the key value actually is painted.

You can cancel the key event in the KeyDown event handler as well, although this is not demonstrated. The program will display an alert when the Enter key is pressed. An alternative alert message when the Escape key is pressed.

Windows Forms class that uses KeyDown on TextBox: C#

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

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

    private void textBox1_KeyDown(object sender, KeyEventArgs e)
    {
      //
      // Detect the KeyEventArg's key enumerated constant.
      //
      if (e.KeyCode == Keys.Enter)
      {
        MessageBox.Show("You pressed enter! Good job!");
```

```
        }
        else if (e.KeyCode == Keys.Escape)
        {
            MessageBox.Show("You pressed escape! What's wrong?");
        }
    }
  }
}
```

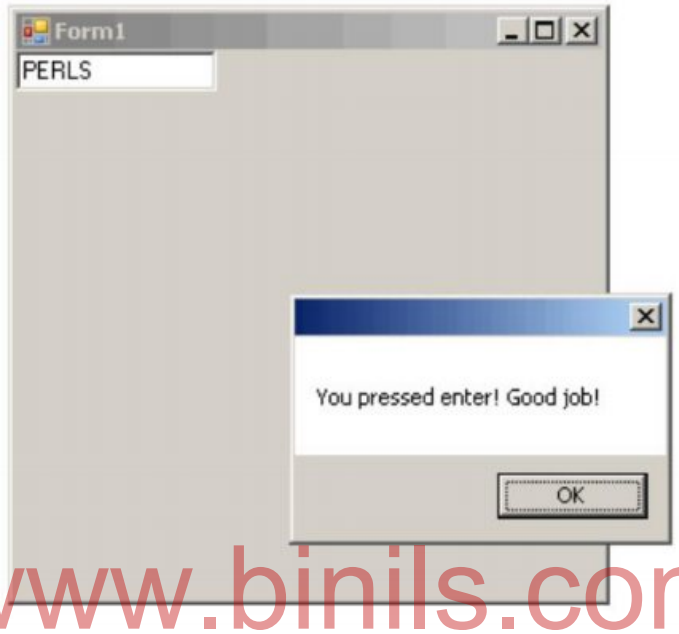


Fig. 3.33 Key Down

Event: KeyUp

Occurs when a key is released while the control has focus.

Namespace: System.Windows.Forms
Assembly: System.Windows.Forms (in System.Windows.Forms.dll)

Syntax

public event KeyEventHandler KeyUp

3.3 C# MDI Form (Multiple Document Interface)

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).

Single document interface (SDI)

Applications 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.

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. 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.

Fig. 3.34 Multiple Forms inside an MDI Form

Any windows can become an MDI parent, if you set the IsMdiContainer property to True.

```
IsMdiContainer = true;
```

Multiple-document interface (MDI) applications allow 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.

You have probably seen an MDI application where you can display multiple "child" windows inside of a main application window.

Creating an MDI Application

It's easy to create an MDI application. You could start by creating a new standard Window's Form application, then change the IsMdiContainer property of the form to true. Try this and notice how the appearance of the form changes. That's it, you know have a MDI forms application. However, all you have now is a container with nothing in it. You need to create the child forms that live in the MDI container as well as provide the mechanism to create these forms.

1. Create a new Windows Forms Application.
2. Go to the project and Add a new Windows Form
3. Select the MDI Parent Form

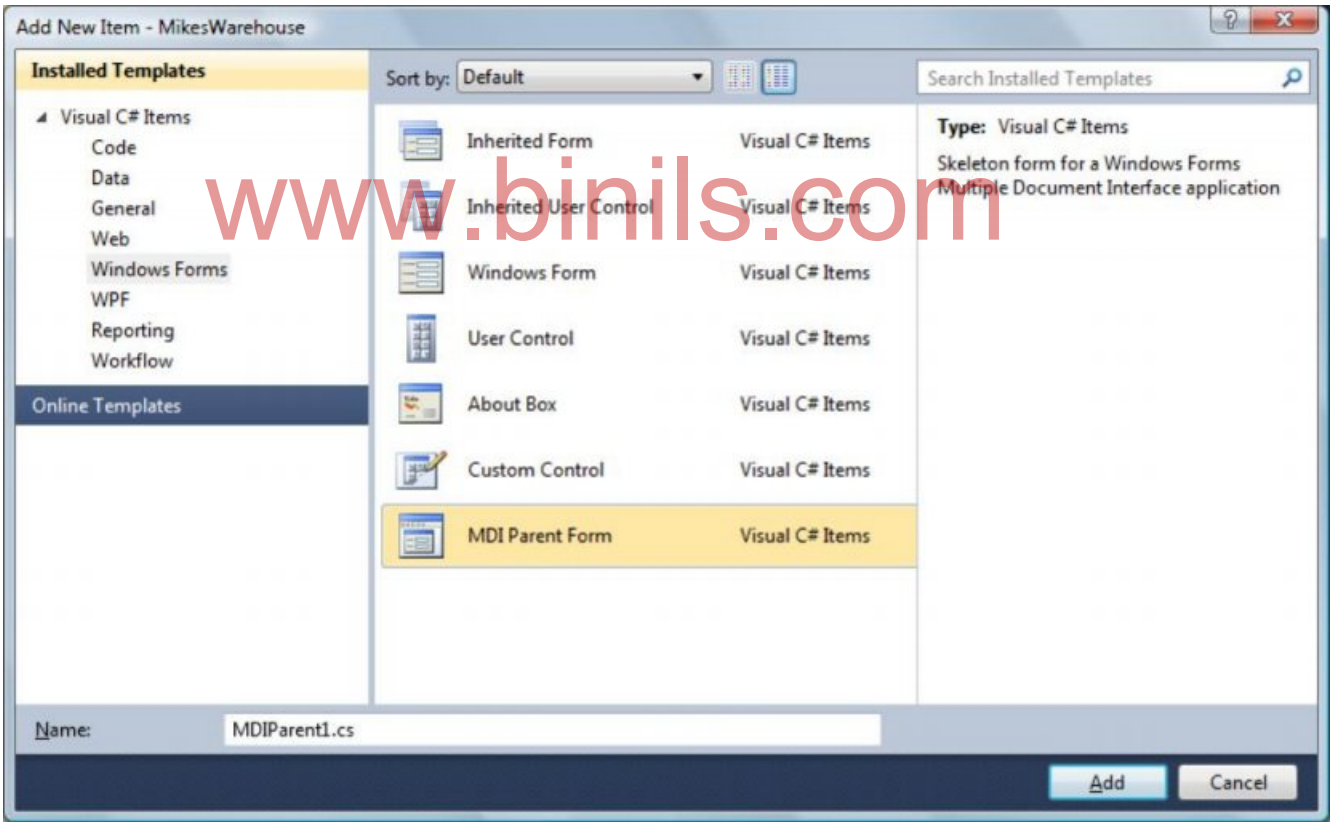


Fig: 3.35 Creating MDI application

Now that we have added an MDI form, we want to make this our main form that executes when the program starts. Also, we no longer need the default Form1 that was added by Visual Studio when we created the project.

Delete Form1 from your project. Now we need to tell our program to start our MDI form. To do this go to your Program.cs file. Your Program class probably looks something like this:

```
static class Program
{
   /// <summary>
   /// The main entry point for the application.
   /// </summary>
   [STAThread]
   static void Main()
   {
      Application.EnableVisualStyles();
      Application.SetCompatibleTextRenderingDefault(false);
      Application.Run(new Form1());
   }
}
```

Notice the last line where the call to Application.Run creates a new Form1. Replace Form1 with the name of you new MDI form. I called mine MDIMain.

Now when you start your application, MDIMain form will be loaded.

Try this out and see what happens.

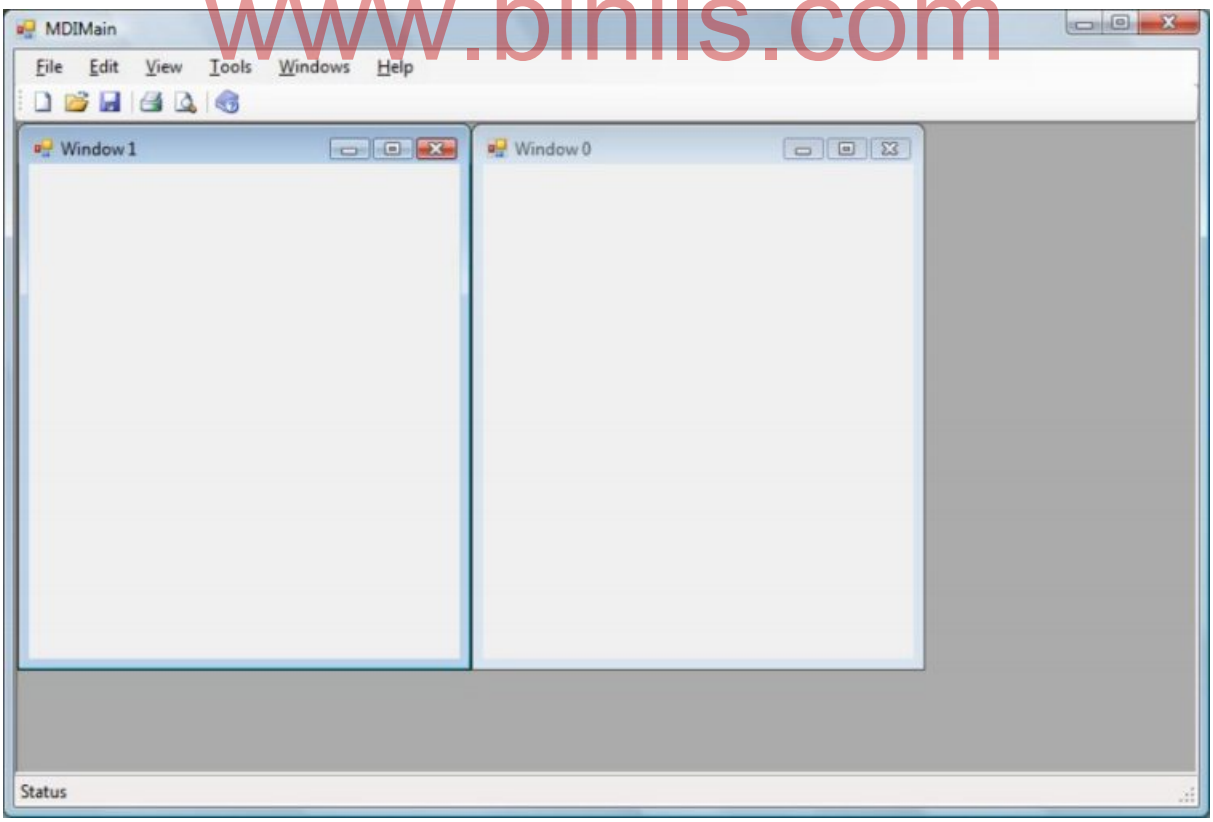


Fig: 3.36 Creating Child windows

Visual Studio creates a new MDI application for you and loads up the MDI form class with a lot example code on how to use the MDI. A menu and a toolbar are added for you. In the menu, you can select Windows->New Window and have a new child window displayed. Keep selecting Windows->New Window and you get more child windows. You can also see how to use the Windows features to arrange the windows by exploring the code for the Windows menu.

Arranging MDI Child Windows

The LayoutMdi Method is used with MdiLayout enumeration to rearrange the child forms in an MDI Parent Form.

Get to Work

With the help of below given example, you can learn how to use LayoutMdi method with MdiLayout enumeration for the Mdi Parent Form and you have to use enumeration in the code of click event of the Cascade Windows menu item.

Step to Create and Implement MDI Child Form

1. Assumes there is an MDI parent form having MenuStrip with option New, Window and Close in New Menu, main form contain one Child form having a RichTexBox.

Fig: 3.36

Add one more control in Main Form MenuStrip as Cascade Windows.

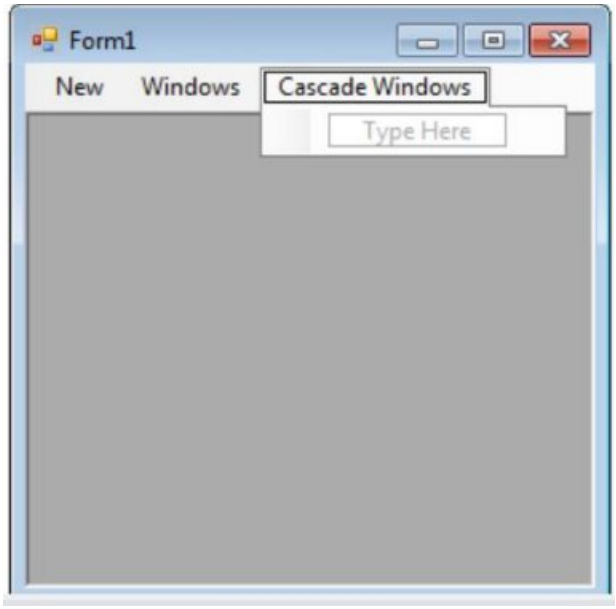


Fig: 3.37

2. Double click on Cascade Windows control and write this Code.

```
private void cascadeWindowToolStripMenuItem_Click(object sender, EventArgs e)
   {
      this.LayoutMdi(System.Windows.Forms.MdiLayout.Cascade);
   }
```


3. Debug the application and click on New button two times then two MDI Child form with RichTextBox will open. Now by using Cascade Windows control in the Main Menu you can arrange all the opened Mdi Child Form in Cascade mode.

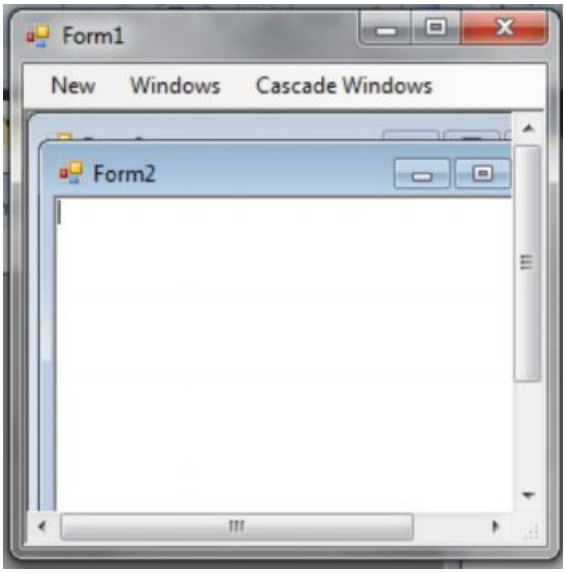


Fig: 3.38

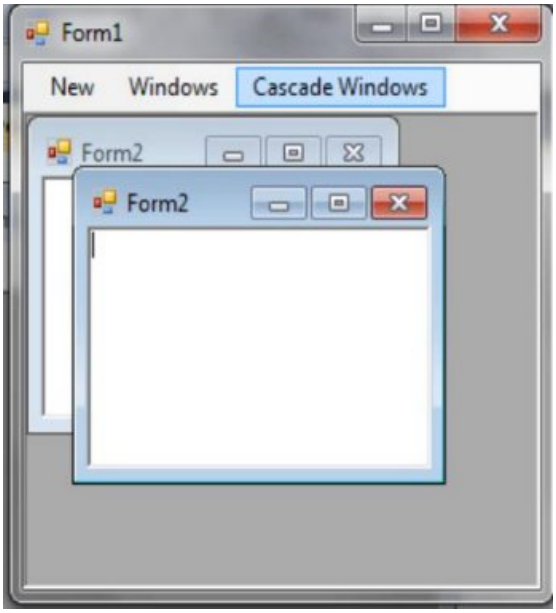


Fig: 3.39

3.4 Menus and Dialog Boxes

A Menu on a Windows Form is created with a MainMenu object, which is a collection of MenuItem objects. MainMenu is the container for the Menu structure of the form and menus are made of MenuItem objects that represent individual parts of a menu.

You can add menus to Windows Forms at design time by adding the MainMenu component and then appending menu items to it using the Menu Designer.

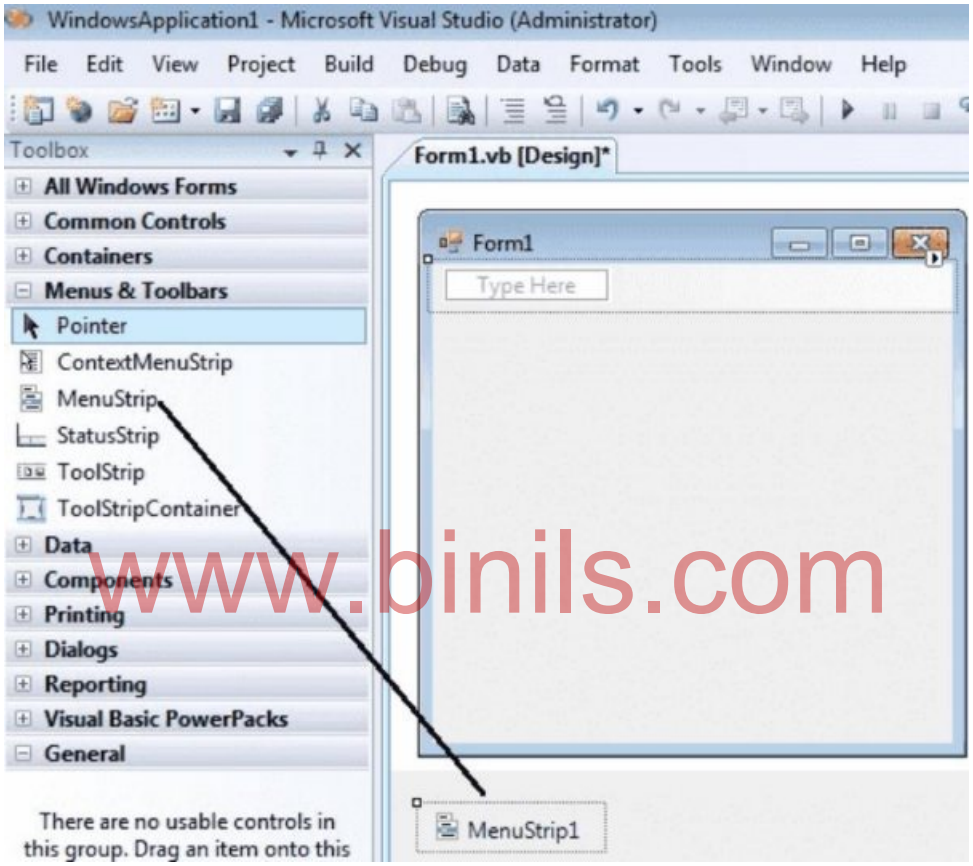


Fig: 3.40 Adding menu to an application

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

Adding Menu, Menu Items to a Menu

First add a MainMenu control to the form. Then to add menu items to it add MenuItem objects to the collection. By default, a MainMenu object contains no menu items, so that the first menu item added becomes the menu heading. Menu items can also be dynamically added when they are created, such that properties are set at the time of their creation and addition.

Fig. 3.41 Creating menu items

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

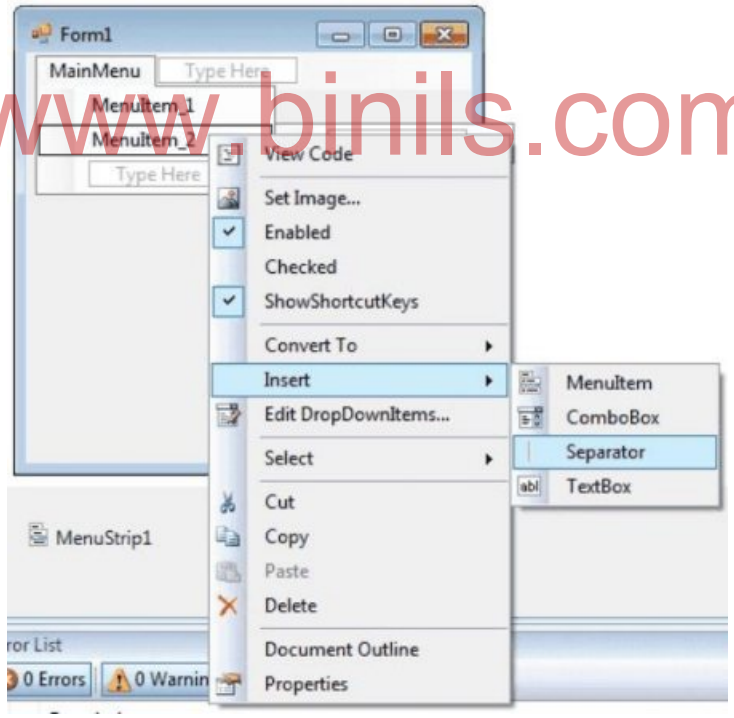


Fig. 3.41 Adding separator

After creating the Menu on the form , you have to double click on each menu item and write the programs there depends on your requirements.

The following C# program shows how to show a messagebox

when clicking a Menu item.

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

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

      private void menu1ToolStripMenuItem_Click(object sender, EventArgs e)
      {
         MessageBox.Show("You  selected MenuItem_1");
      }
   }
}
```

Sub menus

1. We add a control of the type menuStrip to the form using the designer
2. Now in the form a yet invisible menu bar is created, which is still empty. In the designer you could now write in this bar to add menus, but we focus on the dynamic part.
3. Super menus, so menus on the highest level in the menu bar can be edited with the property Items of the class menuStrip.

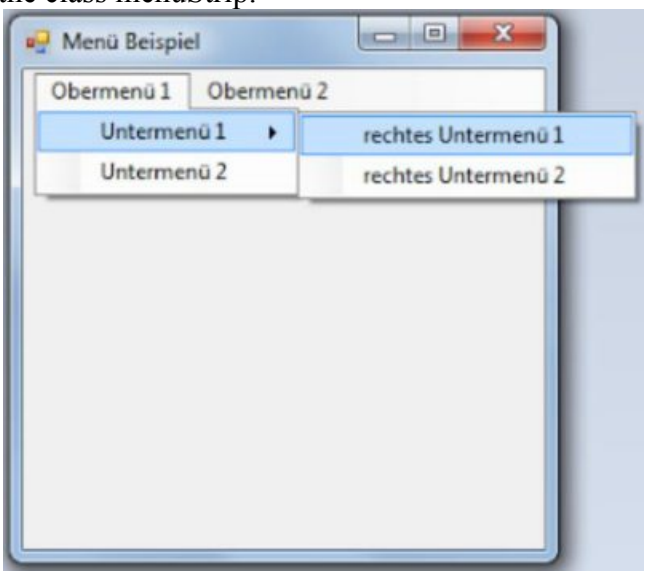


Fig. 3.42 Sub Menu

The following code commands create 2 super menus:

```
menuStrip1.Items.Add("Obermenü 1");
menuStrip1.Items.Add("Obermenü 2");
```

Using indicces (e.g. menuStrip1.Items[0]) we can access these menuentries

Context menus (Popup menus)

Context menus are used inside applications to provide users access to often used commands by means of a right-click of the mouse. Often, context menus are assigned to controls, and provide particular commands that relate to that precise control.

Using Dialog boxes

Dialog boxes are used to gather input from users. You can create your own **dialog boxes** or use the built-in **dialog boxes**, such as the FolderBrowserDialog and FontDialog.,

Using Built-in Dialog Boxes in Your Application

Several built-in dialog box components are available for use in your C# applications. You can find these components in the **Dialogs** tab of the **Toolbox**. When you add components to a Windows form, do don't see them on the form. Instead, they are added to the component tray, underneath the form. The following topics describe the most common built-in dialog boxes.

Table 3.2 Built-in Dialog boxes

Topic	Description
Display a Color Palette	Describes how to use the ColorDialog to apply a color to a Windows form.
Browse a Folder	Describes how to use the FolderBrowserDialog to display a folder path on a Windows form
Display a List of Fonts	Demonstrates how to use the built-in FontDialog to apply a font to text
Save a File to a Folder	Describes how to save text that is added to a RichTextBox control to a location specified in the SaveFileDialog.
Display an OpenFileDialog Dynamically	Describes how to display an OpenFileDialog at run time.

ColorDialog Class

Represents a common dialog box that displays available colors along with controls that enable the user to define custom colors.

The following example illustrates the creation of new ColorDialog. This example requires that the method is called from within an existing form that has a TextBox and Button placed on it.

```
private void button1_Click(object sender, System.EventArgs e)
 {
   ColorDialog MyDialog = new ColorDialog();

 // Keeps the user from selecting a custom color.
   MyDialog.AllowFullOpen = false ;

 // Allows the user to get help. (The default is false.)
   MyDialog.ShowHelp = true ;

 // Update the text box color if the user clicks OK
   if (MyDialog.ShowDialog() == DialogResult.OK)
      textBox1.ForeColor =  MyDialog.Color;
 }
```

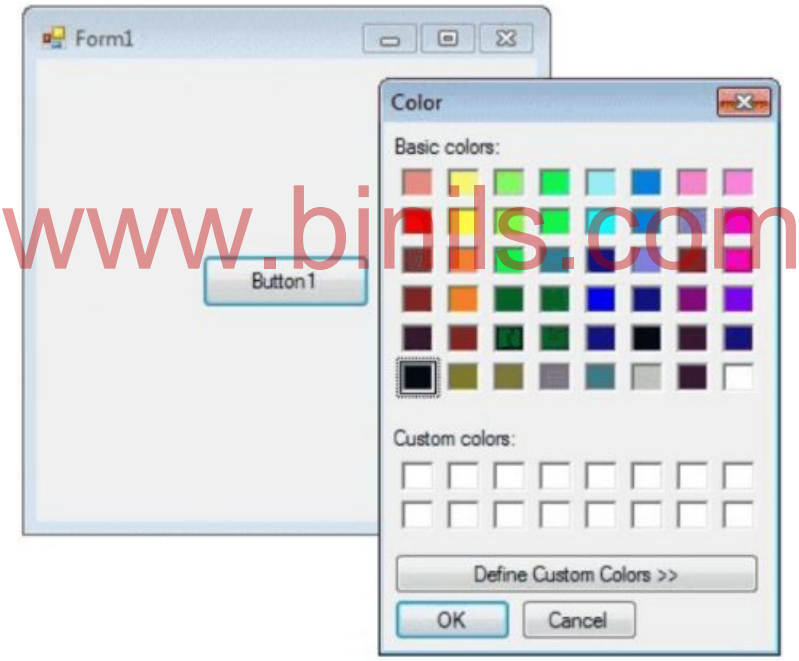


Fig. 3.43 Color Dialog

OpenFileDialog class

FileDialog class d isplays a dialog box from which the user can select a file. FileDialog is an abstract class that contains common behavior for
the OpenFileDialog and SaveFileDialog classes. It is not intended to be used directly

OpenFileDialog class prompts the user to open a file. This class cannot be inherited. This class allows you to check whether a file exists and to open it.

.

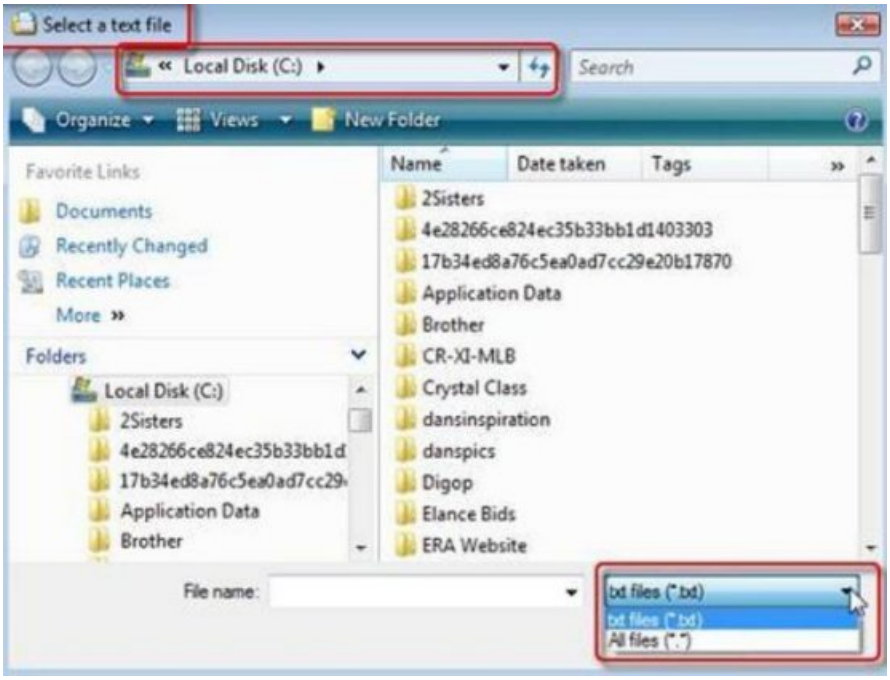


Fig. 3.44 Open File Dialog

To display the folder browser dialog box

1. On the **File** menu, click **New Project**.
   The **New Project** dialog box appears.
2. Click **Windows Forms Application** and then click **OK**.
3. Add a **Label** control to the form, and use the default name, **Label1**.
4. Add a **Button** control to the form, and change the following properties in the **Properties** window:

Property	Value
Name	folderPath
Text	Path

5. Drag a **FolderBrowserDialog** component from the **Dialogs** tab of the **Toolbox** to the form.
6. Double-click the button to create the default event handler in the Code Editor.
7. In the folderPath_Click event handler, add the following code to display the folder browser dialog box and display the selected path in the label.

```
if (folderBrowserDialog1.ShowDialog() == DialogResult.OK)
{
   this.label1.Text = folderBrowserDialog1.SelectedPath;
}
```

8. Press F5 to run the code.
9. When the form appears, click **Path**, click a folder in the list, and then click **OK**.
10. Verify that the selected path appears in the label.
11. Close the application

Review Questions

UNIT - III

Part A (2 marks)

1. What is Windows Forms?
2. What is a control?
3. What is Common control?
4. What is the use of Button control?
5. What is the use of Textbox control?
6. Write the C# code to make a Textbox read only.
7. How will you make a Textbox to accept passwords?
8. How will you make a Timer control to create periodic events for every minute?
9. What is an event?
10. What is meant by event handling?
11. What is MouseMove event?
12. What is meant by MDI?
13. What is a SDI application? Give example.
14. What is a Menu control?

Part B (3 marks)

1. List the features of Windows Forms application.
2. List any 5 common controls in Toolbox and state their purpose.
3. Mention the events associated with Textbox control and explain.
4. Compare RadioButton and Checkbox controls.
5. What are the uses of Picture box control? List its properties.
6. List the uses of Groupbox in windows application.
7. What is the use of MonthCalendar control?
8. What is a Tooltip? How will you create a tool tip for a button by coding?
9. What is a Panel? What are the uses of it?
10. List the members of MouseEventArgs and describe them.
11. List the order of Key events and explain.
12. Differentiate between the KeyDown Event, KeyPress Event and KeyUp Event
13. List and explain the different built-in dialog boxes.

Part C (5 marks)

1. Draw the layout of Visual studio IDE and explain the various parts of it.
2. List the steps to create a new project in C#.
3. What is a ListBox? How will you populate listbox during loading of application? Write the code to bind listbox to list.
4. List any 2 advanced controls and discuss in detail.
5. List the important events associated with a control and explain with suitable code.
6. Write down the procedure of creating MDI application.
7. Describe the procedure of creating a Menu based window form application.
8. Create a Window form application that uses ColorDialog to change the background color of the form.

****

Unit-IV APPLICATION DEVELOPMENT USING ADO.NET

Lesson Objectives:

1. *Describe the features of ADO.NET and their object model for accessing data.*
2. *Create secure connections to a Microsoft SQL Server database by using the SqlConnection and SqlDataAdapter objects.*
3. *Programmatically read data from a SQL Server database by using a SqlDataReader object.*
4. *Store multiple tables of data in a DataSet object, and then display that data in DataGrid controls.*
5. *Explain what a stored procedure is and the reasons for using stored procedures when accessing a database.*

4.1. Introduction

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

ADO.NET includes .NET Framework data providers for connecting to a database, executing commands, and retrieving results.

ADO.NET is a data-access technology that enables applications to connect to data stores and manipulate data contained in them in various ways. It is based on the .NET Framework and it is highly integrated with the rest of the Framework class library. The ADO.NET API is designed so it can be used from all programming languages that target the .NET Framework, such as Visual Basic, C#, J# and Visual C++.

ADO uses a small set of Automation objects to provide a simple and efficient interface to OLE DB. This interface makes ADO a good choice for developers in higher level languages, such as Visual Basic and VBScript, who want to access data without having to learn the DETAILS of COM and OLE DB.

ADO.NET provides functionality to developers writing managed code similar to the functionality provided to native component object model (COM) developers by ActiveX Data Objects (ADO)

4.1.1. ADO.NET Object Model:

The ADO.NET object model consists of two key components as follows:

- **Connected model** (.NET Data Provider - a set of components including the Connection, Command, DataReader, and DataAdapter objects):
  We have the control over the database connection, so we have to explicitly open/close the objects in model have to directly talk to the database and hence are database specific Connection, Command and Data Reader are members of this set.

- **Disconnected model** (DataSet):
  It's complimentary to earlier model in the sense that the object itself decides when the connection will be opened and closed. we don't have to do it implicitly as a result only

one component of this model which talks to the database directly is data adapter whereas the cache which contains the data never speaks to the database directly and isn't database specific.

4.1.2. ADO.NET Architecture:

ADO.NET Components

There are two components of ADO.NET that you can use to access and manipulate data:

- .NET Framework data providers
- The DataSet

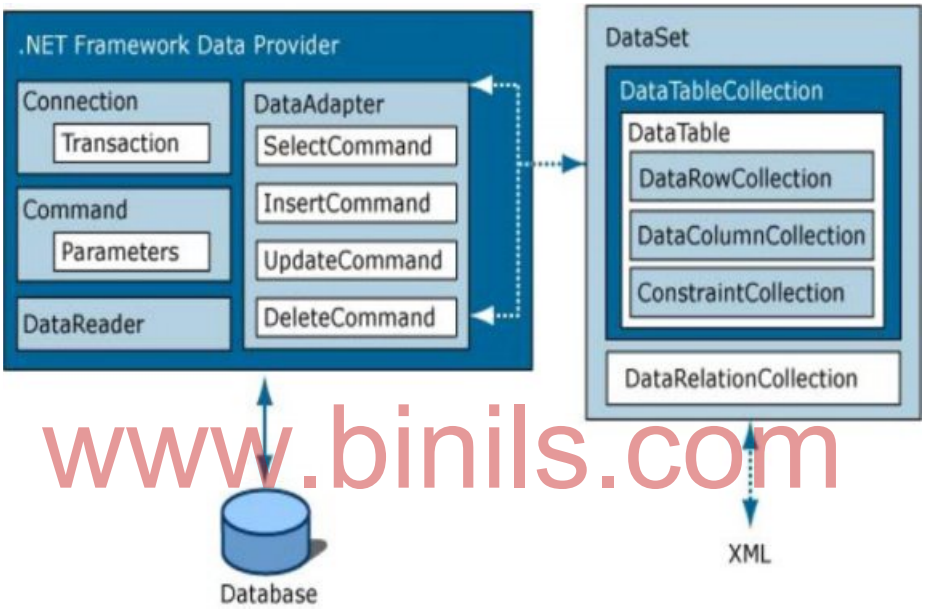


Figure 4.1: ADO.NET Architecture

4.1.3. DOT NET Framework Data Providers

The NET Framework Data Providers are components that have been explicitly designed for data manipulation and fast, forward-only, read-only access to data.

1. The **Connection** object provides connectivity to a data source.

2. The **Command** object enables access to database commands to return data, modify data, run stored procedures, and send or retrieve parameter information.

3. The **DataReader** provides a high-performance stream of data from the data source. Finally, the DataAdapter provides the bridge between the **DataSet** object and the **data source**.

4. The **DataAdapter** uses **Command** objects to execute SQL commands at the data source to both load the **DataSet** with data, and reconcile changes made to the data in the **DataSet** back to the data source.

i) The Connection object

Listed below are the common connection object methods we could work with:

A. **Open** - Opens the connection to our database
B. **Close** - Closes the database connection
C. **Dispose** - Releases the resources on the connection object. Used to force garbage collecting, ensuring no resources are being held after our connection is used.
D. **State** - Tells you what type of connection state your object is in, often used to check whether your connection is still using any resources. Ex. if (*ConnectionObject.State == ConnectionState.Open*)

ii) The Command Object

- **ExecuteReader** - Simply executes the SQL query against the database, using the *Read()* method to traverse through data.
- **ExecuteNonQuery** – Used whenever you work with SQL stored procedures with parameters.
- **ExecuteScalar** - Returns a lightning fast single value as an object from your database Ex. *object val = Command.ExecuteScalar();* Then check if != null.
- **ExecuteXmlReader** - Executes the SQL query against SQL Server only, while returning an XmlReader object.

- **Prepare** – Equivalent to ADO's Command. *Prepared = True property*. Useful in caching the SQL command so it runs faster when called more than once. Ex. *Command.Prepare();*
- **Dispose** – Releases the resources on the Command object. Used to force garbage collecting, ensuring no resources are being held after our connection is used.

iii) The DataReader Object

- **Read** – Moves the record pointer to the first row, which allows the data to be read by
column name or index position.
- **HasRows** - HasRows checks if any data exists, and is used instead of the Read method.
Ex. if (DataReader.HasRows).
- **IsClosed** - A method that can determine if the DataReader is closed.
- **Next Result** - Equivalent to ADO's NextRecordset Method, where a batch of SQL **statements** are executed with this method before advancing to the next set of data results.

- **Close** – Closes the DataReader

iv) The DataAdapter

Using an adapter, you can read, add, update, and delete records in a data source. To allow you to specify how each of these operations should occur, an adapter supports the following four properties:

- **SelectCommand** – reference to a command that retrieves rows from the data store.
- **InsertCommand** – reference to a command for inserting rows into the data store.
- **UpdateCommand** – reference to a command for modifying rows in the data store.
- **DeleteCommand** – reference to a command for deleting rows from the data store.

2. The DataSet

The ADO.NET DataSet is explicitly designed for data access independent of any data source. As a result, it can be used with multiple and differing data sources, used with XML data, or used to manage data local to the application.

The ADO.NET DataSet contains DataTableCollection and their DataRelationCollection . It represents a collection of data retrieved from the Data Source.

The **DataSet** contains a collection of one or more **DataTable** objects made up of rows and columns of data, as well as primary key, foreign key, constraint, and relation information about the data in the **DataTable** objects.

We can use Dataset in combination with **DataAdapter** class. The DataSet object offers a disconnected data source architecture. The Dataset can work with the data it contain, without knowing the source of the data coming from. That is, the Dataset can work with a disconnected mode from its Data Source . It gives a better advantage over **DataReader** , because the DataReader is working only with the connection oriented Data Sources.

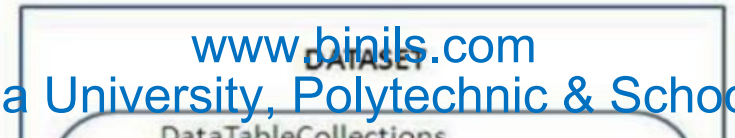
In any .NET data access page, before you connect to a database, you first have to import all the necessary namespaces that will allow you to work with the objects required. As we're going to work with SQL Server, we'll first import the namespaces we need. Namespaces in .NET are simply a neat and orderly way of organizing objects, so that nothing becomes ambiguous.

Note

(Namespaces: All the classes are defined in single name called namespaces in ASP.NET.)
Example:

```
1. <%@ Import Namespace="System" %> <BR>
2. <%@ Import Namespace="System.Data" %> <BR>
3. <%@ Import Namespace="System.Data.SqlClient" %>)
```

The Dataset contains the copy of the requested data. The Dataset contains more than one Table at a time. We can set up Data Relations between these tables within the

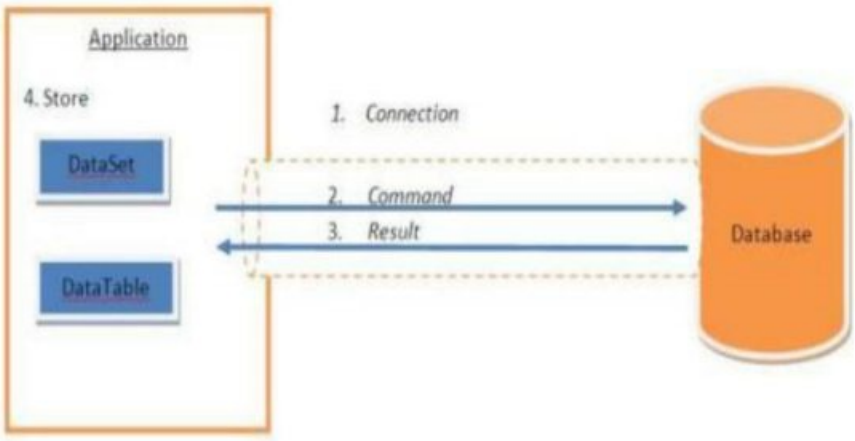


DataSet. The data set may comprise data for one or more members, corresponding to the number of rows.

In a typical situation requiring data access, we need to perform four major tasks:

1. Connecting to the database
2. Passing the request to the database, i.e., a command like select, insert, or update.
3. Getting back the results, i.e., rows and/or the number of rows effected.
4. Storing the result and displaying it to the user.

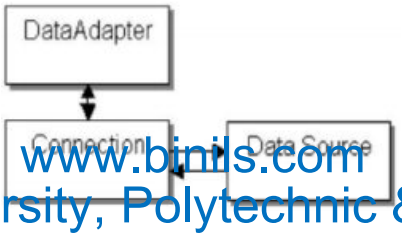
This can be visualized as:



4.1.4. The Connection Class

The ADO.NET Connection class is used to establish a connection to the database. The Connection class uses a ConnectionString to identify the database server location, authentication parameters, and other information to connect to the database. This ConnectionString is typically stored in the *web.config*.

The Connection class is also the heart of client-side transactions with ADO.NET, connection pooling, and schema tables with the OLE DB provider. Following figure shows the relationship between a connection, a data source, and a data adapter:



Each data provider has a Connection class. Below Table shows the name of various connection classes for data providers.

Table: Data provider connection classes

	Data Provider		Connection Class
1.	OldDB	1.	OleDbConnection
2.	Sql	2.	SqlConnection
3.	ODBC	3.	OdbcConnection
4.	Oracle	4.	OracleConnection

Connection String is a normal String representation which contains Database connection information to establish the connection between Database and the Application.

The Connection String includes parameters such as the '*name of the driver'*, '*Server name*' and '*Database name*', as well as security information such as '*user name*' and '*password*'.

Data providers use a connection string containing a collection of parameters to establish the connection with the database. Let's have some examples:

Microsoft SQL Server Connection String

```
SqlConnection conn = new SqlConnection(
    "Data Source=DatabaseServer;Initial Catalog=Northwind;User
        ID=YourUserID;Password=YourPassword");
```

OLEDB Data Provider Connection String

```
connetionString = "Provider=Microsoft.Jet.OLEDB.4.0;
 Data Source=yourdatabasename.mdb;";
 cnn = new OleDbConnection(connetionString);
```

ODBC Connection String

```
connetionString = "Driver={Microsoft Access Driver (*.mdb)};
DBQ=yourdatabasename.mdb;";
cnn = new OdbcConnection(connetionString);
```

Note: You have to provide the necessary information to the Connection String attributes.

When you have a connection string, you're ready to connect to your data source. A connection represents a live connection to the data source.

Let's have our first example of creating a connection. We are going to use the Northwind database in our examples.

First, import the "**System.Data.OleDb**" namespace. We need this namespace to work with Microsoft Access and other OLE DB database providers.

We create a dbconn variable as a new **OleDbConnection** class with a connection string which identifies the OLE DB provider and the location of the database. Then we open the database connection:

```
<%@ Import Namespace="System.Data.OleDb" %>
<script runat="server">
sub Page_Load
dim dbconn
dbconn=New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;
 data source=" & server.mappath("northwind.mdb"))
 dbconn.Open()
end sub
</script>
```

Now we have a connection ready with our database. Whenever we want to retrieve data, we just need to open the connection, perform the operation, and close the connection.

4.1.5. The Command Class

The Command class is provided by all standard ADO.NET providers, and it almost always encapsulates a **SQL statement** or a **stored procedure call** that can be executed against a data source.

Command objects can retrieve rows; directly insert, delete, or modify records; calculate totals and averages; alter the structure of a database; or fill a disconnected DataSet when used with a DataAdapter.

Connection object, which it uses to communicate with the data source and define a few key properties, such as **CommandText**(the stored procedure or embedded SQL command) and **CommandType**.

The Command class is provided in several provider-specific varieties, including SqlCommand and OleDbCommand.

To execute a Command, you use one of the Command object methods, including

- **ExecuteNonQuery( )**,
- **ExecuteReader( )**, and
- **ExecuteScalar( )**, depending on the type of Command.

Occasionally, a provider may define an additional method, such as the **ExecuteXmlReader( )** method offered by the SQL Server provider, which retrieves query results as an XML document.

Command objects commonly provide three methods that are used to execute commands on the database:

ExecuteNonQuery: Executes commands that have no return values such as INSERT, UPDATE or DELETE

ExecuteScalar: Returns a single value from a database query (return first column of the first row in the resultset.)

ExecuteReader: Returns a result set by way of a DataReader object

Let's have a simple code snippet using **ExecuteReader Method** to show how this works:

```
public void CallExecuteReader()
    {
        SqlConnection conn = new SqlConnection();
        conn.ConnectionString =
ConfigurationManager.ConnectionStrings["connString"].ConnectionString;

        try
        {
            SqlCommand cmd = new SqlCommand();
            cmd.Connection = conn;
            cmd.CommandText = "SELECT EMPNO,ENAME FROM EMP";
            cmd.CommandType = CommandType.Text;
            conn.Open();

            SqlDataReader reader =
cmd.ExecuteReader(CommandBehavior.CloseConnection);
            if (reader.HasRows)
            {
                while (reader.Read())
                {
                    MessageBox.Show("Employee No: " + reader["EMPNO"].ToString() + "
Name :" + reader["ENAME"].ToString());
                }
            }
             cmd.Dispose();
            conn.Dispose();
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }
```


4.1.6. The DataReader Class

The DataReader object represents a read-only, forward-only stream of data, which is ideal for quickly retrieving query results.

You can't create a **DataReader** directly. Instead, you must use the **ExecuteReader( )** method of a **Command** object **that returns a DataReader**.

Imagine these lines of code in a console application:

```
using System;
using System.Data;
using System.Data.SqlClient;

namespace Unit04
{
        class ReaderDemo
        {
                static void Main()
                {
                        ReaderDemo rd = new ReaderDemo();
                        rd.SimpleRead();
                }

                public void SimpleRead()
                {
                        // declare the SqlDataReader, which is used in
                        // both the try block and the finally block
                        SqlDataReader rdr = null;

                        // create a connection object
                        SqlConnection conn = new SqlConnection(
"Data Source=(local);Initial Catalog=Northwind;Integrated Security=SSPI");

                        // create a command object
                        SqlCommand cmd  = new SqlCommand(
                                "select * from Customers", conn);

                        try
                        {
                                // open the connection
                                conn.Open();

                                // 1. get an instance of the SqlDataReader
                                rdr = cmd.ExecuteReader();

                                // print a set of column headers
                                Console.WriteLine( "Contact Name      City       Company Name");
                                Console.WriteLine("------------          ------------       ------------");

                                // 2. print necessary columns of each
                record
                                while (rdr.Read())
                                {
                                        // get the results of each column
                                        string contact = (string)rdr["ContactName"];
                                        string company = (string)rdr["CompanyName"];
                                        string city    = (string)rdr["City"];
```

```
                        // print out the results
                        Console.Write("{0,-25}", contact);
                        Console.Write("{0,-20}", city);
                        Console.Write("{0,-25}", company);
                        Console.WriteLine();
                    }
                }
                finally
                {
                    // 3. close the reader
                    if (rdr != null)
                    {
                        rdr.Close();
                    }

                    // close the connection
                    if (conn != null)
                    {
                        conn.Close();
                    }
                }
            }
        }
}
```


4.1.7. The DataAdapter Class

The DataAdapter class serves as a bridge between a DataSet and a data source. The DataAdapter both retrieve a DataSet from a data source and updates any changes made to the DataSet back to the data source.

DataAdapter has several methods associated with it. Most commonly used methods among them are listed below:

Fill: Fill method is used to fetch records from the database and update them into the datatables of dataset. Uses SelectCommand for execution. Syntax for Fill method is :

```
SqlDataAdapter adapter = new SqlDataAdapter();
adapter.Fill(ds);
```

Here sampleAdapter is a SqlDataAdapter containing select query for Employee, employee set is the dataset and Employee is the database table.

FillSchema: FillSchema method is used to create an empty table in dataset containing the same schema as that of a specific table in the database. Constraints of the corresponding database table is also copied and reflected in the datatable of dataset. Uses SelectCommand for execution but copies only the schema of the table and not the data. Syntax for this method is shown below:

```
sampleAdapter.FillSchema(empDataSet, SchemaType.Source, "Employee");
```

Here empDataSet is the dataset and Employee is the database table name.

Update: Manipulated records of the dataset are updated back in the database using this method. Records that are inserted, updated and deleted from the dataset are pushed into the database using this method. Uses InsertCommand or UpdateCommand or DeleteCommand for the above mentioned purpose.

Syntax for Update method is shown below:

sampleAdapter.Update(employeeTable);

Before this statement, sampleAdapter will include an UpdateCommand.

employeeTable is the datatable of the dataset.

- **Dispose:** This method is used to release all resources used by the dataadapter.

Here is the syntax:

sampleAdapter.Dispose();

Dataadapters are used for two main purposes. They are listed below:

- To fetch data from the database and display them in the datatables of dataset. This is achieved using **Fill** method of dataadapter.
- To fetch data from dataset and update them in necessary tables of database. This is achieved using **Update** method of dataadapter.

There are four different types of Dataadapters. Their purpose and usage is mentioned below:

- **OleDbDataAdapter:** Dataadapters used to interact with databases through OLE DB provider. It belongs to "System.Data.OleDb" namespace.
- **SqlDataAdapter:** Dataadapters used to interact with SQL Server database with version 7.0 or higher through Tabular data services. It belongs to "System.Data.SqlClient" namespace.
- **OdbcDataAdapter:** Dataadapters used to interact with databases exposed by an ODBC provider. It belongs to "System.Data.Odbc" namespace.
- **OracleDataAdapter:** Dataadapters used to interact with Oracle database. It belongs to "System.Data.OracleClient" namespace.

Dataadapter has four different properties which controls database updates. The properties are "*SelectCommand*", "*UpdateCommand*", "*InsertCommand*" and "*DeleteCommand*". These commands are used to read, update, add and delete records from a database respectively.

4.1.8. The DataSet Class

The dataset is a disconnected, in-memory representation of data. It can be considered as a local copy of the relevant portions of the database. The DataSet is persisted in memory and the data in it can be manipulated and updated independent of the database. When the use of this DataSet is finished, changes can be made back to the central database for updating. The data in DataSet can be loaded from any valid data source like Microsoft SQL server database, an Oracle database or from a Microsoft Access database.

The structure of a "***System.Data.DataSet***" is similar to that of a relational database. It is organized in a hierarchical object mode of tables, rows, columns, constraints, and relationships.

The DataSet object is made up of two objects:

- **"*DataTableCollection*"** object containing null or multiple 'DataTable' objects (Columns, Rows, Constraints).
- "***DataRelationCollection***" object containing null or multiple 'DataRelation' objects which establish a parent/child relation between two 'DataTable' objects.

```
//Create a DataSet
    DataSet dset = new DataSet();
```

There are two types of DataSets:

1. **Typed DataSet**
2. **Untyped DataSet**

1. **Typed dataset** is similar to a dataset with only difference is that the schema is already present in typed dataset. So if any mismatch in the column will generate compile time errors rather than runtime error as in the case of normal dataset

   Example: Let us look into a small example which explain the Typed DataSet,

1. Using DataSet:


```
//Create DataAdapter
SqlDataAdapter daEmp = new SqlDataAdapter("SELECT empno, empname,
                                empaddress FROM EMPLOYEE",conn);
//Create a DataSet Object
DataSet dsEmp = new DataSet();

//Fill the DataSet
daEmp.Fill(dsEmp,"EMPLOYEE");

//Let us print first row and first column of the table
Console.Write(dsEmp.Tables["EMPLOYEE"].Rows[0][0].ToString());

//Assign a value to the first column
dsEmp.Tables["EMPLOYEE"].Rows[0][0] = "12345";
//This will generate runtime error as empno column is integer
```

If we observe above code we will get a runtime error when this code gets executed as the value assigned to the column (empno) does not take string value.

2. Using Typed DataSet:

```
//Create DataAdapter

    SqlDataAdapter daEmp = new SqlDataAdapter("SELECT empno, empname,
                empaddress FROM EMPLOYEE",conn);
```

//Create a DataSet Object. Note that an instance of the EmployeeDS class is //created: the class that maps to the EmployeeDS Schema and inherits from //the DataSet class, not the generic DataSet class itself.

EmployeeDS dsEmp = new EmployeeDS( );

//Fill the DataSet

daEmp.Fill(dsEmp,"EMPLOYEE");

//Let us print first row and first column of the table

Console.Write(dsEmp.EMPLOYEE[0].empno.ToString());

//Assign a value to the first column

dsEmp.EMPLOYEE[0].empno = "12345"; //This will generate compile time error.

If we see above code, a typed dataset is very much similar to a normal dataset. But the only difference is that the schema is already present for the same. Hence any mismatch in the column will generate compile time errors rather than runtime error as in the case of normal dataset. Also accessing the column value is much easier than the normal dataset as the column definition will be available in the schema.

2. An **Untyped dataset** is not defined by a schema, instead, you have to add tables, columns and other elements to it yourself, either by setting properties at design time or by adding them at run time.

//Create DataAdapter

SqlDataAdapter daEmp = new SqlDataAdapter("SELECT empno, empname, empaddress FROM EMPLOYEE",conn);

//Create a DataSet Object

DataSet dsEmp = new DataSet();

//Fill the DataSet

daEmp.Fill(dsEmp,"EMPLOYEE");

//Let us print first row and first column of the table

Console.Write(dsEmp.Tables["EMPLOYEE"].Rows[0][0].ToString());

//Assign a value to the first column

dsEmp.Tables["EMPLOYEE"].Rows[0][0] = 12345 ;

DataSets are memory structures that do not contain any data by default. DataSets will have to be filled with data. This can be done in several ways.

1) By calling the “Fill” method of **DataAdapter**. For example:

//Create DataAdapter

SqlDataAdapter daEmp = new SqlDataAdapter("SELECT empno, empname,empaddress FROM EMPLOYEE",conn);

//Create a DataSet Object

DataSet dsEmp = new DataSet();

//Fill the DataSet

daEmp.Fill(dsEmp,"EMPLOYEE");

2) Manually populate the DataSets by creating "**DataRow**" objects and call the AddNew method. For example:

DataSet dset;
DataTable dtbl;
DataRow drow;

//create a new row

drow=dtbl.NewRow();

//manipulate the newly added row using an index or the column name

drow["LastName"]="Altindag";
drow[1]="Altindag";

//After data is inserted into the new row, the Add method is used //to add the row to the

DataRowCollection dtbl.Rows.Add(drow);

//You can also call the Add method to add a new row by passing in an array of values, typed as Object

dtbl.Rows.Add(new object[] {1, "Altindag"});

1) Read an XML Document or stream into the dataset.
2) Copy the contents of one DataSet with another.
3) Copy the contents of one DataTable into a DataSet

A DataSet can store not only the data but also information about the data such as original, modified, inserted and deleted. Update of the underlying data-store is also possible. This can be done by calling the "**Update**" method of the TableDataAdapter or DataAdapter.

4.2. ACCESSING DATA USING DATA ADAPTERS AND DATASETS

ASP.NET includes features that enable you to add data access to your ASP.NET Web pages with little or no code. You can connect to databases, XML data and files, and business objects as data sources. You can then display data by using a variety of controls that provide great flexibility in how you present data on the page.

Datasets store a copy of data from the database tables. However, Datasets cannot directly retrieve data from Databases. DataAdapters are used to link Databases with DataSets.

DataSets < ----- DataAdapters < ----- DataProviders < ----- Databases

DataSets and **DataAdapters** are used to display and manipulate data from databases.

Reading Data into a Dataset

To read data into Dataset, you need to:

- Create a database connection and then a dataset object.

- Create a DataAdapter object and refer it to the DB connection already created.
- Note that every DataAdapter has to refer to a connection object. For example, **SqlDataAdapter** refers to **SqlDataConnection.**

The Fill method of **Data Adapter** has to be called to populate the Dataset object. The above mentioned steps are elaborated by the following examples :

Step 1): first create a connection to database. We would explore later that there is no need of opening and closing database connection explicitly while you deal with DataAdapter objects. All you have to do is, create a connection to database using the code like this:

SqlConnection con = new SqlConnection ("data source=localhost; uid= sa; pwd= abc; database=Northwind");

We would use Northwind database by using OleDbConnection.

The Code would Look like:
OleDbConnection con= new OleDbConnection ("Provider =Microsoft.JET.OLEDB.4.0;" + "Data Source=C:\\Program Files\\Microsoft Office\\Office\\Samples\\Northwind.mdb");

Step 2:) Now, create a Dataset object which would be used for storing and manipulating data.

DataSet myDataSet = new DataSet ("Northwind");

Since the name of source database is Northwind, we have passed the same name in the constructor.

Step 3:) The DataSet has been created but as we said before, this DataSet object cannot directly interact with Database. We need to create a DataAdapter object which would refer to the connection already created. The following line would declare a DataAdapter object:

OleDbAdapter myDataAdapter = new OleDbAdapter (CommandObject, con);

The above line demonstrates one of many constructors of OleDbAdapter class. This constructor takes a command object and a database connection object. The purpose of command object is to retrieve suitable data needed for populating DataSet. As we know SQL commands directly interacting with database tables, a similar command can be assigned to CommandObject.

*OleDbCommand CommandObject = new OleDbCommand ("Select * from employee");*

Whatever data you need for your Dataset should be retrieved by using suitable command here. The second argument of OleDbAdapter constructor is connection object con. Alternative approach for initializing DataAdapter object:

Place a null instead of CommandObject while you initialize the OleDbAdapter object:

OleDbAdapter myDataAdapter = new OleDbAdapter (null, con);

Then you assign your query to the CommandObject and write:

myDataAdapter.SelectCommand = CommandObject;

Step 4:) Now, the bridge between the DataSet and Database has been created. You can populate dataset by using the Fill command:

myDataAdapter.Fill (myDataSet, "EmployeeData");

The first argument to Fill function is the DataSet name which we want to populate. The second argument is the name of DataTable. The results of SQL queries go into DataTable. In this example, we have created a DataTable named EmployeeData and the values in this table would be the results of SQL query: "*Select * from employee*". In this way, we can use a dataset for storing data from many database tables.

Step 5:) DataTables within a Dataset can be accessed using Tables. To access EmployeeData, we need to write:

myDataSet.Tables["EmployeeData"].

To access rows in each Data Table, you need to write:

myDataSet.Tables["EmployeeData].Rows

The following code would combine all the steps we have elaborated so far.


```
<%@ Page Language= "C#" %>
<%@ Import Namespace= "System.Data" %>
<%@ Import Namespace= "System.Data.OleDb" %>
<html>
<body>
<table border=2>
<tr>
<td><b> Employee ID </b></td>
<td><b> Employee Name </b></td>
</tr>

<% OleDbConnection con= new OleDbConnection ("Provider
=Microsoft.JET.OLEDB.4.0;" + "Data Source=C:\\Program Files\\Microsoft
Office\\Office\\Samples\\Northwind.mdb");
    <%
        DataSet myDataSet = new DataSet();
        OleDbCommand CommandObject = new OleDbCommand ("Select * from
            employee");
    OleDbAdapter myDataAdapter = new OleDbAdapter (CommandObject, con);
    myDataAdapter.Fill (myDataSet, "EmployeeData");
foreach(DataRow dr in myDataSet.Tables["EmployeeData"].Rows)
{
    Response.write ("<tr>");
        for (int j = 0; j <2 ; j++)
        {
            Response.write ( "<td>" + dr[j].ToString() + "</td"> );
```

```
            }
    Response.write ("</tr>");
    %>
</table>
</body>
</html>
```

The Code above would iterate in all rows of Employee table and display ID and name of every employee.

for (int j = 0 ; j < dr.Table.Columns.Count ; j++)

As we said earlier, there is no need of opening and closing database connection explicitly. DataAdapter class handles both these functions.

4.2.1. WORKING WITH DATA GRID

The DataGrid control displays the fields of a data source as columns in a table. Each row in the control represents a record in the data source. The control supports selection, editing, deleting, paging, and sorting.

The DataGrid control with strong features is the most complicated control included within the ASP.NET framework. Like the Repeater and DataList controls, it enables to format and display records from a database table. However, it has several advanced features, such as support for sorting and paging through records, which makes it unique.

Records can be displayed in a DataGrid without using templates. A data source can be simply bound to the DataGrid, and it automatically displays the records. The following example, displays all the records from the Employees database table in a DataGrid

The following steps are used for Databinding with DataGridView in ADO.NET

DataGridView is very powerful and flexible control for displaying records in a tabular (row-column) form. Here I am describing a different way of databinding with a DataGridView control.

Take a windows Form Application -> take a DataGridView control. Follow the given steps.
Step 1 : Select DataGridView control and click at smart property.
Step 2 : After clicking, a pop-up window will be open.
Step 3 : Click ComboBox.
Step 4 : Click at Add Project Data Source. A new window will be opened to choose Data Source Type.
Step 5 : Choose Database (By default it is selected) and click the next button. A new window will be open to Database Model.
Step 6 : Select DataSet (By default it is selected) and click the next button. A new window will be open.
Step 7 : Click at New Connection button.
Step 8 : Write Server name, User name and Password of your SQL server and select Database name. Look at the following figure.
Step 9 : Click "ok" button. After clicking ok button, you will reach the Data Source Configuration Wizard.

Step 10 : Click the next button.
Step 11 : Click on Table to explore all tables of your Database.
Step 12 : Click on the selected Database table to explore all columns.
Step 13 : Check the CheckBox to select columns.
Step 14 : Click the Finish button. You will note that the DataGridView will show all columns of the table (Here, "Student_detail"). And Run the application.

Now we bind the DataGridView with database by code. Take another DataGridView control and write the following code on the form load event.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace DatabindingWithdataGridView
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }
      SqlDataAdapter dadapter;
      DataSet dset;
      string connstring = "server=.;database=student;user=sa;password=wintellect";
      private void Form1_Load(object sender, EventArgs e)
      {
         dadapter = new SqlDataAdapter("select * from student_detail", connstring);
         dset = new System.Data.DataSet();
         dadapter.Fill(dset);
         dataGridView1.DataSource = dset.Tables[0].DefaultView;
      }
   }
}
```


Creating Columns in a DataGrid Control

The DataGrid control displays the columns in a variety of ways. By default, the columns are generated automatically based on fields in the data source. However, in order to control the content and layout of columns more precisely, the following types of columns can be defined:

Type of Column	Description
Bound column	Allows specifying which data source field to display and specifies the format of that field, using a .NET formatting expression. For details see Adding Bound Columns to a DataGrid Web Server Control.
Hyperlink column	Displays information as hyperlinks. A typical use is to display data (such as a customer number or product name) as a hyperlink that users can click to navigate to a separate page that provides details about that item. For details see Adding Hyperlink Columns to a DataGrid Web Server Control.
Button column	Allows adding a button for each item in the grid and defining custom functionality for that button. For example, you might create a button labeled "Add to Shopping Cart" that runs your custom logic when a user clicks it. You can also add predefined buttons for Select, Edit, Update, Cancel, and Delete functions.
Edit, Update, Cancel column	Allows creating in-place editing. For more details, see "Editing Items" below.
Template column	Allows creating combinations of HTML text and server controls to design a custom layout for a column. The controls within a template column can be data-bound. Template columns gives great flexibility in defining the layout and functionality of the grid contents, because you have complete control over how the data is displayed and what happens when users interact with rows in the grid. For details see Adding Template Columns to a DataGrid Web Server Control.

Data Grid Events

The DataGrid control supports several events. One of them, the ItemCreated event, gives you a way to customize the itemcreation process. The ItemDataBound event also gives you the ability to customize the DataGrid items, but after the data is available for inspection.

For example, if you were using the DataGrid control to display a to-do list, you could display overdue items in red text, completed items in black text, and other tasks in green text.

The remaining events are raised in response to button or LinkButton clicked in grid items. They are designed to implement common data manipulation tasks. Four events of this type are supported:

- EditCommand
- DeleteCommand
- UpdateCommand
- CancelCommand

When the user clicks one of the buttons (labeled by default Edit, Delete, Update, or Cancel, respectively), the corresponding event is raised.

The DataGrid control also supports the ItemCommand event that is raised when a user clicks a button that is not one of the predefined buttons above. This event can be used for custom functions by setting a button's CommandName property to a value needed, and then testing for it in the ItemCommand event handler.

(For example, you could use this approach when selecting an item, as documented in allowing Users to Select Items in a DataList Web Server Control.) By default, a DataGrid simply displays all the columns from its data source. However, if False value is assigned to the DataGrid control's AutoGenerateColumns property, columns can be created individually to have more control over the formatting.

Adding a BoundColumn to a DataGrid

The default column used in a DataGrid is a BoundColumn. If only limited columns are to be displayed and controlled from a data source, declaration of one or more BoundColumns controls is done explicitly.The following example demonstrates it :-

Step 1: Drag GirdView from Toolbox on your design page.

Step 2: Check out the code of aspx page. It will be something like this

```
<asp :GridView ID="GridView1â€³ runat="server">
</asp:GridView>
```

Step 3: Now add one bound column, this way

```
<asp :GridView ID="GridView1â€³ runat="server"
AutoGenerateColumns="false">
<columns>
<asp :BoundField HeaderText="ColumnName" DataField="ColumnName" />
</columns>
</asp:GridView>
```


Note: Set the AutoGenerateColumns to false.

Note: HeaderText="ColumnName" is the name appearing as column heading in gridview.

Note: DataField="CoumnName" is the name of column returning from the SELECT query result.

Step 4: You can add one or more bound columns also

```
<asp :GridView ID="GridView1â€³ runat="server" AutoGenerateColumns="false">
<columns>
<asp :BoundField HeaderText="ColumnName" DataField="DataColumnName" />
<asp :BoundField HeaderText="ColumnName" DataField="DataColumnName" />
<asp :BoundField HeaderText="ColumnName" DataField="DataColumnName" />
</columns>
</asp:GridView>
```

Step 5: Example: Say your code behind class contains -

```
{
Strinf sql = "SELECT emp_name,emp_id,emp_salary,emp_email FROM .....";
```

```
SqlDataAdapter ad = new SqlDataAdapter(sql,connectionObject);
DataSet ds=new DataSet();
ad.Fill(ds);
GridView1.DataSource = ds;
GridView1.DataBind();
}
```

Now, in some situations you want some additional columns to retrieve for later use.

But, while binding the DataSet to GridView you want to display limited columns.

Here, you retieve columns: emp_name,emp_id,emp_salary and emp_email

But, you dont want to display email in gridview but want to use it in your code behind class somewhere.

So, you need to set the AutoGenerateColumns = false.

Then, you need to bound the columns which you want to bind to gridview.

```
<asp :GridView ID="GridView1â€³ runat="server" AutoGenerateColumns="false">
<columns>
<asp :BoundField HeaderText="Employee Name" DataField="emp_name" />
<asp :BoundField HeaderText="Employee Id" DataField="emp_id" />
<asp :BoundField HeaderText="Salary" DataField="emp_salary" />
</columns>
</asp:GridView>
```


Step 6: Run the application

4.3. Create an ADO.NET application– Using Stored Procedures

Stored Procedures are a set of sql commands which are compiled and are stored inside the database.

Sample application Using a Stored Procedure with a Command

Creating a sample application using a Stored Procedure with a Command object then we need to specify it as:

Initially create a object of SqlConnection class which is available in System.Data.SqlClient namespace. Open the connection using the Open() method.

```
SqlConnection con = new SqlConnection("Data Source=  ; initial  catalog= Northwind ;
User Id=  ; Password=  '");
 con.open();
```

Create the following stored procedure on the Region table in the Northwind database which accepts two parameters and does not have any output parameters.

```
CREATE PROCEDURE RegionUpdate (@RegionID INTEGER,
        @RegionDescription NCHAR(50)) AS
        SET NOCOUNT OFF
        UPDATE Region
        SET RegionDescription = @RegionDescription
```

Create a **SqlCommand** object with the parameters as the name of the stored procedure that is to be executed and the connection object con to which the command is to be sent for execution.

```
        SqlCommand command = new SqlCommand("RegionUpdate",con);
```

Change the command objects **CommandType** property to stored procedure.

```
            command.CommandType = CommandType.StoredProcedure;
```

Add the parameters to the command object using the Parameters collection and the **SqlParameter** class.


```
        command.Parameters.Add(new
SqlParameter("@RegionID",SqlDbType.Int,0,"RegionID"));
        command.Parameters.Add(new
SqlParameter("@RegionDescription",SqlDbType.NChar,50,"RegionDescription"));
```

Specify the values of the parameters using the **Value** property of the parameters

```
        command.Parameters[0].Value=4;
        command.Parameters[1].Value="SouthEast";
```

Execute the stored procedure using the **ExecuteNonQuery** method which returns the number of rows effected by the stored procedure.

```
         int i=command.ExecuteNonQuery();
```

Now let us see how to execute stored procedures which has output parameters and how to access the results using the output parameters.

Create the following stored procedure which has one output parameter.

```
ALTER PROCEDURE RegionFind(@RegionDescription NCHAR(50) OUTPUT,
@RegionID INTEGER )AS
SELECT @RegionDescription =RegionDescription from Region where
<A href="mailto:RegionID=@RegionID">RegionID=@RegionID</A>
```

The above stored procedure accepts **regionID** as input parameter and finds the **RegionDescription** for the RegionID input and results it as the output parameter.

```
SqlCommand command1 = new SqlCommand("RegionFind",con);
command1.CommandType = CommandType.StoredProcedure;
```

Add the paremeters to the command1

```
command1.Parameters.Add(new SqlParameter
("@RegionDescription",SqlDbType.NChar
,50,ParameterDirection.Output,false,0,50,"RegionDescription",DataRowVersion.Default,null));
command1.Parameters.Add(new SqlParameter("@RegionID" , SqlDbType.Int, 0 ,
"RegionID" ));
```

Observe that the parameter RegionDescription is added with the ParameterDirection as Ouput. specify the value for the input parameter RegionID.


```
command1.Parameters["@RegionID"].Value = 4;
```

Assign the UpdatedRowSource property of the SqlCommand object to UpdateRowSource.OutputParameters to indicate that data will be returned from this stored procedure via output parameters.

```
command1.UpdatedRowSource = UpdateRowSource.OutputParameters;
```

Call the stored procedure and access the RegionDescription for the RegionID using the value property of the parameter.

```
command1.ExecuteNonQuery();
string newRegionDescription =(string)
command1.Parameters["@RegionDescription"].Value;
```

Close the sql connection.

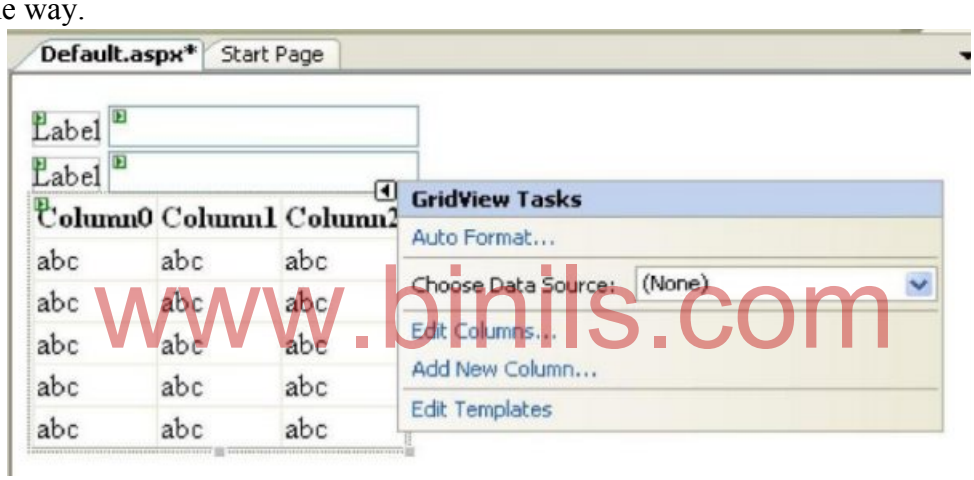
```
con.Close();
```

In the same way you can call the stored procedure that returns a set of rows by defining the parameters as appropriate and executing the command using ExecuteReader() that is used to traverse the records returned by the command.

Creating a Typed DataSet using Visual Studio .NET IDE

Follow these steps to create a small Web application by using Visual Studio .NET. The Web application uses a typed DataSet to display the results of an improvised SQL query in the **Northwind** database.

1. Start **Visual Studio .NET**.
2. Create a new Web Application project named "**TDS**" in Visual C# .NET.
3. Make sure that the "**Default.aspx**" page is open in the Editor window. If the page is not open, double-click "Default.aspx" in the Solution Explorer to open the page.
4. Under the **Editor** window, click **Design** to switch to **Design view**.
5. To open the **toolbox**, press CTRL+ALT+X. In the toolbox, click **Web Forms**. Select and drag the following to the upper-left corner of the page: two rows each of a label followed by a text box (positioned to the right of each label). Under these, add a "**GridView**" in the same way.



6. Click the top label. Press **F4** to display the **Properties window**. Change the 'Text' property to 'Title'. Click the other label, and then change its 'Text' property to 'YearReleased'.
7. To add a new **DataSet** to the project, press **CTRL+SHIFT+A**, and then click DataSet in the list of templates. Name the DataSet the following: "dsProducts.xsd". Click "Ok". Visual Studio will recommend placing the DataSet file inside the "**App_Code**" folder, which you should allow it to do for you. Note that the file is actually an XML Schema.
8. The "dsProducts.xsd" will open in design mode, and the "**TableAdapter Configuration Wizard**" will launch. For now, just click **Cancel**, as we will add tables by dragging them from the **Server Explorer**.
9. To create a **typed DataSet**, press **CTRL+ALT+S** to open the **Server Explorer**. We can start designing the dataset. Now defining the dataset can be done in two ways,
   A. Using the tool box
   B. Using the server explorer
10. Here we will use the second method. Now the first thing we must do is to Add a new Data Source to our project

    To add a Data Source:

    a. In "**Server Explorer**" right click on "**Data Connection**" Select "**New Connection**". This displays the **Add Connection dialog window**.

b. Enter the information to connect to your instance of SQL Server or MSDE and the **DVDCollectionDatabase.mdf**.
(*Generally this sample database file will be under structure "C:\Program files\ Microsoft Visual Studio8\Common7\IDE\ ProjectTempaltesCache\ CSharp\ Starter Kits\1033\ MovieCollection.zip\ DVDCollectionDatabase.mdf")*

c. Select **OK** to dismiss the dialog.

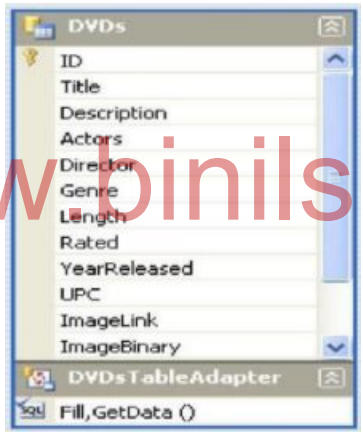
d. Select **Next**. This displays the Choose Your Database Objects page.

e. Note that you can choose from "Tables", "Views", "Stored Procedures", or "Functions".

f. Expand the "Tables" node and select the "DVDs" table. We will use all of the columns in the table, but you can select only those columns that you need for your application.

g. Select **Finish** to exit the wizard.

11. Drag the "DVDs" tables to your DataSet Designer window. The window should now resemble the screen shot below. What are we looking at? For each table we added, Visual Studio created a **strongly typed DataTable** (the name is based on the original table) and a **TableAdapter**. The **DataTable** has each column defined for us. The table adapter is the object we will use to fill the table. By default we have a **Fill()** method that will find every row from that table.



12. By default it will have a TableAdapter (**DVDsTableAdapter** here) with **Fill()** and **GetData()** methods that can be used to fill and fetch data from the database without implementing a line of code.

13. To write code to display the typed DataSet, double-click directly on the Web Form (not on a Web Control). The Web Form's codebehind appears, and the insertion point is inside the **Page_Load** event.

14. In the Page_Load event procedure, create a **Connection** object by passing the connection string to the default constructor of the SqlConnection class:

SqlConnection con =new SqlConnection(@"Data Source=.\SQLEXPRESS;AttachDbFilename=D:\Program Files\Microsoft Visual Studio 8\Common7\IDE\ProjectTemplatesCache\CSharp\StarterKits\1033\MovieCollection.zip\DVDCollectionDatabase.mdf;Integrated Security=True;User Instance=True");

15. Create a **SqlCommand** object that is then passed to the **SqlDataAdapter** object. Pass an improvised **SQL statement** and the new Connection object to the **SqlCommand**

constructor. The former sets the **CommandText** property of the new SqlCommand object. You can also pass the name of a stored procedure.

*SqlCommand cmd = new SqlCommand("select * from DVDs", con);*

16. Create an instance of the **SqlDataAdapter** object, passing the new **SqlCommand** object to the constructor:

SqlDataAdapter sda = new SqlDataAdapter(cmd);

17. Now you create the objects that are required to connect to the database and return data. The following is the code for the typed DataSet. Note that an instance of the **dsProducts** class is created: the class that maps to the **dsProducts** Schema and inherits from the **DataSet class**, not the generic DataSet class itself.

dsProducts tds = new dsProducts( );

18. Call the **Fill** method of the **SqlDataAdapter**, passing in the typed DataSet object and the DataSet's typed DataTable TableName property:

sda.Fill(tds, "DVDs");

19. To set the **Text** property of the text box controls to the strongly typed columns in the typed **DataSet's DataTable**, use the following format:

tds.DataTableName[RowIndex].ColumnName

For this sample application, the RowIndex is hard-coded to 5:

TextBox1.Text = tds.DVDs[5].Title;
TextBox2.Text = tds.DVDs[5].YearReleased;

Because the Rows collection is zero-based, when the page loads, note that the text box controls display the product and category names of the item in the sixth row of the GridView.

20. To display all of the results in the GridView1, set the **DataSource** property of the GridView1 to the new typed DataSet, and call DataBind():

GridView1.DataSource = tds;
GridView1.DataBind();

Complete Code Listing (Default.aspx):

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" > <head runat="server">
<title>Untitled Page</title>
</head>
<body>
<form id="form1" runat="server">
<div>
<asp:Label ID="Label1" runat="server" Text="Title"></asp:Label> <asp:TextBox
ID="TextBox1" runat="server"></asp:TextBox> <br />
```

```
<asp:Label ID="Label2" runat="server" Text="Releasing Year"></asp:Label> <asp:TextBox
ID="TextBox2" runat="server"></asp:TextBox> <asp:GridView ID="GridView1"
runat="server"> </asp:GridView>
</div>
</form>
</body>
</html>
```

Complete Code Listing (Default.aspx.cs):

```
public partial class _Default : System.Web.UI.Page {
 protected void Page_Load(object sender, EventArgs e)
 {
 dsProducts tds = new dsProducts();

 SqlConnection con = new SqlConnection(@"Data Source=.\SQLEXPRESS;
 AttachDbFilename=D:\Program Files\Microsoft Visual Studio 8\ Common7\ IDE\
 ProjectTemplatesCache \CSharp \Starter Kits\
 1033\MovieCollection.zip\DVDCollectionDatabase.mdf; Integrated Security=True;User
 Instance=True");

 SqlDataAdapter sda = new SqlDataAdapter("select * from DVDs", con);

 sda.Fill(tds, "DVDs");
 TextBox1.Text = tds.DVDs[5].Title;
 TextBox2.Text = tds.DVDs[5].YearReleased;
 GridView1.DataSource = da;
 GridView1.DataBind();
 }
 }
```

Review Questions
UNIT - IV

Part A (2 marks)

1. What is ADO.Net?
2. What is a connected model?
3. What is a Dataset?
4. List the contents of a Dataset.
5. Write the Connection string for ODBC.
6. What is the use of Fillschema method of DataAdapter?
7. What is a data grid?

Part B (3 marks)

1. What are the key components of ADO.NET object model? Explain.
2. What is the use of Command object?
3. What is the use of DataReader?
4. List the methods of Connection object and explain.
5. List the methods of Command object and explain.
6. List and discuss the purpose and usage of different DataAdapters.
7. Write notes on: Dataset class
8. What are the importance of Resultset?
9. List the steps to read data into Dataset.
10. What are the operations that can be performed on Data grid?
11. Mention the advantages of ADO.NET
12.

Part C (5 marks)

1. Draw the block diagram of Dataset and explain the steps to access data from a data base.
2. What is Connection class? Discuss he different Data providers and the corresponding Connection class.
3. Discuss the importance of Command object in data base access,
4. Explain the two types of Datasets in detail.
5. Explain with code the procedure of reading data into Dataset.
6. Explain the procedure of displaying data in Datagrid.
7. List the advantages of using Dataset and DataAdapter over DataReader in the development of Database application.

****

UNIT V

Objective

- To study what is VXML and its advantages
- To study about HTML and XML and difference between them
- To study how to browsing and parsing happens in XML
- How to create XML file and wellformed XML document with its attributes and entities
- To learn what is DTD and how to declare namespaces and need for XML schema and its uses
- What are building blocks and elements of XML
- To study about XML serialization in .NET framework and Fundamentals of SOAP, how to use SOAP with .NET Framework

5.1 Introduction

XML stands for Extensible Markup Language. It is a text-based markup language derived from Standard Generalized Markup Language (SGML). XML tags identify the data and are used to store and organize the data, rather than specifying how to display it like HTML tags, which are used to display the data.

XML adopts many successful features of HTML. There are three important characteristics of XML that make it useful in a variety of systems and solutions:

1. XML is extensible: XML allows you to create your own self-descriptive tags or language, that suits your application.
2. XML carries the data, does not present it: XML allows you to store the data irrespective of how it will be presented.
3. XML is a public standard: XML was developed by an organization called the World Wide Web Consortium (W3C) and is available as an open standard.

Is XML a Programming Language?

A programming language consists of grammar rules and its own vocabulary which is used to create computer programs. These programs instruct the computer to perform specific tasks. XML does not qualify to be a programming language as it does not perform any computation or algorithms. It is usually stored in a simple text file and is processed by special software that is capable of interpreting XML.

The following are the list of uses of XML.

i. XML can work behind the scene to simplify the creation of HTML documents for large web sites.
ii. XML can be used to exchange the information between organizations and systems
iii. XML can be used for offloading and reloading of databases

iv. XML can be used to store and arrange the data, which can customize your data handling needs.
v. XML can easily be merged with style sheets to create almost any desired output.
vi. Virtually, any type of data can be expressed as an XML document.

5.1.1 Advantages

XML is widely used in the era of web development. It is also used to simplify data storage and data sharing.

The main features or advantages of XML are given below.

1) XML separates data from HTML

If you need to display dynamic data in your HTML document, it will take a lot of work to edit the HTML each time the data changes. With XML, data can be stored in separate XML files. This way you can focus on using HTML/CSS for display and layout, and be sure that changes in the underlying data will not require any changes to the HTML.

With a few lines of JavaScript code, you can read an external XML file and update the data content of your web page.

2) XML simplifies data sharing

In the real world, computer systems and databases contain data in incompatible formats. XML data is stored in plain text format. This provides a software- and hardware-independent way of storing data. This makes it much easier to create data that can be shared by different applications.

3) XML simplifies data transport

One of the most time-consuming challenges for developers is to exchange data between incompatible systems over the Internet. Exchanging data as XML greatly reduces this complexity, since the data can be read by different incompatible applications.

4) XML simplifies Platform change

Upgrading to new systems (hardware or software platforms), is always time consuming. Large amounts of data must be converted and incompatible data is often lost. XML data is stored in text format. This makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.

5) XML increases data availability

Different applications can access your data, not only in HTML pages, but also from XML data sources. With XML, your data can be available to all kinds of "reading machines" (Handheld computers, voice machines, news feeds, etc), and make it more available for blind people, or people with other disabilities.

6) XML can be used to create new internet languages

A lot of new Internet languages are created with XML. Here are some examples:

- XHTML
- WSDL for describing available web services
- WAP and WML as markup languages for handheld devices
- RSS languages for news feeds
- RDF and OWL for describing resources and ontology
- SMIL for describing multimedia for the web

5.1.2 HTML Vs XML

XML is not a replacement for HTML. In fact, they are used together. XML and HTML were designed with different goals:

- XML was designed to carry data - with focus on what data is
- HTML was designed to display data - with focus on how data looks
- XML tags are not predefined like HTML tags are
- HTML is about displaying information, whereas XML is about describing information
- XML is extensible and it does Not Use Predefined Tags. HTML works with predefined tags like <p>, <h1>, <table>, etc.

```
<HTML>
<BODY>
<H1> Seshasayee Institute of Technology </H1>
<H2> ISO Certified || Autonomous Institution </H2>
<H3> Trichy </H3>
</BODY>
</HTML>
```

-
```
<?xml version="1.0" encoding="UTF-8" ?>
<College>
<H1> Seshasayee Institute of Technology </H1>
<H2> ISO 9001:2008  Certified || Autonomous Institution </H2>
<H3> Trichy </H3>
</College>
```

HTML	XML
HTML is an abbreviation for Hyper Text Markup Language.	XML stands for Extensible Markup Language.
HTML was designed to display data with focus on how data looks.	XML was designed to be a software and hardware independent tool used to transport and store data, with focus on what data is.
HTML is a markup language itself.	XML provides a framework for defining markup languages.
HTML is a presentation language.	XML is neither a programming language nor a presentation language.
HTML is case insensitive.	XML is case sensitive.
HTML is used for designing a web page to be rendered on the client side.	XML is used basically to transport data between the application and the database.
HTML has it own predefined tags.	While what makes XML flexible is that custom tags can be defined and the tags are invented by the author of the XML document.
HTML is not strict if the user does not use the closing tags.	XML makes it mandatory for the user the close each tag that has been used.

5.1.3 Browsing and parsing XML

Browsing Objects in XML (BOX) is a tool for reading Unified Modeling Language (UML) models represented in XML Metadata Interchange(XMI) format and exporting them to vector graphics formats, including Vector Markup Language (VML) and Scalable Vector Graphics (SVG).

A **parser** is a piece of program that takes a physical representation of some data and converts it into an in-memory form for the program as a whole to use. **Parsers** are used everywhere in software. An **XML Parser** is a **parser** that is designed to read **XML** and create a way for programs to use **XML**.

An XML parser is the piece of software that reads XML files and makes the information from those files available to applications and programming languages, usually through a known interface like the DOM

It is designed to read XML and create a way for programs to use XML. There are different types, and each has its advantages. Unless a program simply and blindly copies the whole XML file as a unit, every program must implement or call on an XML parser.

5.1.4 Creating a XML

XML is designed to

• Separate syntax from semantics to provide a common framework for

structuring information
• Allow self-made markup for any imaginable application domain.
• Support internationalization (Unicode) and platform independence.
• Be the future of structured information, including databases.

Syntax of the XML document

Let us discuss the syntax of XML document with the help of the following example

<?xml version”1.0”?>
<note>
<to> HOD </to>
<from>Principal </from>
<heading>Reminder </heading>
<body> This Saturday we have meeting at 10.00 AM </body>
</note>

The first line in the document is the XML declaration and it should always be included. It defines the XML version of the document. In this case, the document conforms to the 1.0 specification of XML:

<?xml version”1.0”?>

The next line defines the first element of the document called the root the element

<note>

The next line defined 4 child elements of the root: to, form,heading andbody.

<to> HOD </to>
<from>Principal </from>
<heading>Reminder </heading>
<body> This Saturday we have meeting at 10.00 AM </body>

The last line defines the end of the root element:

</note>

In HTML, some elements do not have a closing tag. The code is legal in html:

<p>this is paragraph
<p>this is another paragraph

Moreover XML tags are case sensitive. The tag <Letter> is different from the tag<letter>. Opening and closing tags must therefore be written with the same case. For example,

<Message> This is incorrect </message>
<message> This is correct </message>

In addition to that all XML elements must be properly nested.

This text is bold and italic

All XML documents must have root tag. All XML documents must contain a single tag pair to define the root element. All other elements must be nested within the root element. All elements can have sub (Children) elements. Sub element must be in pairs and correctly nested within their parent element.
<root>
<child>

<subchild>
</subchild>
</child>
</root>
Another thing that must be remembered is that, the attribute values must be included within quotes. XML elements can have attributes in name/value pairs just like in HTML.

<?xml version=”1.0”?>
<note date=”01/10/2016”>
<to>HOD</to>
<from>Principal </from>
<heading Reminder</heading>
<body> This Saturday we have meeting at 10.00 AM </body>
</note>

5.1.5 Details and well formed XML document

In XML, a valid document must conform to the rules in its DTD (Document Type Definition) or schema, which defines what elements can appear in the document and how elements may nest within one another. If a document isn’t well-formed it doesn’t go far in the XML world so you need to play by some very basic rules when creating an XML document. A well-formed document must have these components:

- All beginning and ending tags match up. In other words, opening and closing parts must always contain the same name in the same case: <tag> . . . </tag>or <TAG> . . . </TAG>, but not <tag> . . . </TAG>.
- Empty elements follow special XML syntax, for example, <empty_element/>.
- All attribute values occur within single or double quotation

  marks: <elementid=”value”> or <element id=’value’>.
- Non DTD XML files must use the predefined character entities for amp(&), apos(single quote), gt(>), lt(<), quot(double quote).
- It must follow the ordering of the tag. i.e., the inner tag must be closed before closing the outer tag.

- Each of its opening tags must have a closing tag or it must be a self ending tag.(<title>....</title> or <title/>).
- It must have only one attribute in a start tag, which needs to be quoted.
- amp(&), apos(single quote), gt(>), lt(<), quot(double quote)entities other than these must be declared.

Example of well-formed XML document:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE address
[
  <!ELEMENT address (name,company,phone)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT company (#PCDATA)>
  <!ELEMENT phone (#PCDATA)>
]>
<address>
  <name>Tanmay Patil</name>
  <company>TutorialsPoint</company>
  <phone>(011) 123-4567</phone>
</address>
```


Above example is said to be well-formed as:

- It defines the type of document. Here, the document type is **element**type.
- It includes a root element named as **address**.
- Each of the child elements among name, company and phone is enclosed in its self explanatory tag.
- Order of the tags is maintained.

5.1.6 XML Components

XML components are elements, entities, comments, attributes and processing instructions which will be discussed detail in following topics

5.1.7 Elements

XML elements can be defined as building blocks of an XML. Elements can behave as containers to hold text, elements, attributes, media objects or all of these. Each XML document contains one or more elements, the scope of which are either delimited by start and end tags, or for empty elements, by an empty-element tag.

Following is the syntax to write an XML element:

<element-name attribute1 attribute2>

....content

</element-name>

where

element-name is the name of the element. The *name* its case in the start and end tags must match.

attribute1, attribute2 are attributes of the element separated by white spaces. An attribute defines a property of the element. It associates a name with a value, which is a string of characters. An attribute is written as:

name = "value"

name is followed by an = sign and a string *value* inside double(" ") or single(' ') quotes.

Empty Element

An empty element (element with no content) has following syntax:

<name attribute1 attribute2.../>

Example of an XML document using various XML element:


```
<?xml version="1.0"?>
<contact-info>
   <address category="residence">
      <name>Tanmay Patil</name>
      <company>TutorialsPoint</company>
      <phone>(011) 123-4567</phone>
   <address/>
</contact-info>
```

XML Elements Rules

Following rules are required to be followed for XML elements:

An element *name* can contain any alphanumeric characters. The only punctuation mark allowed in names are the hyphen (-), under-score (_) and period (.).

Names are case sensitive. For example, Address, address, and ADDRESS are different names.

Start and end tags of an element must be identical.

An element, which is a container, can contain text or elements as seen in the above example.

5.1.8 Entities

Before we understand the Character Entities, let us first understand what an XML entity is.

As put by W3 Consortium the definition of entity is as follows:

The document entity serves as the root of the entity tree and a starting-point for an XML processor.

This means, entities are the placeholders in XML. These can be declared in the document prolog or in a DTD. There are different types of entities and this chapter will discuss Character Entity.

Both, the HTML and the XML, have some symbols reserved for their use, which cannot be used as content in XML code. For example, < and > signs are used for opening and closing XML tags. To display these special characters, the character entities are used.

There are few special characters or symbols which are not available to be typed directly from keyboard. Character Entities can be used to display those symbols/special characters also.

Types of Character Entities

There are three types of character entities:

Predefined Character Entities

Numbered Character Entities

Named Character Entities

Predefined Character Entities

They are introduced to avoid the ambiguity while using some symbols. For example, an ambiguity is observed when less than (<) or greater than (>) symbol is used with the angle tag(<>). Character entities are basically used to delimit tags in XML. Following is a list of pre-defined character entities from XML specification. These can be used to express characters without ambiguity.

Ampersand: &

Single quote: '

Greater than: >

Less than: <

Double quote: "

Numeric Character Entities

The numeric reference is used to refer to a character entity. Numeric reference can either be in decimal or hexadecimal format. As there are thousands of numeric references available, these are a bit hard to remember. Numeric reference refers to the character by its number in the Unicode character set.

General syntax for decimal numeric reference is:

&# decimal number ;

General syntax for hexadecimal numeric reference is:

&#x Hexadecimal number ;

The following table lists some predefined character entities with their numeric values:

Entity name	Character	Decimal reference	Hexadecimal reference
quot	"	"	"
amp	&	&	&
apos	'	'	'
lt	<	<	<
gt	>	>	>

Named Character Entity

As its hard to remember the numeric characters, the most preferred type of character entity is the named character entity. Here, each entity is identified with a name.

For example:

'Aacute' represents capital Á character with acute accent.

'ugrave' represents the small ù with grave accent.

5.1.9 Comments

XML comments are similar to HTML comments. The comments are added as notes or lines for understanding the purpose of an XML code.

Comments can be used to include related links, information and terms. They are visible only in the source code; not in the XML code. Comments may appear anywhere in XML code.

Syntax

XML comment has following syntax:

```
<!-------Your comment----->
```

A comment starts with <!-- and ends with -->. You can add textual notes as comments between the characters. You must not nest one comment inside the other.

Example

Following example demonstrates the use of comments in XML document:

```
<?xml version="1.0" encoding="UTF-8" ?>

<!---Students grades are uploaded by months---->

<class_list>

  <student>

    <name>Tanmay</name>

    <grade>A</grade>

  </student>

</class_list>
```


Any text between <!-- and --> characters is considered as a comment.

XML Comments Rules

Following rules are needed to be followed for XML comments:

- Comments cannot appear before XML declaration.
- Comments may appear anywhere in a document.
- Comments must not appear within attribute values.
- Comments cannot be nested inside the other comments.

5.1.10 Processing instructions

It describes the Processing Instructions (PIs). As defined by the XML 1.0 Recommendation,

"Processing instructions (PIs) allow documents to contain instructions for applications. PIs are not part of the character data of the document, but MUST be passed through to the application.

Processing instructions (PIs) can be used to pass information to applications. PIs can appear anywhere in the document outside the markup. They can appear in the prolog, including the document type definition (DTD), in textual content, or after the document.

Syntax

Following is the syntax of PI:

<?target instructions?>

Where:

target - identifies the application to which the instruction is directed.

instruction - it is a character that describes the information for the application to process.

A PI starts with a special tag <? and ends with ?>. Processing of the contents ends immediately after the string ?> is encountered.

Example

PIs are rarely used. They are mostly used to link XML document to a style sheet. Following is an example:

<?xml-stylesheet href="tutorialspointstyle.css" type="text/css"?>

Here, the *target* is xml stylesheet. *href="tutorialspointstyle.css"* and *type="text/css"* are *data* or *instructions* that the target application will use at the time of processing the given XML document.

In this case, a browser recognizes the target by indicating that the XML should be transformed before being shown; the first attribute states that the type of the transform is XSL and the second attribute points to its location.

Processing Instructions (PI) Rules

A PI can contain any data except the combination ?>, which is interpreted as the closing delimiter. Here are two examples of valid PIs:

<?welcome to pg=10 of tutorials point?>

<?welcome?>

5.1.11 Attributes

Attributes are part of the XML elements. An element can have multiple unique attributes. Attribute gives more information about XML elements. To be more precise, they define properties of elements. An XML attribute is always a *name-value* pair.

Syntax

An XML attribute has following syntax:

<element-name attribute1 attribute2 >

....content..

< /element-name>

where *attribute1* and *attribute2* has the following form:

name = "value"

value has to be in double (" ") or single (' ') quotes. Here, *attribute1* and *attribute2* are unique attribute labels.

Attributes are used to add a unique label to an element, place the label in a category, add a Boolean flag, or otherwise associate it with some string of data. Following example demonstrates the use of attributes:

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE garden [

  <!ELEMENT garden (plants)*>

  <!ELEMENT plants (#PCDATA)>

  <!ATTLIST plants category CDATA #REQUIRED>

]>

<garden>

  <plants category="flowers" />

  <plants category="shrubs">

  </plants>

</garden>
```


Attributes are used to distinguish among elements of the same name. When you do not want to create a new element for every situation. Hence, use of an attribute can add a little more detail in differentiating two or more similar elements.

In the above example, we have categorized the plants by including attribute *category* and assigning different values to each of the elements. Hence we have two categories of *plants*, one *flowers* and other *color*. Hence we have two plant elements with different attributes.

You can also observe that we have declared this attribute at the beginning of the XML.

Attribute Types

Following table lists the type of attributes:

Attribute Type	Description
StringType	It takes any literal string as a value. CDATA is a StringType. CDATA is character data. This means, any string of non-markup characters is a legal part of the attribute.
TokenizedType	This is more constrained type. The validity constraints noted in the grammar are applied after the attribute value is normalized. The TokenizedType attributes are given as: ID : It is used to specify the element as unique. IDREF : It is used to reference an ID that has been named for another element. IDREFS : It is used to reference all IDs of an element. ENTITY : It indicates that the attribute will represent an external entity in the document. ENTITIES : It indicates that the attribute will represent external entities in the document. NMTOKEN : It is similar to CDATA with restrictions on what data can be part of the attribute. NMTOKENS : It is similar to CDATA with restrictions on what data can be part of the attribute.
EnumeratedType	This has a list of predefined values in its declaration. out of which, it must assign one value. There are two types of enumerated attribute: **NotationType :** It declares that an element will be referenced to a NOTATION declared somewhere else in the XML document. **Enumeration :** Enumeration allows you to define a specific list of values that the attribute value must match.

Element Attribute Rules

Following are the rules that need to be followed for attributes:

An attribute name must not appear more than once in the same start-tag or empty-element tag.

An attribute must be declared in the Document Type Definition (DTD) using an Attribute-List Declaration.

Attribute values must not contain direct or indirect entity references to external entities.

The replacement text of any entity referred to directly or indirectly in an attribute value must not contain either less than sign <

5.2 DTD

A document type definition (**DTD**) is a set of markup declarations that define a document type for an SGML-family markup language (SGML, XML, HTML). A Document Type Definition (**DTD**) defines the legal building blocks of an XML document. It defines the document structure with a list of legal elements and attributes.

5.2.1 Declarations in DTD

The document type (DOCTYPE) declaration consists of an internal, or references an external Document Type Definition (DTD). It can also have a combination of both internal and external DTDs. The DTD defines the constraints on the structure of an XML document. It declares all of the document's element types, children element types, and the order and number of each element type. It also declares any attributes, entities, notations, processing instructions, comments, and PE references in the document.

The Internal DTD:


```
<!DOCTYPE root_element [

Document Type Definition (DTD):
  elements/attributes/entities/notations/
  processing instructions/comments/PE references
]>
```

Example:

```
<?xml version="1.0" standalone="yes" ?>

<!--open the DOCTYPE declaration -
  the open square bracket indicates an internal DTD-->
<!DOCTYPE foo [

<!--define the internal DTD-->
  <!ELEMENT foo (#PCDATA)>

<!--close the DOCTYPE declaration-->
]>

<foo>Hello World.</foo>
```

Rules:

- The document type declaration must be placed between the XML declaration and the first element (root element) in the document.
- The keyword DOCTYPE must be followed by the name of the root element in the XML document.
- The keyword DOCTYPE must be in upper case.

The External DTD:

External DTDs are useful for creating a common DTD that can be shared between multiple documents. Any changes that are made to the external DTD automatically updates all the documents that reference it. There are two types of external DTDs: private, and public.

Rules:

- If any elements, attributes, or entities are used in the XML document that are referenced or defined in an external DTD, standalone="no" must be included in the XML declaration.

"Private" External DTDs:

Private external DTDs are identified by the keyword SYSTEM, and are intended for use by a single author or group of authors.

```
<!DOCTYPE root_element SYSTEM "DTD_location">
```


where:

- DTD_location: relative or absolute URL

Example:

```
<!--inform the XML processor
  that an external DTD is referenced-->
<?xml version="1.0" standalone="no" ?>

<!--define the location of the
  external DTD using a relative URL address-->
<!DOCTYPE document SYSTEM "subjects.dtd">

<document>
  <title>Subjects available in Mechanical Engineering.</title>
  <subjectID>2303</subjectID>
    <subjectname>Fluid Mechanics</subjectname>
    <prerequisite>
      <subjectID>1001</subjectID>
      <subjectname>Mathematics</subjectname>
    </prerequisite>
```

```
  <classes>4 hours per week (lectures and tutorials) for one
    semester.</classes>
  <assessment>tutorial assignments and one 2hr exam at end of
    course.</assessment>
  <syllabus>
    Fluid statics. The Bernoulli equation. Energy equation. Momentum
    equation. Differential Continuity equation. Differential Energy
    equation. Differential Momentum equation. Dimensional Analysis.
    Similitude. Laminar flow. Turbulent flow. Lift and Drag. Boundary
    layer theory.
  </syllabus>
  <textbooks>
    <author>Foobar</author>
    <booktitle>The Study of Fluid Mechanics</booktitle>
  </textbooks>
</document>
```

The external DTD ("subjects.dtd") referenced in the example above contains information about the XML document's structure:

subjects.dtd:

```
<!--see Element Type Declarations
  for an explanation of the following syntax-->
<!ELEMENT document
  (title*,subjectID,subjectname,prerequisite?,
  classes,assessment,syllabus,textbooks*)>
<!ELEMENT prerequisite (subjectID,subjectname)>
<!ELEMENT textbooks (author,booktitle)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT subjectID (#PCDATA)>
<!ELEMENT subjectname (#PCDATA)>
<!ELEMENT classes (#PCDATA)>
<!ELEMENT assessment (#PCDATA)>
<!ATTLIST assessment assessment_type (exam | assignment) #IMPLIED>
<!ELEMENT syllabus (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT booktitle (#PCDATA)>
```


"Public" External DTDs:

Public external DTDs are identified by the keyword PUBLIC and are intended for broad use. The "DTD_location" is used to find the public DTD if it cannot be located by the "DTD_name".

```
<!DOCTYPE root_element PUBLIC "DTD_name" "DTD_location">
```

where:

- DTD_location: relative or absolute URL
- DTD_name: follows the syntax:

```
"prefix//owner_of_the_DTD//
 description_of_the_DTD//ISO 639_language_identifier"
```

Example:

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
  "http://www.w3.org/TR/REC-html40/loose.dtd">
<HTML>
<HEAD>
<TITLE>A typical HTML file</TITLE>
</HEAD>
<BODY>
  This is the typical structure of an HTML file. It follows
  the notation of the HTML 4.0 specification, including tags
  that have been deprecated (hence the "transitional" label).
</BODY>
</HTML>
```

5.2.1.1 Elements

Elements are the **main building blocks** of both XML and HTML documents.

Examples of HTML elements are "body" and "table". Examples of XML elements could be "note" and "message". Elements can contain text, other elements, or be empty. Examples of empty HTML elements are "hr", "br" and "img".

Examples:

<body>some text</body>

<message>some text</message>

5.2.1.2 Attributes

Attributes provide **extra information about elements**.

Attributes are always placed inside the opening tag of an element. Attributes always come in name/value pairs. The following "img" element has additional information about a source file:

The name of the element is "img". The name of the attribute is "src". The value of the attribute is "computer.gif". Since the element itself is empty it is closed by a " /".

5.2.1.3 Entity and notation

Some characters have a special meaning in XML, like the less than sign (<) that defines the start of an XML tag.

Most of you know the HTML entity: " ". This "no-breaking-space" entity is used in HTML to insert an extra space in a document. Entities are expanded when a document is parsed by an XML parser.

The following entities are predefined in XML:

Entity References	Character
<	<
>	>
&	&
"	"
'	'

5.2.2 Construction of an XML document

XML documents are constructed based on the DTD which is used to create an well formed XML document. XML document should follow all the rules and syntax of the DTD format.

5.2.3 XML Namespaces

A Namespace is a set of unique names. Namespace is a mechanisms by which element and attribute name can be assigned to group. The Namespace is identified by URI(Uniform Resource Identifiers).

5.2.4 Declaring namespaces

A Namspace is declared using reserved attributes. Such an attribute name must either be **xmlns** or begin with **xmlns:** shown as below:

```
<element xmlns:name="URL">
```

Syntax

- The Namespace starts with the keyword **xmlns**.
- The word **name** is the Namespace prefix.
- The **URL** is the Namespace identifier.

Example

Namespace affects only a limited area in the document. An element containing the declaration and all of its descendants are in the scope of the Namespace. Following is a simple example of XML Namespace:

```
<?xml version="1.0" encoding="UTF-8"?>
<cont:contact xmlns:cont="www.tutorialspoint.com/profile">
   <cont:name>Tanmay Patil</cont:name>
   <cont:company>TutorialsPoint</cont:company>
   <cont:phone>(011) 123-4567</cont:phone>
</cont:contact>
```

Here, the Namespace prefix is **cont**, and the Namespace identifier (URI) as *www.tutorialspoint.com/profile*. This means, the element names and attribute names with the **cont** prefix (including the contact element)

5.2.5 Default namespaces

A "default namespace" is a namespace declaration that does not use a namespace prefix. The scope of the default namespace is the element for which the namespace was declared and the related content, just as with the namespace scope discussed earlier. The benefit of using a default namespace is that the namespace prefix can be omitted.

For example, when adding a new namespace to an existing XML document, writing a namespace prefix for each element to which the new namespace will be applied involves a tremendous amount of tedious work. The larger the XML document, the greater the labor involved, and the greater the likelihood of notation errors. In this type of situation, adding only a default namespace declaration to the XML document in question eliminates the need to write a namespace prefix for each and every element, saving a lot of time.

On the other hand, there are drawbacks. One drawback is that omitting the namespace prefix makes it more difficult to understand which element belongs to which namespace, and which namespace is applicable. In addition, programmers should remember that when a default namespace is declared, the namespace is applied only to the element, and not to any attributes.

5.2.6 XML Schema

XML Schema is commonly known as XML Schema Definition (XSD). It is used to describe and validate the structure and the content of XML data. XML schema defines the elements, attributes and data types. Schema element supports Namespaces. It is similar to a database schema that describes the data in a database.

Syntax

You need to declare a schema in your XML document as follows:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

Example

The following example shows how to use schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="contact">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="name" type="xs:string" />
         <xs:element name="company" type="xs:string" />
         <xs:element name="phone" type="xs:int" />
      </xs:sequence>
   </xs:complexType>
</xs:element>
</xs:schema>
```


The basic idea behind XML Schemas is that they describe the legitimate format that an XML document can take.

5.2.7 Need and use of Schema

One of the greatest strength of XML Schemas is the support for data types.

- It is easier to describe allowable document content
- It is easier to validate the correctness of data
- It is easier to define data facets (restrictions on data)
- It is easier to define data patterns (data formats)
- It is easier to convert data between different data types
- XML Schemas Secure Data Communication

When sending data from a sender to a receiver, it is essential that both parts have the same "expectations" about the content.

With XML Schemas, the sender can describe the data in a way that the receiver will understand.

A date like: "03-11-2004" will, in some countries, be interpreted as 3.November and in other countries as 11.March.

However, an XML element with a data type like this:

<date type="date">2004-03-11</date>

ensures a mutual understanding of the content, because the XML data type "date" requires the format "YYYY-MM-DD".

5.2.8 Building blocks

The purpose of an XML Schema is to define the legal building blocks of an XML document:

- the elements and attributes that can appear in a document
- the number of (and order of) child elements
- data types for elements and attributes
- default and fixed values for elements and attributes

5.2.9 Simple elements

Elements

As we saw in the XML - Elements chapter, elements are the building blocks of XML document. An element can be defined within an XSD as follows:

```
<xs:element name="x" type="y"/>
```

Definition Types

You can define XML schema elements in following ways:

Simple Type - Simple type element is used only in the context of the text. Some of predefined simple types are: xs:integer, xs:boolean, xs:string, xs:date. For example:

```
<xs:element name="phone_number" type="xs:int" />
```

Complex Type - A complex type is a container for other element definitions. This allows you to specify which child elements an element can contain and to provide some structure within your XML documents. For example:

```
<xs:element name="Address">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="name" type="xs:string" />
```

```
            <xs:element name="company" type="xs:string" />
        <xs:element name="phone" type="xs:int" />
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

In the above example, *Address* element consists of child elements. This is a container for other <xs:element> definitions, that allows to build a simple hierarchy of elements in the XML document.

Global Types - With global type, you can define a single type in your document, which can be used by all other references. For example, suppose you want to generalize the *person* and *company* for different addresses of the company. In such case, you can define a general type as below:

```
<xs:element name="AddressType">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="name" type="xs:string" />
            <xs:element name="company" type="xs:string" />
      </xs:sequence>
   </xs:complexType>
</xs:element>
```


Now let us use this type in our example as below:

```
<xs:element name="Address1">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="address" type="AddressType" />
            <xs:element name="phone1" type="xs:int" />
      </xs:sequence>
   </xs:complexType>
</xs:element>
<xs:element name="Address2">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="address" type="AddressType" />
            <xs:element name="phone2" type="xs:int" />
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Instead of having to define the name and the company twice (once for *Address1* and once for *Address2*), we now have a single definition. This makes maintenance simpler, i.e.,

if you decide to add "Postcode" elements to the address, you need to add them at just one place.

5.2.10 Defining attributes

Attributes in XSD provide extra information within an element. Attributes have *name* and *type* property as shown below:

```
<xs:attribute name="x" type="y"/>
```

5.2.11 Complex elements

A complex element is an XML element that contains other elements and/or attributes.

There are four kinds of complex elements:

- empty elements
- elements that contain only other elements
- elements that contain only text
- elements that contain both other elements and text

Examples of Complex Elements

A complex XML element, "product", which is empty:

```
<product pid="1345"/>
```


A complex XML element, "employee", which contains only other elements:

```
<employee>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</employee>
```

5.3 XMLwith.NET

5.3.1 XMLSerialization in the .NETFramework

Serialization is the process of converting an object into a form that can be readily transported. For example, you can serialize an object and transport it over the Internet using HTTP between a client and a server. On the other end, deserialization reconstructs the object from the stream.

XML serialization serializes only the public fields and property values of an object into an XML stream. XML serialization does not include type information. For example, if you have a **Book** object that exists in the **Library** namespace, there is no guarantee that it is deserialized into an object of the same type.

Note

XML serialization does not convert methods, indexers, private fields, or read-only properties (except read-only collections). To serialize all an object's fields and properties, both public and private, use the DataContractSerializer instead of XML serialization.

Advantages of Using XML Serialization

The **XmlSerializer**class gives you complete and flexible control when you serialize an object as XML. If you are creating an XML Web service, you can apply attributes that control serialization to classes and members to ensure that the XML output conforms to a specific schema.

For example, **XmlSerializer** enables you to:

- Specify whether a field or property should be encoded as an attribute or an element.
- Specify an XML namespace to use.
- Specify the name of an element or attribute if a field or property name is inappropriate.

Another advantage of XML serialization is that you have no constraints on the applications you develop

5.3.2 SOAP Fundamentals

SOAP is an acronym for Simple Object Access Protocol. It is an XML-based messaging protocol for exchanging information among computers. SOAP is an application of the XML specification.

Below mentioned are some important point which the user should take note of. These points briefly describes the nature of SOAP −

- SOAP is a communication protocol designed to communicate via Internet.
- SOAP can extend HTTP for XML messaging.
- SOAP provides data transport for Web services.
- SOAP can exchange complete documents or call a remote procedure.
- SOAP can be used for broadcasting a message.
- SOAP is platform- and language-independent.
- SOAP is the XML way of defining what information is sent and how.
- SOAP enables client applications to easily connect to remote services and invoke remote methods.

Although SOAP can be used in a variety of messaging systems and can be delivered via a variety of transport protocols, the initial focus of SOAP is remote procedure calls transported via HTTP.

A SOAP message is an ordinary XML document containing the following elements −

- **Envelope** − Defines the start and the end of the message. It is a mandatory element.

- **Header** − Contains any optional attributes of the message used in processing the message, either at an intermediary point or at the ultimate end-point. It is an optional element.
- **Body** − Contains the XML data comprising the message being sent. It is a mandatory element.
- **Fault** − An optional Fault element that provides information about errors that occur while processing the message.

All these elements are declared in the default namespace for the SOAP envelope

The following block depicts the general structure of a SOAP message –

```
<?xml version="1.0"?>

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2001/12/soap-envelope" SOAP-
ENV:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <SOAP-ENV:Header>

    ...

    ...

  </SOAP-ENV:Header>



  <SOAP-ENV:Body>

    ...

    ...

    <SOAP-ENV:Fault>

      ...

      ...

    </SOAP-ENV:Fault>

    ...

  </SOAP-ENV:Body>

  </SOAP_ENV:Envelope>
```

5.3.3 Using SOAP with the .NET Framework

The advantage of using the SOAP interface over the XML interface from statically-typed languages like C# and VB.NET is that there is less XML processing to do, since the web services infrastructure in .NET handles deserializing the SOAP XML responses from the SOAP interface into strongly-typed .NET collections.

To use the SOAP interface from a .NET project, you can use the Add Web Reference functionality in Visual Studio

How to use Soap ?

The Bing SOAP interface is most efficiently accessed by referencing the ***Web Service Description Language*** (WSDL) document from a Microsoft Visual Studio project. The WSDL defines the ports and messages that comprise the Bing API SOAP web service.

To add a Web reference in Microsoft Visual Studio

1. From **Solution Explorer** in an existing or newly created project, right-click **References** and, from the pop-up menu, select **Add Service Reference**.

   If you are using Microsoft Visual Studio 2005, this pop-up menu includes **Add Web Reference**. In this case, click **Add Web Reference** and proceed to Step 3.

   If you are using Microsoft Visual Studio 2008, proceed to Step 2.

2. Click **Advanced** on the **Add Service Reference** dialog box, then click **Add Web Reference** on the **Service Reference Settings** dialog box.

3. Type the following address in the URL text box: **http://api.bing.net/search.wsdl?AppID=***YourAppId***&Version=2.2**. For information about obtaining an AppId, see Bing Developer Center.

4. Click **Go.**

5. You can accept the default web reference name net.bing.api suggested in the **Web reference name** text box, or type your own name for the web reference in the text box. Click **Add Reference** to add the web reference to your project.

Review Questions
UNIT - V

Part A (2 marks)

1. What is XML?
2. What is an XML parser?
3. Write the syntax of XML element.
4. What is a DTD?
5. What is an XML namespce ?
6. What is default namespace?
7. What is serialization?
8. What are the importance of SOAP?
9.

Part B (3 marks)

1. Is XML a programming language?
2. Give an example for well formed XML document.
3. List the rules of XML element.
4. Mention the three types of character entities and give a brief note on each.
5. Describe the method of commenting in XML.
6. List the rules of XML comments.
7. Differentiate between internal DTD and external DTD.
8. List the need and use of schema.
9. Write notes on: default namespace
10. Mention the advantages of using XML serialization.
11. List the properties of SOAP
12. Explain the format of SOAP message.

Part C (5 marks)

1. Mention and describe the advantages of XML.
2. Compare HTML and XML in detail.
3. List and explain the components of well formed XML document
4. List and explain the entities of XML.
5. Discuss the three types of attributes in detail.
6. With an example explain the method of linking XML and its DTD.
7. Describe the procedure of using SOAP with .NET framework.

******