



# KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed University Established Under Section 3 of UGC Act 1956)

Coimbatore - 641021.

(For the candidates admitted from 2016 onwards)

## DEPARTMENT OF CS, CA & IT

---

**SUBJECT : .NET PROGRAMMING**

**SEMESTER : V**

**L T P C**

**SUBJECT CODE: 16CTU502A**

**CLASS : III B. Sc. CT**

**4 0 0 4**

---

**Instruction Hours / week: L: 4 T: 0 P: 0    Marks: Int : 40 Ext : 60    Total: 100**

### SCOPE

Understanding the platform. Determinism and concurrency; Handling input and output securely; safe error handling and logging; Engineering for security features; software security in operations.

### OBJECTIVES

- Grasp the fundamentals of a programming language and know the basic differences between programming languages
- .NET using WPF is a latest technology followed in the IT field.
- Choose the architecture based on the problem to be solved.
- Differentiate between the types of applications supported by .Net
- Build, compile and execute a VB .Net program
- Apply techniques to develop error-free software

### UNIT I

Introduction to .NET: .NET framework features & architecture, CLR, common Type system, MSIL, Assemblies and class libraries. Introduction to visual studio, Project basics, types of project in .Net, IDE of VB .Net – Menu bar, Tool bar, Solution Explorer, Toolbox, Properties Window, Form Designer, Output Window, Object browser. The environment: Editor tab, format tab, general tab, docking tab. Visual development & event driven programming – Methods and events.

### UNIT II

The VB .Net Language: The VB .Net Language – Variables- declaring variables, Data type of variables, forcing variables declarations, scope & lifetime of a variable, constants, arrays, types of arrays, control array, Structure programming – Modularity – Information hiding – abstraction – events – subroutines and functions – message box – input box. Control flow statement: conditional statement, loop statement.

### **UNIT III**

Working with WPF: Introduction: Understanding Windows Graphics – WPF: A Higher Level API – The architecture of WPF. XAML: Basics, properties and events in XAML – loading and compiling – Layout. Classic controls: The Control class – content controls – text controls – list controls – Range based controls.

### **UNIT IV**

Objects and Collections: Understanding objects, properties, methods. Understanding collections. Files: Introduction – classification of files – processing files – handling files and folder using class – directory class – file class.

### **UNIT V**

Database programming with ADO .Net: overview of ADO, from ADO to ADO .Net, accessing data using server explorer. Creating connection, command, data adapter and data set with OLEDB and SQLDB. Display data on data bound controls, display data on a data grid. Generate reports using CrystalReportViewer.

### **Suggested Readings**

1. Shrishchavan (2007). Visual Basic .Net (1st ed.). New Delhi: Pearson education.
2. Bryan Newsome (2012). Beginning Visual Basic. John Wiley & Sons, Inc.
3. Matthew MacDonald Pro (2008). Windows Presentation Foundation with .Net 3.5 Apress
4. Duncan Mackenzie and Kent Sharkey (2006). Sams Teach Yourself Visual Basic .Net (1st ed.). New Delhi: Techmedia.
5. Ian Griffiths, Chris Shells (2005). Programming Windows Presentation Foundation (1st ed.). O'Reilly Publishers
6. Jeffrey R. Shapiro (2002). The Complete Reference Visual Basic .Net. New Delhi: Tata McGraw Hill Ed.

### **Websites**

1. [www.startvbdotnet.com](http://www.startvbdotnet.com)
2. [www.functionx.com](http://www.functionx.com)
3. [www.dotnetspider.com](http://www.dotnetspider.com)
4. [www.developerfusion.com](http://www.developerfusion.com)
5. [http://www.wdftutorial.net/HelloWPF.html](http://www.wdftutorial.net>HelloWPF.html)



## KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed University Established Under Section 3 of UGC Act 1956)

Coimbatore - 641021.

(For the candidates admitted from 2016 onwards)

### DEPARTMENT OF CS, CA & IT

**SUBJECT : . NET PROGRAMMING**

**SEMESTER : V**

**L T P C**

**SUBJECT CODE: 16CTU502A**

**CLASS : III B. Sc. CT**

**4 0 0 4**

### LECTURE PLAN

**STAFF NAME: Mr. K. VEERASAMY**

S. No	Lecture Duration (Hr)	Topics	Support Materials
<b>UNIT-I</b>			
1.	1	Introduction to .NET: .NET framework features & architecture	S1: 1-4, S1:27-30
2.	1	CLR, common Type system, MSIL, Assemblies and class libraries.	S1:31-37, W1
3.	1	Introduction to visual studio, Project basics	S1:6-10
		Types of project in .Net	W1
4.	1	IDE of VB .Net, Menu bar	S1:19-22
		Tool bar, Solution Explorer	S1:10-23, W1
5.	1	Toolbox, Properties Window	S1:14-18
6.	1	Form Designer, Output Window, and Object browser.	S1:36-51
7.	1	The environment: Editor tab, format tab, general tab, docking tab	S1:24-31
8.	1	Visual development & event driven programming	S1:19-30
		Methods and events.	S1:76-83, W2 S2:187-189,
9.	1	Recapitulation and discussion of important questions	
<b>Total No of Hours Planned For Unit – I</b>			<b>9</b>

UNIT-II			
1.	1	The VB .Net Language: The VB .Net Language	S1:99-101
		Variables - declaring variables	S111-116
2.	1	Data type of variables, forcing variables declarations, scope & lifetime of a variable	S1:104-105 S1:44-48 W1 S3:86-90
3.	1	Constants,	S1:130-131
		Arrays, types of arrays, control array	W2
4.	1	Structure programming – Modularity	S1:204 T:77
5.	1	Information hiding – abstraction	S1:292-298
6.	1	Events – subroutines and functions	S1:204 T2:77
7.	1	Message box – input box	S1:142-146, 156-158
8.	1	Control flow statement: conditional statement.	S1:125-129
		Loop statement.	S1:90-103, W4
9.	1	Recapitulation and discussion of important questions	
		<b>Total No of Hours Planned For Unit – II</b>	<b>9</b>
UNIT-III			
1.	1	Working with WPF: Introduction: Understanding	S3:1-3, W2
2.	1	Windows Graphics – WPF: A Higher Level API	S3:4-21 W2
		The architecture of WPF	
3.	1	XAML: Basics	S3:23-29, W2
4.	1	Properties and events in XAML	S3:32-36, W2
5.	1	Loading and compiling – Layout	S3:49-58, W2
6.	1	Classic controls: The Control class	S3:183-193
7.	1	Content controls, Text controls	S4:139-141
8.	1	List controls, Range based controls.	S3:215-217
9.	1	Recapitulation and discussion of important questions	
		<b>Total No of Hours Planned For Unit – III</b>	<b>9</b>
UNIT-IV			
1.	1	Objects and Collections: Understanding objects	S1:334-335
2.	1	Properties, Methods	S1:338-345
3.	1	Understanding collections	S1:335 T2:483
4.	1	Files: Introduction	S1:399-400
5.	1	Classification of files	S1:400-402
6.	1	Processing files	S1:402
7.	1	Handling files and folder using class	S1:400-408
8.	1	Directory class, File class	S1:410-415, 428-235
9.		Recapitulation and discussion of important questions	

		<b>Total No of Hours Planned For Unit – IV</b>	<b>9</b>
<b>UNIT-V</b>			
1.	1	Database programming with ADO .Net: overview of ADO	S2:293-299,320-322 S2:515-516
2.	1	from ADO to ADO .Net, Accessing data using server explorer	S2:320-325
3.	1	Creating connection, command and Data adapter	S2:320-335
4.	1	Data set with OLEDB	S2:325-330
5.	1	Data set with SQLDB	S2:325-330
6.	1	Display data on data bound controls	S2:363-373
7.	1	Display data on a data grid	S2:363-373
8.	1	Generate reports using Crystal Report Viewer.	S2:363-373
9.	1	Recapitulation and discussion of important questions	
10.	1	<b>Discussion of previous ESE Question papers</b>	
11.	1	<b>Discussion of previous ESE Question papers</b>	
12.	1	<b>Discussion of previous ESE Question papers</b>	
		<b>Total No. of Hours Planned For Unit – V</b>	<b>12</b>
		<b>Total No. of Hours Planned: 60</b>	

**Suggested Readings:**

1. Shrishchavan (2007). Visual Basic .Net (1st ed.). New Delhi: Pearson education.
2. Bryan Newsome (2012). Beginning Visual Basic. John Wiley & Sons, Inc.
3. Matthew MacDonald Pro (2008). Windows Presentation Foundation with .Net 3.5 Apress
4. Duncan Mackenzie and Kent Sharkey (2006). Sams Teach Yourself Visual Basic .Net (1<sup>st</sup> Ed.). New Delhi: Techmedia.
5. Ian Griffiths, Chris Shells (2005). Programming Windows Presentation Foundation (1<sup>st</sup> Ed.). O'Reilly Publishers
6. Jeffrey R. Shapiro (2002). The Complete Reference Visual Basic .Net. New Delhi: Tata McGraw Hill Ed.

**Websites:**

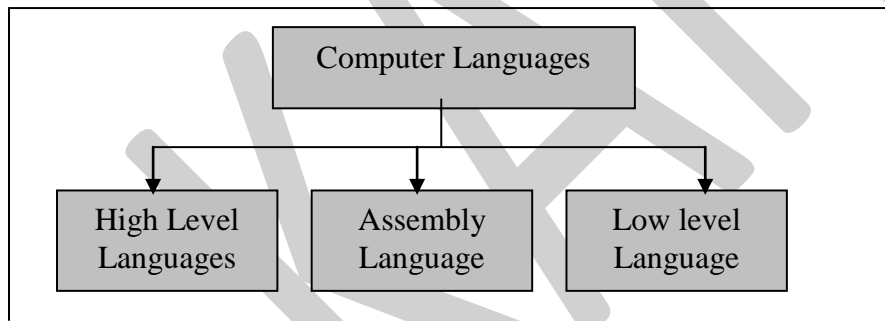
1. [www.startvbdotnet.com](http://www.startvbdotnet.com)
2. [www.functionx.com](http://www.functionx.com)
3. [www.dotnetspider.com](http://www.dotnetspider.com)
4. [www.developerfusion.com](http://www.developerfusion.com)
5. <http://www.wdftutorial.net/HelloWPF.html>

## UNIT – I

Introduction to .NET: .NET framework features & architecture, CLR, common Type system, MSIL, Assemblies and class libraries. Introduction to visual studio, Project basics, types of project in .Net, IDE of VB .Net – Menu bar, Tool bar, Solution Explorer, Toolbox, Properties Window, Form Designer, Output Window, Object browser. The environment: Editor tab, format tab, general tab, docking tab. Visual development & event driven programming – Methods and events.

### Introduction to .NET

As every living object in the world needs a language to communicate with each other. Similarly the computer needs a language to communicate with the human beings. So to communicate with the computer, we need a language as just like different languages are spoken all over the world, there are different languages to communicate with the computer. Such languages are known as Computer Languages. Compute Languages can be broadly divided into three types namely high level languages, Assembly level languages and Low level languages as shown in the figure.



**Fig Programming Languages**

High level languages are so powerful which are widely used by the humans to communicate with the computer. They are called as high level languages since they are easier to learn and use. So many computer languages come under this category. Examples of such languages are BASIC, FORTRAN, COBOL, Pascal, C, C++, ALGOL, PL/1, Ada, Visual Basic, Visual C++, and Visual Basic. Net and so on. In these languages, we use English like words as instructions to communicate with the

computer. People used a language called Assembly Language to communicate with the computers before the introduction of high level languages.

In Assembly language, the instructions would be in the form of Mnemonic Codes or Symbolic Codes. These codes are nothing but the instructions in the form of abbreviations. For example the assembly language uses the abbreviated codes like ADD, SUB or MULT and so on for to Add, Subtract or Multiply. Because of the complexity, the assembly language is not widely used when compared to that of high level languages. Computers can only understand binary values such as 1 and 0. With the help of translators, the high level language code will be converted into machine language. Before the translators were introduced, only machine language is used for the communication.

.Net framework is a platform or an environment in which the user creates applications using any one of the programming language that supported by .Net. It is a platform neutral framework. The .NET Framework is a collection of services and classes. It is a layer between the operating system and the programming language. It supports many programming languages, including VB.NET, C# etc. NET provides a common set of class libraries, which can be accessed from any .NET based programming language. There will not be separate set of classes and libraries for each language. If you know any one .NET language, you can write code in any .NET language. It is a major technology change. Just like the computer world moved from DOS to Windows, now they are moving to .NET. .NET technology was introduced by Microsoft, to catch the market from the SUN's Java. Few years back, Microsoft had only VC++ and VB to compete with Java, but Java was catching the market very fast. With the world depending more and more on the Internet/Web and java related tools becoming the best choice for the web applications. Thousands of programmers moved to java from VC++ and VB. This was alarming for Microsoft and many of the Microsoft fans.

In order to tackle this Microsoft planned a project called **Generation Windows Services (NGWS)**, under the direct supervision of Mr. Bill Gates. The outcome of the project is what we now know as .NET. Even though .NET has borrowed most of its ideas from Sun's J2EE, it has really outperformed their competitors.

Microsoft's VC++ was a powerful tool. But it was too complex. It has too many data types, and developers had to learn many libraries including Windows SDK, MFC, ATL, COM etc. There were many data type compatibility issues while exchanging data between different layers. Visual Basic was too easy, and many serious programmers hated it just for that reason. Even though Visual basic was very easy to use, it was not very flexible to develop serious applications. Sun's Java became a very good choice for

these reasons. It had the flexibility and power of C++ and at the same time easy enough to catch the attention of VB programmers.

Microsoft recognized these factors and they introduced the .NET considering all these factors. All unwanted complexities are eliminated and a pure object oriented programming model was introduced. This makes programmer's life very easy. .NET framework comes with a single class library. And that's all programmers need to learn!! Whether they write the code in C# or VB.NET or J#, it doesn't matter you just use the .NET class library. There is no classes specific to any language. There is nothing more you can do in a language, which you can't do in any other .NET language. You can write code in C# or VB.NET with the same number of lines of code, same performance and same efficiency, because everyone uses same .NET class library. We cannot define .NET as a 'single thing'. It is a new, easy, and extensive programming platform. It is not a programming language, but it supports several programming languages. By default .NET comes with few programming languages including C# (C Sharp), VB.NET, J# and managed C++. .NET is a common platform for all the supported languages. It gives a common class library, which can be called from any of the supported languages. So, developers need not learn many libraries when they switch to a different language. Only the syntax is different for each language.

### **History and Version of .NET Framework**

<b>Generation</b>	<b>Version number</b>	<b>Year</b>	<b>Development tool</b>	<b>Distributed with</b>
1.	1.0	2002	Visual Studio .NET	N/A
	1.1	2003	Visual Studio .NET 2003	Windows Server 2003
2.	2.0	2005	Visual Studio 2005	Windows Server 2003 R2
3.	3.0	2006	Expression Blend	Windows Vista, Windows Server 2008
	3.5	2007	Visual Studio 2008	Windows 7, Windows Server 2008 R2
4.	4.0	2010	Visual Studio 2010	N/A
	4.5	2012	Visual Studio 2012	Windows 8, Windows Server 2012



---

**Features of .NET Framework.**

**i) Net Framework 2.0 Features**

a).ADO.NET: New features in ADO.NET include support for user-defined types (UDT), asynchronous database operations, XML data types, large value types, snapshot isolation, and new attributes that allow applications to support multiple active result sets (MARS) with SQL Server 2005.

b).ASP.NET: The Microsoft .NET Framework 2.0 includes significant enhancements to all areas of ASP.NET. For Web page development, new controls make it easier to add commonly used functionality to dynamic Web pages. New data controls make it possible to display and edit data on an ASP.NET Web page without writing code. An improved code-behind model makes developing ASP.NET pages easier and more robust. Caching features provide several new ways to cache pages, including the ability to build cache dependency on tables in a SQL Server database.

c). NET Remoting: NET Framework Remoting now supports IPv6 addresses and the exchange of generic types. The classes in the System.Runtime.Remoting.Channels.Tcp namespace support authentication and encryption using the Security Support Provider Interface (SSPI). Classes in the new System.Runtime.Remoting.Channels.Ipc namespace allow applications on the same computer to communicate quickly without using the network.

d).XML: The new System.Xml.XmlReaderSettings class allows specification of the type of verifications to be done when using a subclass of XmlReader to read XML data.

**ii) .Net Framework 3.0/3.5 Features**

a).Windows Presentation Foundation (WPF): Windows Presentation Foundation (WPF) is a next-generation presentation system for building Windows client applications. The core of WPF is a resolution-independent and vector-based rendering engine that is built to take advantage of modern graphics hardware. WPF extends the core with a comprehensive set of application-development features that include Extensible Application Markup Language (XAML), controls, data binding, layout, 2-D and 3-Dgraphics, animation, styles, templates, documents, media, text, and typography. WPF is included in the Microsoft .NET Framework, so you can build applications that incorporate other elements of the .NET Framework class library.

b).Windows Communication Foundation (WCF): Windows Communication Foundation (WCF) is Microsoft's unified programming model for building service-oriented applications. WCF allows you to build many kinds of distributed applications including "traditional" Web Services so that your services support SOAP and will therefore be compatible with older .NET (and other) technologies.

c).Windows Workflow Foundation (WWF): Windows Workflow Foundation, a core component of .NET Framework 3.0, provides a programming model, run-time engine, and tools for building workflow applications. Workflows can be authored in code, XAML markup, or a combination of both, known as code-separation, which is similar to the ASP.NET mode.

d).Windows Card Space (WCS): Windows CardSpace (InfoCard) is a Digital Identity to online services. Digital Identity is how a user will be electronically represented. Such as for a debit/credit card, each card has a digital identity and password. If any user uses the site on internet then he enters their username and password, for identity, but this is not secure. WCS reduces these types of problems.

### iii) .Net Framework 4.0 Features

**a).Application Compatibility and Deployment:** The .NET Framework 4 is highly compatible with applications that are built with earlier .NET Framework versions, except for some changes that were made to improve security, standards compliance, correctness, reliability, and performance. The .NET Framework 4 does not automatically use its version of the common language runtime to run applications that are built with earlier versions of the .NET Framework. To run older applications with .NET Framework 4, you must compile your application with the target .NET Framework version specified in the properties for your project in Visual Studio, or you can specify the supported runtime with the <supportedRuntime> Element in an application configuration file.

**b).Managed Extensibility Framework:** The Managed Extensibility Framework (MEF) is a new library in the .NET Framework 4 that helps you build extensible and composable applications. MEF enables you to specify points where an application can be extended, to expose services to offer to other extensible applications and to create parts for consumption by extensible applications. It also enables easy discoverability of available parts based on metadata, without the need to load the assemblies for the parts.

**c).Parallel Computing:** The .NET Framework 4 introduces a new programming model for writing multithreaded and asynchronous code that greatly simplifies the work of application and library developers. The new model enables developers to write efficient, fine-grained, and scalable parallel code in a natural idiom without having to work directly with threads or the thread pool. The new System.Threading.Tasks namespace and other related types support this new model.

**d).Web :** Web Forms, including more integrated support for ASP.NET routing, enhanced support for Web standards, updated browser support, new features for data controls, and new features for view state management. Web Forms controls, including a new Chart control. Dynamic Data, including support for existing Web applications, support for many-to-many relationships and inheritance, new field templates and attributes, and enhanced data filtering. Microsoft Ajax, including additional support for client-based Ajax applications in the Microsoft Ajax Library. Visual Web Developer, including improved IntelliSense for JScript, new auto-complete snippets for HTML and ASP.NET markup, and enhanced CSS compatibility. Deployment, including new tools for automating typical deployment tasks. Multi-targeting, including better filtering for features that are not available in the target version of the .NET Framework

**e) Windows Presentation Foundation (WPF) Features in 4.0:** Windows Presentation Foundation (WPF) version 4 contains changes and improvements in such as New controls, including Calendar, Data Grid, and Date Picker. VisualStateManager supports changing states of controls. Touch and Manipulation enables you to create applications that receive input from multiple touches simultaneously on Windows 7. Graphics and animation supports layout rounding, Pixel Shader version 3.0, cached composition, and easing functions. Text has improved text rendering and supports customizing the caret color and selection color in text boxes. Binding is supported on the Command property of an InputBinding, dynamic objects, and the Text property. XAML browser applications (XBAPs) support communication with the Web page and support full-trust deployment. New types in the System.Windows.Shell namespace enable you to communicate with the Windows 7 taskbar and pass data to the Windows shell. The WPF and Silverlight Designer in Visual Studio 2010 has various designer improvements to help create WPF or Silverlight applications.

**f).Windows Workflow Foundation Features in 4.0:**Windows Workflow Foundation (WF) provides improvements such as :

- Improved workflow activity model: The Activity class provides the base abstraction of workflow behavior.
- Rich composite activity options: Workflows benefit from new flow-control activities that model traditional flow-control structures, such as Flowchart, TryCatch, and Switch<T>.
- Expanded built-in activity library: New features of the activity library include new flow-control activities, activities for manipulating member data, and activities for controlling transactions.

#### iv).Net Framework 4.5 Features

**a).NET for Windows Store Apps:** Windows Store apps are designed for specific form factors and leverage the power of the Windows operating system. A subset of the .NET Framework 4.5 is available for building Windows Store apps for Windows by using C# or Visual Basic.

**b)Portable Class Libraries:** The Portable Class Library project in Visual Studio 2012 enables you to write and build managed assemblies that work on multiple .NET Framework platforms. Using a Portable Class Library project, you choose the platforms (such as Windows Phone and .NET for Windows Store apps) to target.

#### c) ASP.NET: ASP.NET 4.5 includes the following new features:

- Support for new HTML5 form types.
- Support for model binders in Web Forms. These let you bind data controls directly to data-access methods, and automatically convert user input to and from .NET Framework data types.
- Support for unobtrusive JavaScript in client-side validation scripts.
- Improved handling of client script through bundling and minification for improved page performance.
- Integrated encoding routines from the AntiXSS library (previously an external library) to protect from cross-site scripting attacks.
- Support for WebSockets protocol.

d) Windows Presentation Foundation (WPF) Features in 4.5: In the .NET Framework 4.5, Windows Presentation Foundation (WPF) contains changes and improvements in the following areas:

- The new Ribbon control, which enables you to implement a ribbon user interface that hosts a Quick Access Toolbar, Application Menu, and tabs.
- The new INotifyDataErrorInfo interface, which supports synchronous and asynchronous data validation.
- New features for the VirtualizingPanel and Dispatcher classes.
- Improved performance when displaying large sets of grouped data, and by accessing collections on non-UI threads.
- Data binding to static properties, data binding to custom types that implement the ICustomTypeProvider interface, and retrieval of data binding information from a binding expression.
- Repositioning of data as the values change (live shaping).
- Ability to check whether the data context for an item container is disconnected.
- Ability to set the amount of time that should elapse between property changes and data source updates.
- Improved support for implementing weak event patterns. Also, events can now accept markup extensions.

e) **Windows Communication Foundation (WCF) Features in 4.5:** In the .NET Framework 4.5, the following features have been added to make it simpler to write and maintain Windows Communication Foundation (WCF) applications:

- Simplification of generated configuration files.
- Ability to configure ASP.NET compatibility mode more easily.
- Validation of WCF configuration files by Visual Studio as part of the build process, so you can detect configuration errors before you run your application.

- New asynchronous streaming support.
- New HTTPS protocol mapping to make it easier to expose an endpoint over HTTPS with Internet Information Services (IIS).
- Web sockets support to enable true bidirectional communication over ports 80 and 443 with performance characteristics similar to the TCP transport.
- XML Editor tooltips.

**f) Windows Workflow Foundation (WF) Features in 4.5:** Several new features have been added to Windows Workflow Foundation (WF) in the .NET Framework 4.5. These new features include:

- State machine workflows, which were first introduced as part of the .NET Framework 4.0.1 (.NET Framework 4 Platform Update 1). This update included several new classes and activities that enabled developers to create state machine workflows. These classes and activities were updated for the .NET Framework 4.5 to include:
- The ability to set breakpoints on states.
- The ability to copy and paste transitions in the workflow designer.
- Designer support for shared trigger transition creation.

### **.NET Framework Architecture**

.NET is a "Software Platform". It is a language-neutral environment for developing rich .NET experiences and building applications that can easily and securely operate within it. When developed applications are deployed, those applications will target .NET and will execute wherever .NET is implemented instead of targeting a particular Hardware/OS combination. The components that make up the .NET platform are collectively called the .NET Framework. The .NET Framework is a managed, type-safe environment for developing and executing applications. The .NET Framework manages all aspects of program execution, like, allocation of memory for the storage of data and instructions, granting and denying permissions to the application, managing execution of the application and reallocation of memory for resources that are not needed.



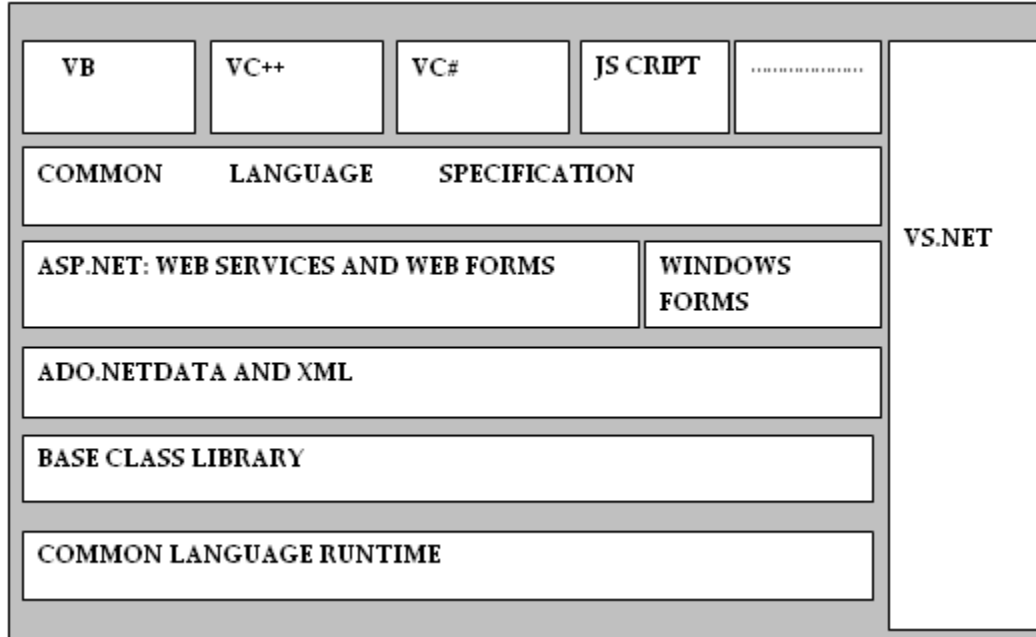


Fig Net Framework architecture

The .NET Framework is designed for cross-language compatibility. Cross-language compatibility means, an application written in Visual Basic .NET may reference a DLL file written in C# (C-Sharp). A Visual Basic .NET class might be derived from a C# class or vice versa. The .NET Framework consists of two main components:

- ▶ Common Language Runtime (CLR)
- ▶ Class Libraries

#### i) Common Language Runtime (CLR)

The CLR is also known as the "execution engine" of .NET. It provides the environment within which the programs run. The main responsibility of the CLR is that manages the execution of programs and provides core services, such as code compilation, memory allocation, thread management, and garbage collection. Through the Common Type System (CTS), it enforces strict type safety, and it ensures that the code is executed in a safe environment by enforcing code access security. The software version of .NET is actually the CLR version. When the .NET program is compiled, the output of the compiler is not an executable file but a file that contains a special type of code called the Microsoft Intermediate Language (MSIL), which is a low-level set of instructions understood by the common language run time. This MSIL defines a set of

portable instructions that are independent of any specific CPU. It's the job of the CLR to translate this Intermediate code into an executable code when the program is executed making the program to run in any environment for which the CLR is implemented. And that's how the .NET Framework achieves Portability. This MSIL is turned into executable code using a JIT (Just in Time) compiler. The process goes like this, when .NET programs are executed, the CLR activates the JIT compiler. The JIT compiler converts MSIL into native code on a demand basis as each part of the program is needed. Thus the program executes as a native code even though it is compiled into MSIL making the program to run as fast as it would if it is compiled to native code but achieves the portability benefits of MSIL.

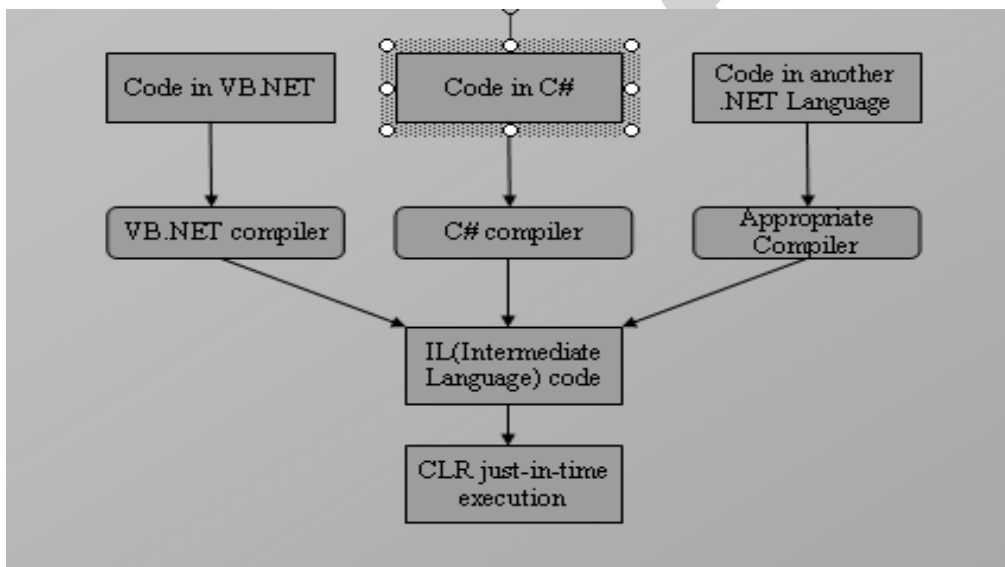


Fig Common Language Runtime

## ii) .Net Framework Class Libraries

Class library is another major entity of the .NET Framework which is designed to integrate with the common language runtime. This library gives the program access to runtime environment. The class library consists of lots of prewritten code that all the applications created in VB .NET and Visual Studio .NET will use. The code for all the elements like forms, controls and the rest in VB .NET applications actually comes from the class library.

## Name Space



.Net provides several ways to organize the Visual basic code. One among them is Namespace. A namespace is a collection of different classes. All VB applications are developed using classes from the .NET System namespace. The namespace with all the built-in VB functionality is the System namespace. All other namespaces are based on this System namespace. Some of the namespace in vb.net includes:

1. System: Includes essential classes and base classes for commonly used data types, events, exceptions and so on
2. System. Collections: Includes classes and interfaces that define various collection of objects such as list, queues, hash tables, arrays, etc
3. System. Data: Includes classes which lets us handle data from data sources
4. System.Data.OleDb: Includes classes that support the OLEDB .NET provider
5. System.Data.SqlClient: Includes classes that support the SQL Server .NET provider
6. System. Diagnostics: Includes classes that allow to debug our application and to step through our code
7. System.Drawing: Provides access to drawing methods
8. System. Reflection: Includes classes and interfaces that return information about types, methods and fields
9. System. Security: Includes classes to support the structure of common language runtime security system
10. System. Threading: Includes classes and interfaces to support multithreaded applications
11. System. Web: Includes classes and interfaces that support browser-server communication
12. System.Web.Services: Includes classes that let us build and use Web Services
13. System.Windows.Forms: Includes classes for creating Windows based forms
14. System.XML: Includes classes for XML support
15. System. Globalization: Includes classes that specify culture-related information
16. Systemic: Includes classes for data access with Files
17. System.Net: Provides interface to protocols used on the internet

Namespace statement can be used to declare a namespace in your own code. Namespace statements can be nested. For example, a Visual Basic .NET module could contain these lines of code:

Prepared by I	<pre>Namespace [Namespace Name1]   Namespace [Namespace Name2]     Public Class [Control Class]       'Statements to implement control class     End Class   End Namespace End Namespace</pre>	12/33

### Fig Namespace

By default, a Visual Basic .NET project declares a root namespace that has the same name as the project. If the programmer needs another name space then import statement can be used to import the name space. For example Imports system.Data. The usage of name space will be discussed in the forthcoming chapters to have better idea about that.

### Assemblies

An assembly is another way of organizing code along with namespace. It is a collection of types and resources that forms a logical unit of functionality. An assembly is the building block of a .NET application. It is a self describing collection of code, resources, and metadata (data about data, example, name, size, version of a file is metadata about that file). An Assembly is a compiled and versioned collection of code and metadata that forms an atomic functional unit. Assemblies take the form of a dynamic link library (.dll) file or executable program file (.exe) but they differ as they contain the information found in a type library and the information about everything else needed to use an application or component. All .NET programs are constructed from these Assemblies.

Assemblies are made of two parts: manifest, contains information about what is contained within the assembly and modules, internal files of IL code which are ready to run. When programming, we don't directly deal with assemblies as the CLR and the .NET framework takes care of that behind the scenes. The assembly file is visible in the Solution Explorer window of the project

An assembly includes:

- Information for each public class or type used in the assembly – information includes class or type names, the classes from which an individual class is derived, etc

- Information on all public methods in each class, like, the method name and return values (if any)
- Information on every public parameter for each method like the parameter's name and type
- Information on public enumerations including names and values
- Information on the assembly version (each assembly has a specific version number)
- Intermediate language code to execute
- A list of types exposed by the assembly and list of other assemblies required by the assembly

### **Common Language Specifications (CLS)**

If we want the code which we write in a language to be used by programs in other languages then it should adhere to the Common Language Specification (CLS). The CLS describes a set of features that different languages have in common. The CLS defines the minimum standards that .NET language compilers must conform to, and ensures that any source code compiled by a .NET compiler can interoperate with the .NET Framework. Some reasons why developers are building applications using the .NET Framework:

- Improved Reliability
- Increased Performance
- Developer Productivity
- Powerful Security
- Integration with existing Systems
- Ease of Deployment
- Mobility Support
- XML Web service Support
- Support for over 20 Programming Languages
- Flexible Data Access

### **Introduction to Visual Studio – Project Basics and Types**

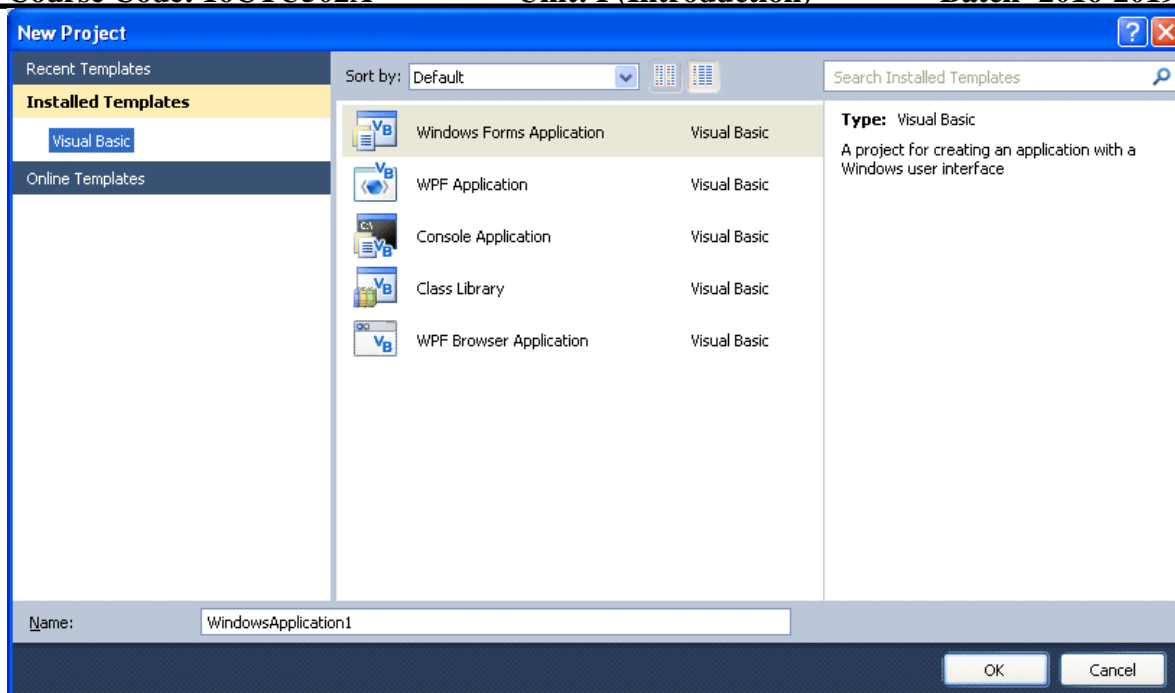
Visual studio is an editor that is used by programmer to build their application in an efficient manner. There is always confusion between Visual Studio .NET (VS.NET) and .NET technology. VS.NET is just an editor, provided by Microsoft to help developers write .NET programs easily. VS.NET editor automatically generates lot of code, allow developers to drag and drop controls to a form, provide short cuts to compile and build the application etc. Visual Studio is a very user friendly tool. But there is enough stuff to confuse any one new to Visual Studio family

### **Projects**

VS.NET allows you to create several types of projects. Most of the time you will be using one of two categories:

1. Windows Application - to create any standard windows application.
2. ASP.NET Web Application - to create a web site.

To create a new project, choose the main menu: File > New > Project. It will give you several options. First you must select a type from the left side of the popup - you may choose Visual Basic Projects or Visual C# projects based on the language you plan to use for development. After selecting a type, you choose a template from the right side. You may choose Windows Application, ASP.NET Web Application or any other template based on the nature of the application you want. For building a project, create a new Windows Project as explained above. It will create a sample form. Go to the main menu and select the menu item Build > Build Solution. This process will compile all the files included in your project and show you the result in the Output window. If the result shows '0 failed', your build is success and your application is ready to deliver!! To Run the application you just Built, go to the main menu and select Debug > Start without Debugging. This will launch the application you just developed You can drag and drop several controls to the form and try running it. When you compile (build) the code, if there is any errors or warnings, the details will be shown in the 'Task List' window. You can click on the specific item in this window to go directly to the line of code associated with the error.



**Fig. Projects in Visual Studio**

## **Integrated Development Environment (IDE)**

The Integrated Development Environment (IDE) is the development environment for all .Net based applications. To say in other words, IDE is shared by all programming languages in Visual Studio. You can view the toolbars towards the left side of the image along with the Solution Explorer window towards the right. This section Details the components of IDE with the screen shots and their usage in developing the applications.

### **1 Startup Page**

Many of the "global" resources in VS.NET are on the VS.NET Start Page. Many links are provided from the start up page. They are organized into three categories such as projects, online resources and profiles. Under the project tab, it is possible to open the existing projects/application or to create new project/applications. Using the online resource option, it is possible to link to Microsoft help and reference page. With the profile option it is possible to change keyboard schemes, windows layout, help filter and show help settings. Following figure shows the screenshot of startup page.

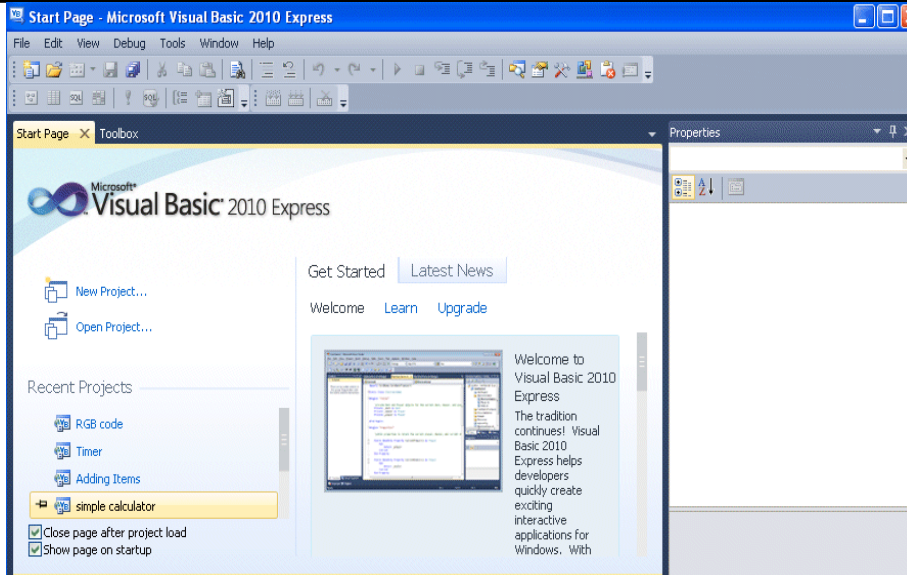


Fig Startup page

## 2 Windows Form Designer

System.Windows.Forms is the foundation class for all forms to be created. All the forms that are created in VB .NET are also inheriting from this base class. This class provides for all the facilities needed for the form. Additional functionality can be added by separate codes.

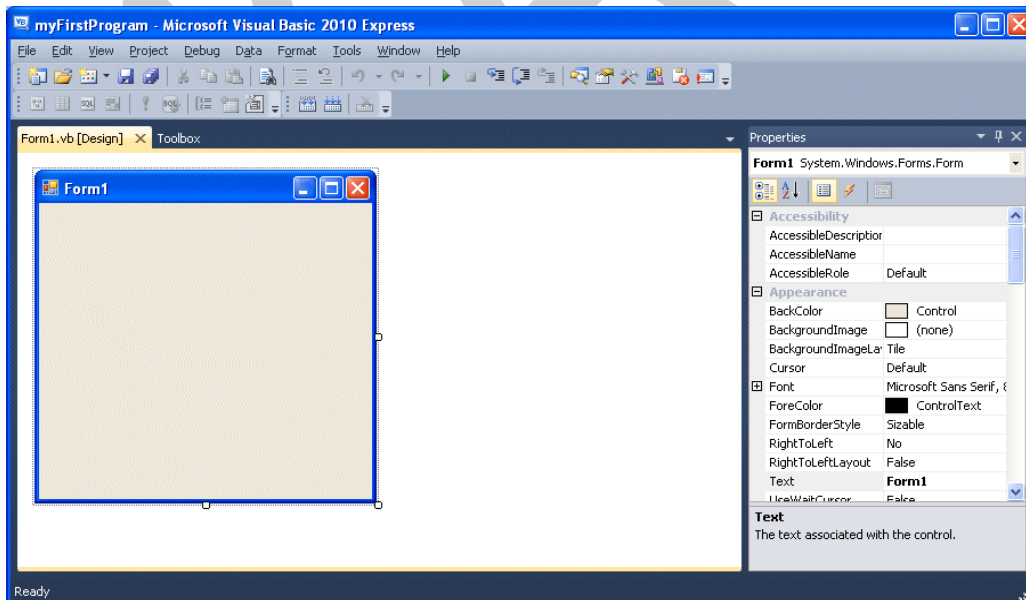


Fig Windows Form Designer



### 3 Solution Explorer Window

The Solution Explorer window gives an overview of the solution we are working with and lists all the files in the project. An image of the Solution Explorer window is shown below. It shows the project named as WindowsApplication34, an assembly file which contains assembly information of the application that is discussed in the previous section and the Form which has the default name Form1.vb. If the programmer adds a new resource for instance, another Form then the form will be visible in solution explorer window.

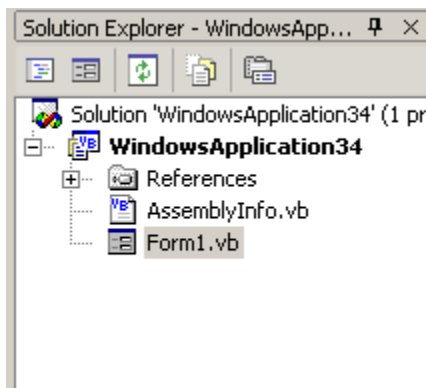


Fig Solution Explorer Windows

### 4 Properties Window

The properties window allows us to set properties for various objects at design time. For example, if you want to change the font, font size, back color, name, text that appears on a button, textbox etc, you can do that in this window. Below is the image of properties window. You can view the properties window by selecting View->Properties Window from the main menu or by pressing F4 on the keyboard.

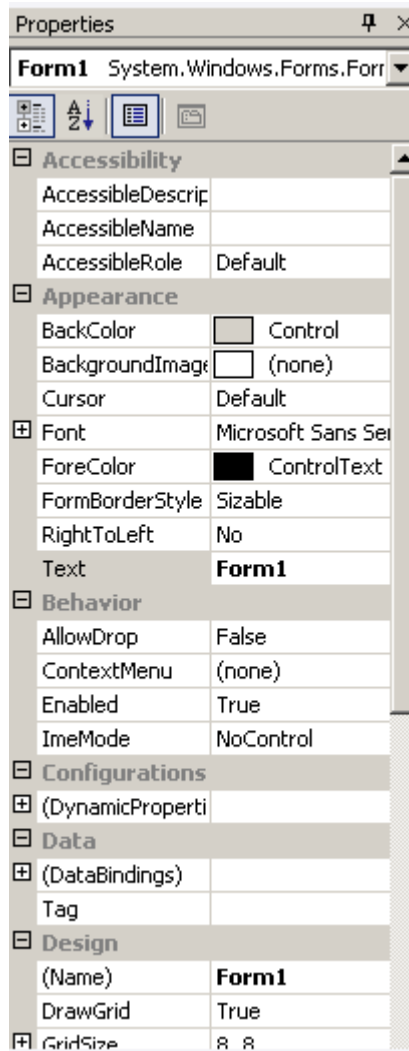


Fig Property Windows

## 5 Toolbox Window

The toolbox window is the window that gives us access to all controls, components, etc. As you can see from the image below, the toolbox uses tabs to divide its contents into categories (Data, Components, Windows Forms and General). The Data tab displays tools for creating datasets and making data connections, the Windows Forms tab displays tools for adding controls to forms, the General tab is left empty by default, the Clipboard Ring tab displays recent items stored in the clipboard and allows us to select from them.



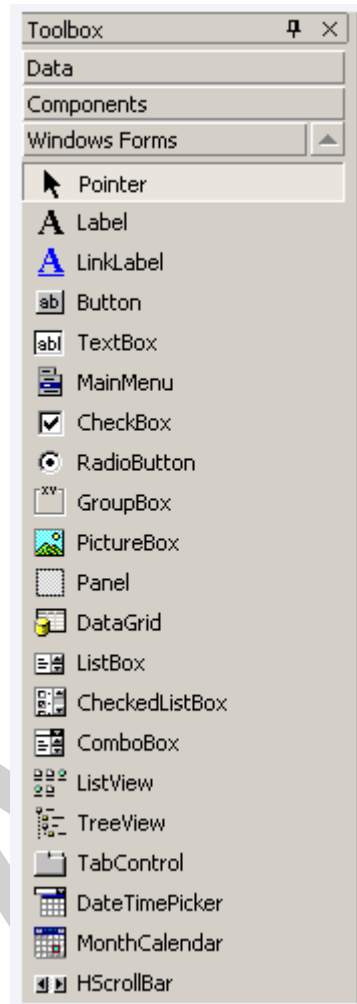


Fig Toolbox

## 6 Intellisense

Intellisense is what that is responsible for the boxes that open as we type the code. IntelliSense provides a list of options that make language references easily accessible and helps us to find the information we need. They also complete the typing for us. The image below displays that.

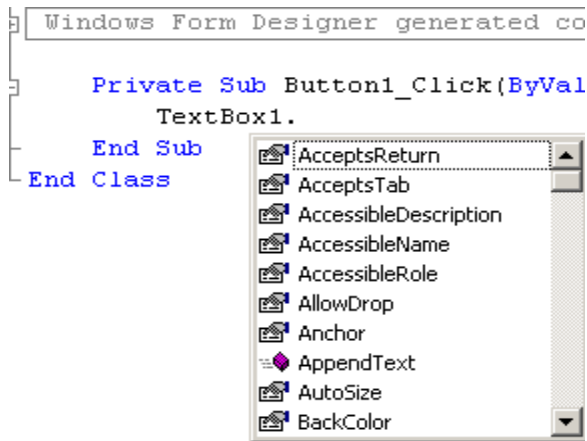


Fig Intellisense

## 7 Server Explorer Window

The Server Explorer window is a tool that provides "drag and drop" feature and helps us work with databases in an easy graphical environment. For example, if we drag and drop a database table onto a form, VB .NET automatically creates connection and command objects that are needed to access that table. The image below displays Server Explorer window.

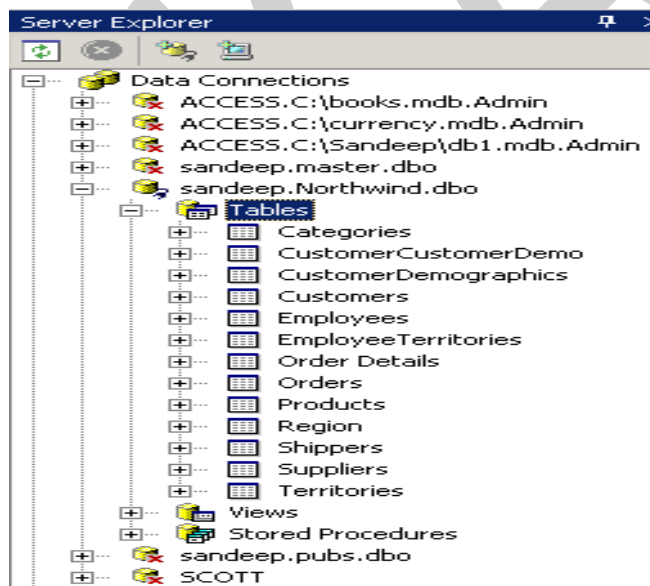
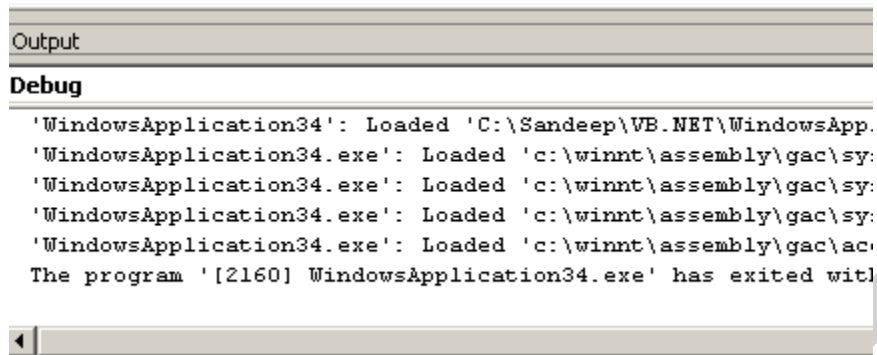


Fig Server Explorer Window

## 8 Output Window

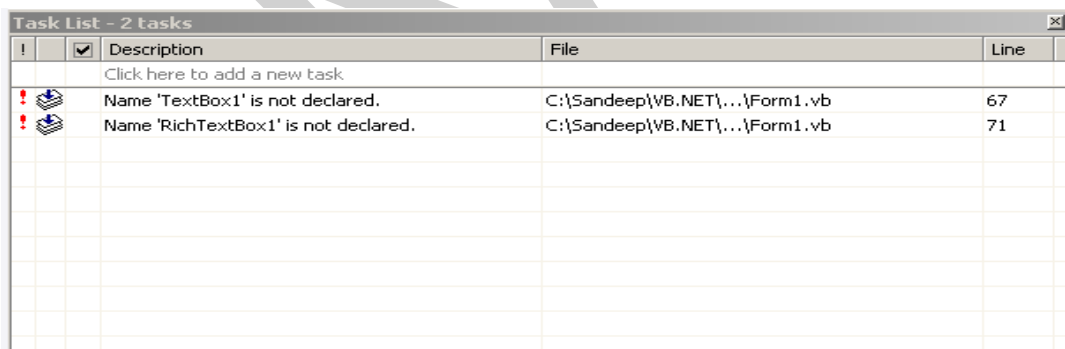
The output window as you can see in the image below displays the results of building and running applications.



**Fig Windows Form Designer**

## 9 Task List Window

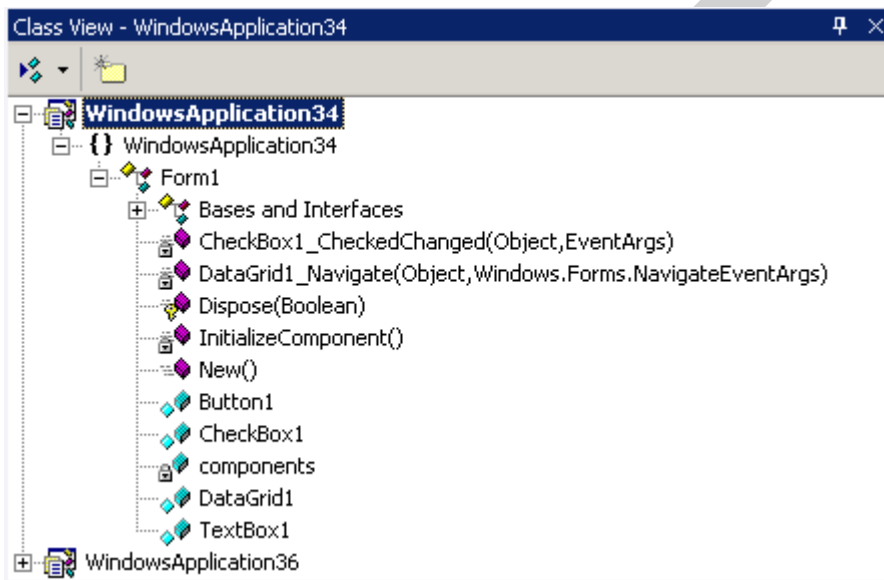
The task list window displays all the tasks that VB .NET assumes we still have to finish. You can view the task list window by selecting View->Show tasks->All or View->Other Windows->Task List from the main menu. The image below shows that. As you can see from the image, the task list displayed "TextBox1 not declared", "RichTextBox1 not declared". The reason for that message is, there were no controls on the form and attempts were made to write code for a textbox and a rich textbox. Task list also displays syntax errors and other errors you normally encounter during coding



**Fig Task List Window**

### 10 Class View Window

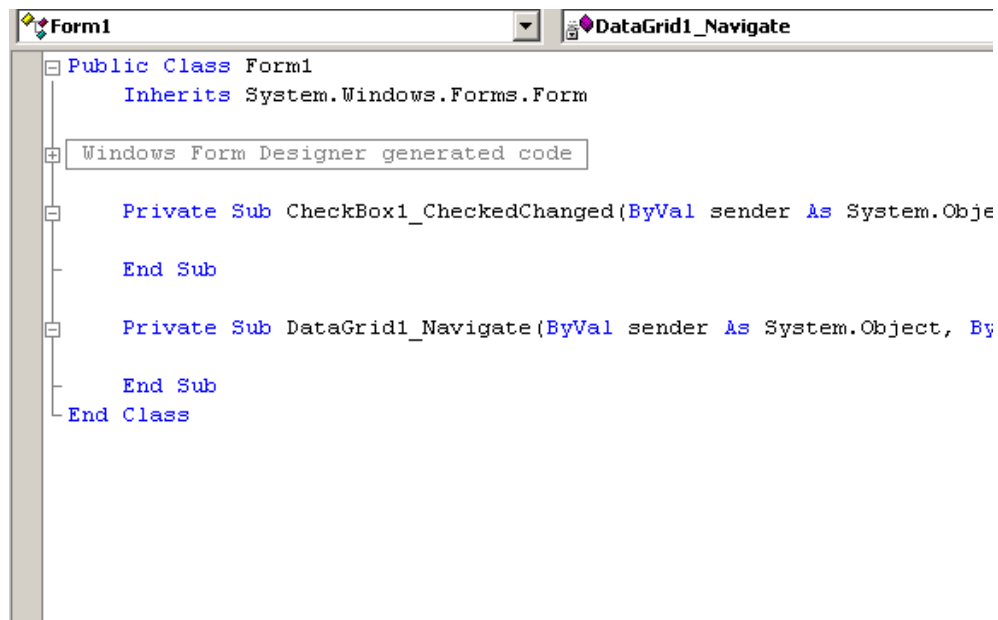
The class view window like the image below is the window that presents solutions and projects in terms of the classes they contain and the members of these classes. Using the class view window also helps us to find a member of a class that we want to work with. As you can notice from the image, the class view window displayed all the methods and events for the controls which were available on the form.



**Fig Class View Window**

### 11 Code Designer Window

Code Designers like the image below allows us to edit and write code. This is the window that opens when we double-click on a form or any control. This is the place where we write all the code for the application. Notice the two drop-down list boxes at the top of the code window in the image below. The left box allows us to select the object's code we are working with and the right box allows us to select the part of code that we want to work. Also notice the "+" and "-" boxes in the code designer. You can use those boxes to display code Visual Basic .NET already created, like, Windows Forms Designer generated code, etc.



**Fig Class View Window**

## **12 Dynamic Help Window**

The dynamic help window displays help which looks up for things automatically. For example, if you want to get help with a form, select the form and select Help->Dynamic Help from the main menu. Doing that displays all the information relating to forms. The image below displays that. You can get help relating to anything with this feature. Say, if you want to know more about the form, select the form and select Dynamic Help from the Help menu. Doing that displays information about the form

## **13 Command Window**

The command window in the image below is a useful window. Using this window we can add new item to the project, add new project and so on. You can view the command window by selecting View->Other Windows->Command Window from the main menu. The command window in the image displays all possible commands with File.

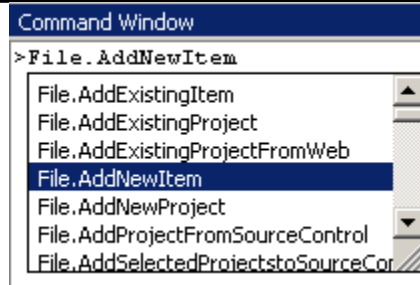


Fig Command Window

## 14 Object Explorer Window

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

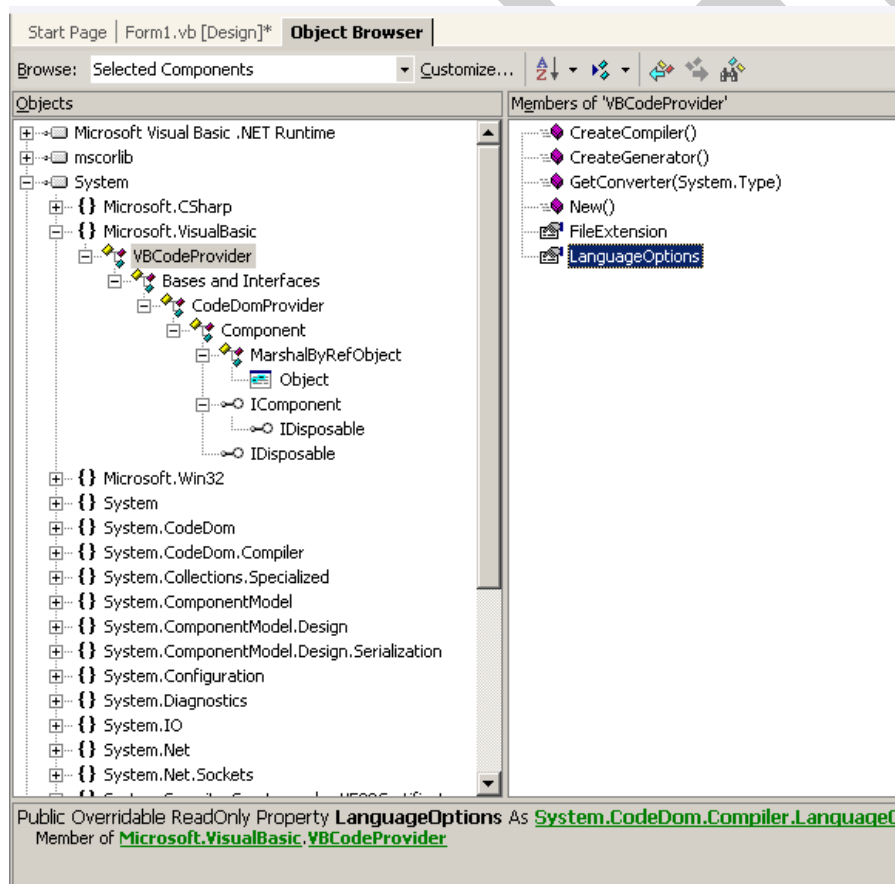
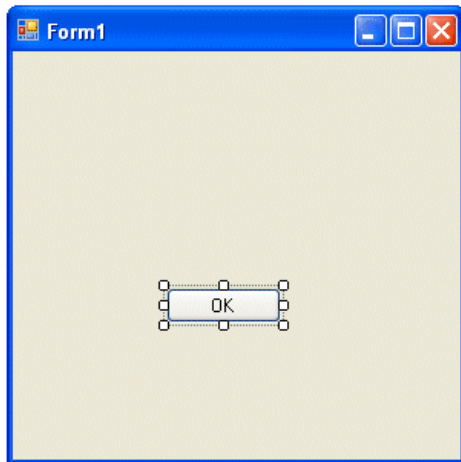


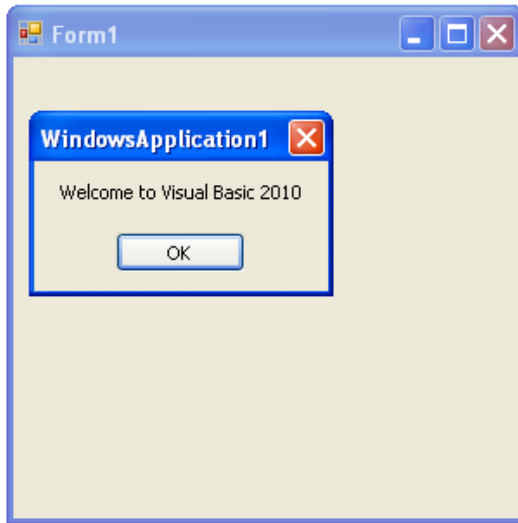
Fig Object Explorer Window

### Creating a simple VB.NET Program

To create your first program, drag the button control into the form, and change its default Text **Button1** to **OK** in the properties window, the word OK will appear on the button in the form, as shown below



2. Double click the button and open the code window. The code window opens as shown in the following figure.
3. Type the code as shown below.
4. MsgBox"Welcome to Visual Basic 2010"
5. Compile and Run your program (Press F5 Key)
6. The output for the program is shown below.



## VB.NET Environment

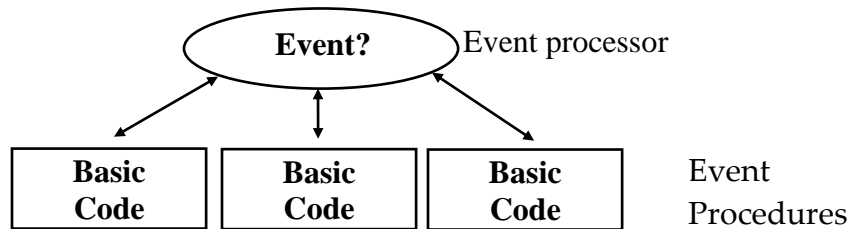
The Integrated Development Environment (IDE) is the development environment for all Net based applications. The three steps for planning a Visual Basic application involve setting up the user interface, defining the properties, and then creating the code. The three steps for planning a Visual Basic application involve setting up the user interface, defining the properties, and then creating the code.

- Design the user interface. When you plan the user interface, you draw a sketch of the screens the user will see when running your project. On your sketch, show the forms and all the controls that you plan to use. Indicate the names that you plan to give the form and each of the objects on the form.
- Plan the properties. For each object, write down the properties that you plan to set or change during the design of the form.
- Plan the Basic code. In this step you plan the procedures that will execute when your project runs. You will determine which events require action to be taken and then make a step-by-step plan for those actions.

## Event Driven Programming

Visual Basic.Net is event-driven programming. It means that the code will not be executed until events are triggered (mouse click, button pressing, menu selection,). Visual Basic.Net is governed by an event processor. Nothing happens until an event is detected. Once an event is detected, the code corresponding to that event (event procedure) is executed. Program control is then returned to the event processor.



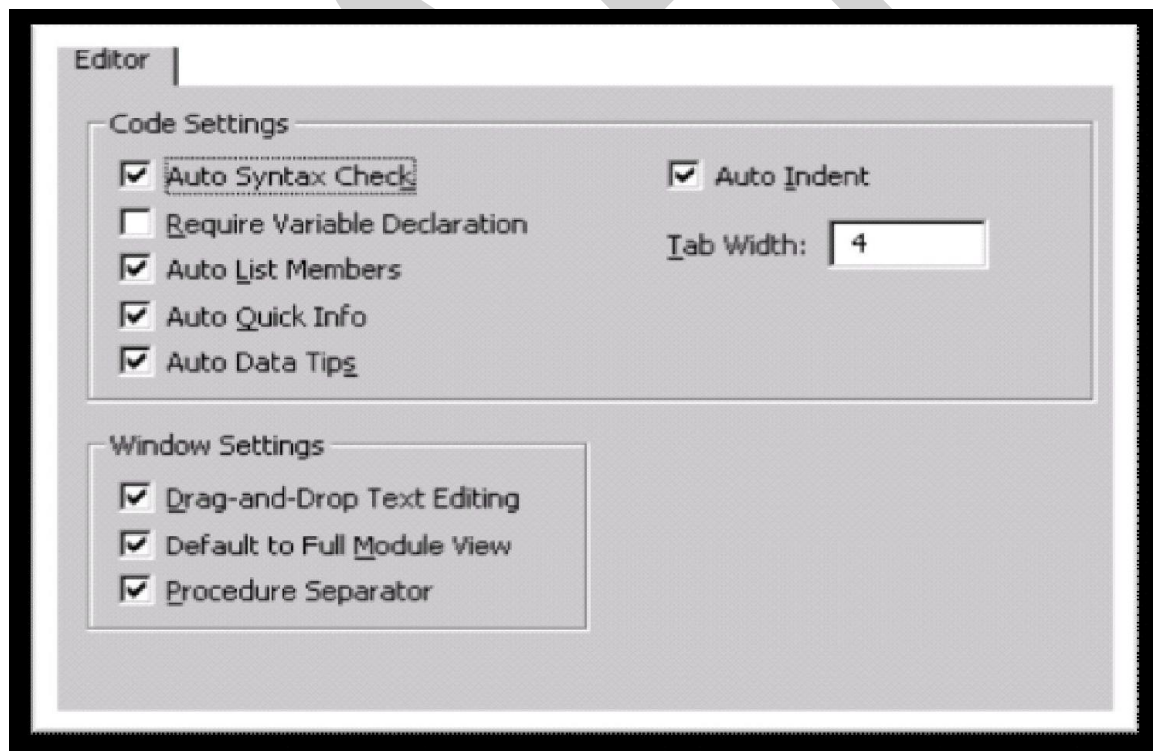


**Fig Event Driven Concept**

All Windows applications are event-driven. For example, nothing happens in Word until you click on a button, select a menu option, or type some text. Each of these actions is an event. The event-driven nature of Visual Basic makes it very easy to work with. As you develop a Visual Basic application, event procedures can be built and tested individually, saving development time. And, often event procedures are similar in their coding, allowing re-use (copy and paste).

### The Editor Tab and Format Tab

The Editor Tab and Format Specifies the code window and project window settings.



### Tab Options

- Auto Syntax Check — Determines whether Visual Basic should automatically verify correct syntax after you enter a line of code.
- Require Variable Declaration — Determines whether explicit variable declarations are required in modules. Selecting this adds the Option Explicit statement to general declarations in any new module.
- Auto List Members — Displays a box that displays information that would logically complete the statement at the current insertion point.
- Auto Quick Info — Displays information about functions and their parameters.
- Auto Data Tips — In the Code window while in Break mode, displays the value of the variable or object property over which your cursor is placed. The display is limited to variables and objects that are in the current scope.

### Editor Format Tab

Editor Format Tab Specifies the appearance of your Visual Basic code.

### Tab Options

Code Colors Determines the foreground and background colors used for the type of text selected in the list box.

- Text List — Lists the text items that have customizable colors.
- Foreground — Specifies the foreground color for the text selected in the Color Text List.
- Background — Specifies the background color for text selected in the Color Text List.
- Indicator — Specifies the margin indicator color.
- Font Specifies the font used for all code.
- Size Specifies the size of the font used for code.

### General Tab

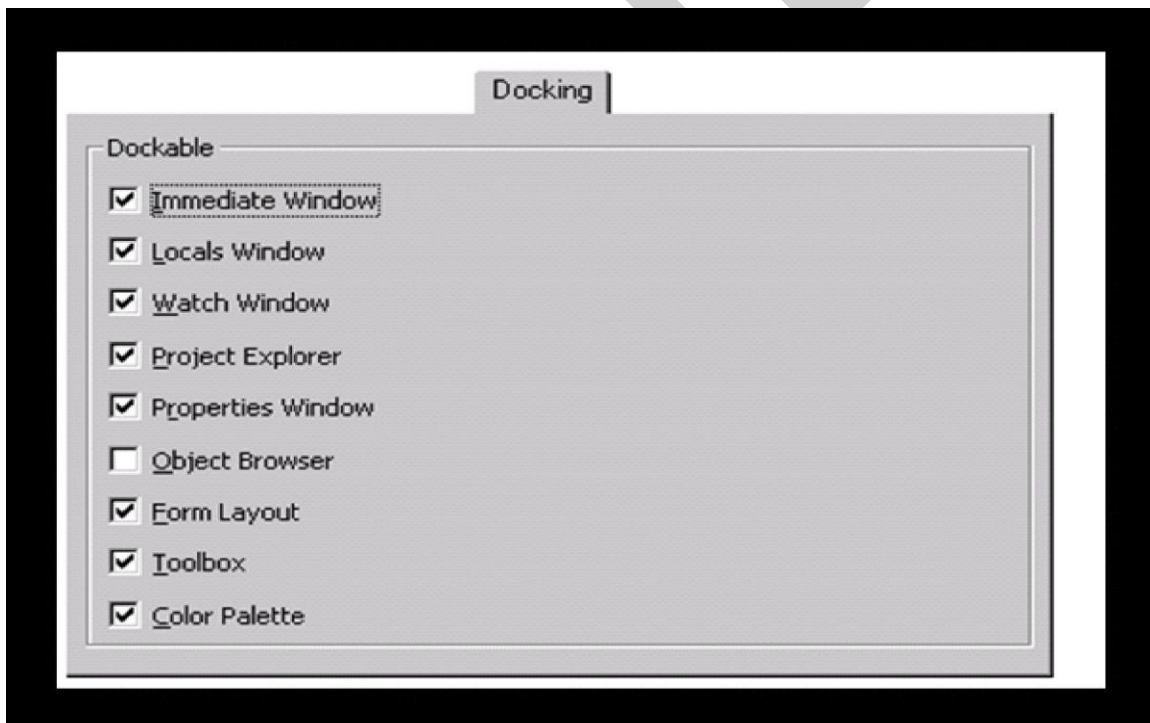
Specifies the settings, the error handling, and compile settings for your current Visual Basic project.

### Form Grid Settings

- Determines the appearance of the form grid at design time.
- Show Grid — Determines whether to show the grid at design time.
- Grid Units — Displays the grid units used for the form. The default is points.
- Width — Determines the width of grid cells on a form (2 to 60 points).
- Height — Determines the height of grid cells on a form (2 to 60 points).
- Align Controls to Grid — Automatically positions the outer edges of controls on grid lines.

### Docking Tab

Allows you to choose which windows you want to be dockable.



A window is docked when it is attached or “anchored” to other windows that are dockable or to the mainwindow when you are in MDI mode. When you move a dockable window, it “snaps” to the location. A window is not dockable when you can move it anywhere on the screen and leave it there.

### Tab Options

#### Dockable

- Lists the windows that are dockable.

- Select the windows you want to be dockable and clear those that you do not. You can have any, none, or all of the windows in the list dockable

### Methods and Events

In Visual Basic you will work with objects, which have properties, methods, and events. Each object is based on a class.

- **Objects:** Think of an **object** as a thing, or a noun. Examples of objects are forms and controls. **Forms** are the windows and dialog boxes you place on the screen; **controls** are the components you place inside a form, such as text boxes, buttons, and list boxes.
- **Properties:** **Properties** tell something about an object, such as its name, color, size, location, or how it will behave. You can think of properties as adjectives that describe objects. When you refer to a property, you first name the object, add a period, and then name the property. For example, refer to the Text property of a form called Form1 as Form1.Text (pronounced “form1 dot text”).
- **Methods:** Actions associated with objects are called *methods*. **Methods** are the verbs of object-oriented programming. Some typical methods are **Move**, **Hide**, **Show**, and **Clear**. You refer to methods as Object.Method (“object dot method”). For example, a **Show** method can apply to different objects. **Form1.Show** shows the form object called Form1; **btnExit.Show** shows the button object called btnExit.
- **Events:** You can write procedures that execute when a particular event occurs. An **event** occurs when the user takes an action, such as clicking a button, pressing a key, scrolling, or closing a window. Events can also be triggered by actions of other objects, such as repainting a form or a timer reaching a preset point.
- **Classes:** A **class** is a template or blueprint used to create a new object. Classes contain the definition of all available properties, methods, and events.

### Points to Ponder

- Programming language acts as an interface between computer and the programmer
- Compute Languages can be broadly divided into three types namely high level languages, Assembly level languages and Low level languages

- High level languages are so powerful which are widely used by the humans to communicate with the computer
- Visual basic.Net is a good example for High level programming language.
- .Net framework is a platform or an environment in which the user creates his applications
- .NET Framework is a collection of services and classes.
- .NET Framework is a layer between the operating system and the programming language.
- The .NET Framework consists of two main components namely Common Language Runtime (CLR) and Class Libraries
- The CLR is also known as the "execution engine" of .NET
- A namespace is a collection of different classes. All VB applications are developed using classes from the .NET System namespace
- All namespaces are based on System namespace
- Assemblies is a collection of types and resources that forms a logical unit of functionality
- The Integrated Development Environment (IDE) is the development environment for all .Net based applications
- The Solution Explorer window gives an overview of the solution we are working with and lists all the files in the project
- The properties window allows us to set properties for various objects at design time.
- The toolbox window is the window that gives us access to all controls and components
- IntelliSense provides a list of options that make language references easily accessible and helps us to find the information we need

- The dynamic help window displays help which looks up for things automatically

**Possible Questions**

**Unit I (8 Marks)**

1. What is .NET? Explain .NET framework.
2. Write note on Components of .Net Framework.
3. Explain the different features of .Net.
4. What is namespace? What are default namespaces available in windows application?
5. Explain with example: - Class, Property, Methods, and Event.
6. What is IDE? Explain the IDE in VB.NET.
7. Explain the different tabs used in VB.Net

**KARPAGAM ACADEMY OF HIGHER EDUCATION**  
**COIMBATORE - 21**

**DEPARTMENT OF COMPUTER SCIENCE,CA & IT**

**CLASS : III B.Sc COMPUTER TECHNOLOGY**

**BATCH : 2016-2019**

**Part -A Online Examinations**

**I One Marks**

**SUBJECT: . NET Programming**

**SUBJECT CODE: 16CTU502A**

**UNIT-I**

S.No.	Question	Option 1	Option 2	Option 3	Option 4	Answer
1	.Net is a technology developed by _____ company	Microsoft	Sun	IBM	Apple	Microsoft
2	NGWS Stands for	Next	Next	Next	Next	Next
3	_____ is also known as the "execution engine" of .NET.	CLR	CTS	MSIL	WPF	CLR
4	Code that targets the Common Langage Runtime is known as	Distributed	Managed	Legacy	Native	Managed
5	compilers must conform source code compiled by a .NET	CLS	CTS	CLR	MSIL	CLS
6	Which of the following task is done by the Garbage collector?	Freeing	Closing	Freeing	Closing	Freeing
7	VB.Net is a _____ programming paradigm.	Procedural	Structured	Object	Monolith	Object
8	Data members of a class are by default _____	public	private	static	volatile	private
9	Member functions of a class are by default _____	public	private	static	volatile	public
10	IDE stands for	Internet	Integrated	Internet	Interface	Integrate
11	The final compiled version of a Project is _____	Form	Software	Compone	Files	Compone
12	_____ is a collection of files that can be compiled to create a	Form	Software	Compone	Project	Project
13	_____ is a collection of projects and files that composed an	Solution	Software	Forms	Project	Solution
14	Every object has a distinct set of attributes knowns as	members	datas	properties	methods	propertie
15	The property that must be set first for any new object is the	Name	Colour	Size	Binding	Name
16	Objects that can be placed on a form are called _____	Pictures	Tools	Buttons	Controls	Controls
17	Controls that do not have physical appearance are called	invisible-at-	visible-at-	virtual	physical	invisible-
18	_____ is a template procedure in which we add the code that	code behind	coding	event	class	event
19	By default Visual Studio saves all Projects in the folder _____	\My	wwwroots\	Either of	Both the	\My
20	The Design window appears _____ by default.	Auto-	Docked	Floating	Closed	Docked

21	_____ windows appears attached to the side, top or bottom	Auto-	Docked	Floating	Closed	Docked
22	_____ can be distributed to other people/computer and do not	Files	Forms	Projects	Compone	Compone
23	_____ are also called programs	Distributabl	Project	Solution	Forms	Distribut
24	_____ is a programming structure that encapsulates data	Class	Object	Collection	methods	Object
25	This is the way we refer to properties of an object in code	{ Object	{ Class }. { P	{ Class }. {	{ Object	{ Object
26	To set a property to some value use _____	(dot) .	(equal)=	(*) astrick	set()	(equal)=
27	When we type the period(dot) after the object name a small	IntelliSense	OnlineHel	QuickMe	DropHel	IntelliSen
28	A property that returns an object is called	Collection	subroutine	Object	Object	Object
29	The process of creating an object is called _____	integration	instantiatio	interfacin	inheritan	instantiati
30	Event driven programs have logical sections of code placed	functions	methods	events	subroutin	events
31	Events can be triggered by	User	Calling	Both	None	Calling
32	An Event that continuously triggers itself is called _____	Looping	repititive	recursive	Nested	recursive
33	_____ is a statement that defines the structure of an event	event	event	event call	event	event
34	The items within the paranthesis of an event declaration are	objects	parameters	properties	events	parameter
35	The _____ parameter returns a reference to the control that	event args	sender	caller	Trigger	sender
36	_____ events eventually exhaust Windows' resources until	Recursive	Repititive	Simple	Continuo	Recurshiv
37	This property is used to change/display the titile of the form	Name	Text	Title	Form	Text
38	The default BackColor of the Form is the system color named	gray	white	pale	control	control
39	A _____ sign at the left of the propertyname in property	+	-	*	x	+
40	The default value of FormBorderStyle property is	FixedSingle	FixedTool	Sizable	SizableT	Sizable
41	Without the _____ the form cannot be resized by the user	Minimize /	Border	Title bar	Control	Border
42	Without the _____ the form cannot be repositioned by the	Minimize /	Border	Title bar	Control	Title bar
43	When the FormBorderStyle property is set to _____ the form	None	FixedTool	Fixed3D	All the	All the
44	WindowState property is _____ by default	Normal	Maximized	Minimuze	None	Normal
45	The Form can be displayed by	by calling	by setting	Both	None	Both (a)
46	_____ method does not simply hides the form, but destroy it	Close()	Hide()	Distroy()	Remove(	Close()
47	A control can be added to a Form by	double	drag a	Select a	All the	All the
48	The Forms Icon is displayed in the	forms	taskbar if	tasklist	All the	All the



49	Which property has to be set to minimize maximize ot restore a	Windows	WindowSt	FormBord	Window	Window
50	Which property is used to change the image of the pointer.	Pointer	Image	icon	Cursor	Cursor
51	Lasso is a techniques of _____	selecting a	clearing a	Aligning	Setting a	selecting
52	Options to Align a group of controls, Make them same size and	Align	Control	Format	Window	Format
53	The control with the tab index _____ first gets focus when the	0	1	Maximum	Minimu	0
54	Transparent forms can be created by setting the _____	Visibility	Transparen	Opacity	Sizable	Opacity
55	Which is the default event of a Button Control	Click	MouseOve	TextChan	GetFocus	Click
56	To change the Height if the text box _____ property has to	Height	Size	Multiline	LineLeng	Multiline
57	What increment of time is applied to the interval property of the	Seconds	Millisecon	Nanoseco	minutes	Milliseco
58	_____ property returns the index of the currently selected	ItemSelecte	Item.Index	SelectedI	IdexSelec	SelectedI
59	_____ property returns the number of items in a List View	Count	List	Number	Items	Count
60	Each item in a Tree View is called _____	branch	subtree	leaf	node	node

**UNIT II**

The VB .Net Language: The VB .Net Language – Variables- declaring variables, Data type of variables, forcing variables declarations, scope & lifetime of a variable, constants, arrays, types of arrays, control array, Structure programming – Modularity – Information hiding – abstraction – events – subroutines and functions – message box – input box. Control flow statement: conditional statement, loop statement.

**Introduction to Visual Basic.NET**

Visual Basic .NET is an object-oriented computer programming language that can be viewed as an evolution of the classic Visual Basic (VB), which is implemented on the .NET Framework. VB.Net provides the easiest, most productive language and tool for rapidly building Windows and Web applications. Visual Basic .NET comes with enhanced visual designers, increased application performance, and a powerful integrated development environment (IDE). It also supports creation of applications for wireless, Internet-enabled hand-held devices. The following are the difference between traditional Visual basic 6.0 and Visual basic.Net

S.No	Visual Basic 6.0	Visual Basic.Net
1	It is a object based language	It is a object oriented programming language
2	It does not support Inheritance	As it is object oriented, it supports all oops concepts such as inheritance
3	Visual basic is a connected architecture	VB.Net supports disconnected architecture
4	There is no support for Threading	It supports Multithreading
5	VB 6.0 does not provide support for Exception handling	It supports Exception handling, which is an added advantage

**Table Difference between visual basic 6.0 and VB.Net**

These are the major difference between these two languages. The next section details about the powerful features of Visual basic.Net.

**Features of Visual Basic.Net**

The integrated development environment (IDE) of Visual Studio has definitely changed with the arrival of .NET technology. Due to these changes, Visual basic too has some changes such as Property pages are replaced by property grid displays, Objects replaces variant data type and the addition of Designers. These changes make the application very simple for the developers. This sections provides some of the significant features of the latest version of Visual basic.

### **1. Simplified Deployment**

With Visual Basic .NET we can build applications more rapidly and deploy and maintain them with efficiency. Visual Basic .NET 2003 and .NET Framework 1.1 makes "DLL Hell" a thing of the past. Side-by-side versioning enables multiple versions of the same component to live safely on the same machine so that applications can use a specific version of a component. XCOPY-deployment and Web auto-download of Windows-based applications combine the simplicity of Web page deployment and maintenance with the power of rich, responsive Windows-based applications.

### **2. Powerful Windows-based Applications**

Visual Basic .NET comes with features such as a powerful new forms designer, an in-place menu editor, and automatic control anchoring and docking. Visual Basic .NET delivers new productivity features for building more robust applications easily and quickly. With an improved integrated development environment (IDE) and a significantly reduced startup time, Visual Basic .NET offers fast, automatic formatting of code as you type, improved IntelliSense, an enhanced object browser and XML designer, and much more.

### **3. Building Web-based Applications**

With Visual Basic .NET we can create Web applications using the shared Web Forms Designer and the familiar "drag and drop" feature. You can double-click and write code to respond to events. Visual Basic .NET 2003 comes with an enhanced HTML Editor for working with complex Web pages. We can also use IntelliSense technology and tag completion, or choose the WYSIWYG editor for visual authoring of interactive Web applications.

### **4. Simplified Data Access**

You can tackle any data access scenario easily with ADO.NET and ADO data access. The flexibility of ADO.NET enables data binding to any database, as well as classes, collections, and arrays, and provides true XML representation of data. Seamless access to ADO enables simple data access for connected data binding scenarios. Using ADO.NET, Visual Basic .NET can gain high-speed access to MS SQL Server, Oracle, DB2, Microsoft Access, and more.

## **5. Improved Coding**

You can code faster and more effectively. A multitude of enhancements to the code editor, including enhanced IntelliSense, smart listing of code for greater readability and a background compiler for real-time notification of syntax errors transforms into a rapid application development (RAD) coding machine.

## **6. Direct Access to the Platform**

Visual Basic developers can have full access to the capabilities available in .NET Framework 1.1. Developers can easily program system services including the event log, performance counters and file system. The new Windows Service project template enables to build real Microsoft Windows NT Services. Programming against Windows Services and creating new Windows Services is not available in Visual Basic .NET Standard, it requires Visual Studio 2003 Professional, or higher.

## **Variable and Data type**

Every programming language needs some ways to organize the data that is used in the user programs. The input which is given by the user should be stored temporarily and to be processed by the program where ever the data is required. Data type and variables are used for organizing user data. Data type in a programming language describes that what type of data a variable can hold and variables are used to store the data temporarily until the data is used by the program.

### **1 Character set**

Character set is nothing but the set of characters allowed and supported in the programming language. Generally a program is a collection of instructions, which contains group of characters. Only a limited set of characters is allowed to write instructions in the program.

<b>S.No</b>	<b>Character Set</b>	<b>Contents</b>
<b>1</b>	Alphabets	A - Z or a - z
<b>2</b>	Digits	0 – 9
<b>3</b>	Special Characters	+ - * / % . , ; ' "   ! \ ~ > < = ( ) { } [ ] # & ^ _ ?

### **2 Identifiers**

An identifier is nothing but the name that is used for naming the variables, Objects etc. In VB.NET, the identifier must begin with either a letter or underscore ('\_'). If an identifier begins with an underscore, it must contain at least one other valid identifier character to disambiguate it from a line continuation. Regular identifiers may not match keywords, but escaped identifiers can match with keywords. An escaped identifier is an identifier delimited by square brackets. Escaped identifiers follow the same rules as regular identifiers except that they may match keywords and may not have type characters. Although Visual Basic .NET is not case sensitive, the case of identifiers is preserved when applications are compiled. When using Visual Basic .NET components from case-sensitive languages, the caller must use the appropriate case.

### 3 Keywords

Keywords are nothing but the words that have some pre-defined meaning. We It is impossible to use these words as identifiers of a program. In Visual Basic.Net the keywords are reserved and it is even impossible to use then as name of subroutines. The Keywords in Visual Basic.Net are shown below.

S.No	Keyword	Usage
1	And	Boolean operator
2	AndAlso	Boolean operator
3	Ansi	Used in the declare statement
4	Append	Used as a symbolic constant
5	As	Used in the variable declaration statement
6	Assembly	Attribute specifier
7	Binary	Option Compare statement
8	Boolean	Variable declaration
9	Byte	Variable declaration
10	Byval	Argument lists

### 4 Variable and Rules

An identifier is used to identify and store some value. If the value of the identifier is changed during the execution of the program, then the identifier is known as variable. When the variable holds numeric data, it is called as numeric variable and when it holds character(s) it is called as Character/String variable. To name a variable in Visual Basic 2010, you have to follow a set of rules.

1. It must be less than 255 characters
2. No spacing is allowed
3. It must not begin with a number
4. Period is not permitted

Examples of valid and invalid variable names are displayed in Table

<b>Valid Variable Name</b>	<b>Invalid Variable Name</b>
Name	Name of student
Student_name	1name
Student123	Student-name
Number1, number2	Number 1

## **5 Declaring Variables**

Before using a variable in the program, it must be declared. The declaration of variable tells the compiler about the type of data to be held by the variable and the memory to be reserved for it and to store the initial value in it. Variables are normally declared in the general section of the codes' windows using the Dim statement. Dim is a keyword. Dim stands for Dimension. The format is as follows:

### **Dim Variable Name As Data Type**

Some of the valid variables are as follows

```
Dim i as integer      // Integer variable declaration
Dim name as string    // String variable declaration
Dim x, y as integer   // Two variables are used in the same declaration
Dim x as double       // Real variable declaration
Dim doDate As Date    // Date is a special data type for Date function
```

It is possible to combine two or more variables in a single statement. The variables names can be separated by using a comma as shown in the above statement. There are two possible formats are available for declaring string such as one for the variable-length string and another for the fixed-length string. For the variable-length string, just use the same format as shown above. However, for the fixed-length string the total number of characters should be specified at the declaration time. The format for fixed length string is

### **Dim VariableName as String \* n**

Where n defines the total number of string that a string can hold.

```
Private Sub Button1_Click (ByVal sender As System.Object, ByVal e_
As System.EventArgs) Handles Button1.Click
```

```
Dim i, j, k As Integer
```

```
    i = 20
```

```
    j = 20
```

```
    k = i + j
```

```
    TextBox1.Text = k
```

```
    Dim a, b, c As String
```

```
    a = " Welcome"
```

```
b = " To "  
c = "Visual Basic"  
TextBox2.Text = a + b + c  
End Sub
```

In the above example, The variable C produces the output string “Welcome to Visual Basic”. Here + symbol denotes Concatenation operator.

## 6 Variable initialization

After declaring various variables using the Dim statements, it is possible to assign values to those variables. The general format of an assignment is

**Variable=Expression**

The variable can be a declared variable or a control property value. The expression could be a mathematical expression, a number, a string, a Boolean value (true or false) and etc. The following are some examples:

```
Number1 = 100  
Number2 = Number1-45  
Username = “AnandKumar”  
Button1.Visible = True  
Label4.Caption = textbox1.Text  
Number3 = Val (Textbox1..Text)
```

## 7 New and Nothing Keyword

The New is a Keyword which is used to create the instance of the class. Unlike value types, such as Integer and Double, objects are reference types, and it should be created explicitly before using them in the program. It is also possible to create an instance of the form as well as all the controls. For example Button is control and the class for button is Button. So the instance can be created in such a way as shown below

```
Dim Button1 As System.Windows.Forms.Button  
Dim Button2 As New System.Windows.Forms.Button()  
Dim frm As New System.Windows.Forms.Form1()
```

In the above statements, the first statement declares an object variable that can contain a reference to a button object. However, the variable Button1 contains the value nothing until you assign an object of type Button to it. The second statement also defines a variable that can contain a button object, but the New keyword creates a button object and assigns it to the variable Button2. Both forms and controls are actually classes; the New keyword can be used to create new instances of these items as needed.



The Nothing keyword represents the default value of any data type. Assigning Nothing to a variable sets it to the default value for its declared type. If that type contains variable members, they are all set to their default values. The following example uses the Nothing keyword

```
Private Sub Button1_Click (ByVal sender As System.Object, ByVal e_ As  
System.EventArgs) Handles Button1.Click
```

```
    Dim i as integer  
    Dim s as Boolean  
    i=Nothing           //sets i to 0  
    s= Nothing          //sets s to False  
End Sub
```

## 8 Implicit and Explicit declarations

There are two ways to declare the variables for the program. They are implicit declaration and Explicit declaration. In the implicit declaration, Visual basic automatically create the variable for the user application. Implicit declaration means that Visual Basic automatically creates a variant for each identifier it recognizes as a variable in an application

The second approach to declaring variable is to explicitly declare them with one of the following keywords Dim, Static, Private, and Public. The choice of keyword has a profound effect on the variable's scope within the application and determines where the variable can be used in the program

```
Dim VariableName as DataType  
Static VariableName as DataType  
Private VariableName as DataType  
Public VariableName as DataType
```

Visual Basic.Net reserves the amount of memory required to hold the variable as soon as the declaration statement is executed. After a variable is declared, it is not possible to change its data type, although it is quite easy to convert the value of a variable and assign the converted value to another variable

## 9. Scope of Variable

The scope of a variable referred to as accessibility of a variable. It includes the information about the variable such as where the variable can be read from and/or written to, and the variable's lifetime, or how long it stays in memory.. Variables can be declared in four different locations in the programs as shown in the following table.

S.No	Location	Usage
------	----------	-------

1	Block	If the variable is declared within a control statement such as an If statement, then that variable's scope is only until the end of the block. The lifetime is until the procedure ends.
2	Procedure	If the variable is declared within a procedure, but outside of any If statement, then the scope is until the End Sub or End Function. The lifetime of the variable is until the procedures ends
3	Module/Class	Variables can be declared outside of any procedure, but it must be within a Class...End Class or Module...End Module statement. The scope is any procedure within this module. The lifetime for a variable defined within a class is until the object is cleaned up by the garbage collector. The lifetime for a variable defined within a module is until the program ends.
4	Project	It is possible to declare a Public variable within a Module...End Module statement, and that variable's scope will be any procedure or method within the project. The lifetime of the variable will be until the program ends.

**Table Scope of the variables**

There are many different ways you can declare variables. Variables can be declared with any one of the scope such as public, private, protected, friend and protected friend. If the variable is declared as public then the variable will be available anywhere in or outside of the project. If it is declared as private then it is visible only within the block where it is declared. When the variable is declared as protected then it can be used in the class where defined and within any inherited class. When used with friend it can only be accessed by code in the same project/assembly. If the variable is declared as protected friend then it has the combination of protected and friend.

## **10 Data Types**

Before writing any program we have to decide the variables that are going to be used in the program, the purpose of the variable and type of data it should. Visual Basic classifies the information mentioned above into two major data types such as numeric data types and the non-numeric data types.

### **1 Numeric Data Types**

Numeric data types are types of data that consist of numbers, which can be computed mathematically with various standard operators such as addition, Subtraction, multiplication and division. Some of the examples of numeric data types are height, weight, student marks, class strength, monthly bills, fees etc. In Visual basic, numeric data are divided into 7 types, depending on the range of values they can store. Calculations that only involve round figures or data that don't need precision can use Integer or Long integer in the computation. Programs that require high precision calculation need to use Single and Double decision data types, they are also called floating point numbers. For currency calculation, currency data types can be used. Lastly, if even more precision is requires performing calculations that involve a many decimal points, decimal data types can be used.

S.No	Data Type	Size	Range of Values
1	Byte	1 byte	0 to 255
2	Integer	2 bytes	-32,768 to 32,767
3	Long	4 bytes	-2,147,483,648 to 2,147,483,648
4	Single	4 bytes	-3.402823E+38 to -1.401298E-45 for negative values 1.401298E-45 to 3.402823E+38 for positive values.
5	Double	8 bytes	-1.79769313486232e+308 to -4.94065645841247E-324 for negative values 4.94065645841247E-324 to 1.79769313486232e+308 for positive values.
6	Currency	8 bytes	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
7	Decimal	12 bytes	+/- 79,228,162,514,264,337,593,543,950,335 if no decimal is use +/- 7.9228162514264337593543950335 (28 decimal places).

## 2 Non-numeric Data Types

Nonnumeric data types are data that cannot be manipulated mathematically using standard arithmetic operators. The non-numeric data comprises text or string data types, the Date data types, the Boolean data types that store only two values (true or false), Object data type and Variant data type .

S.No	Data Type	Size	Range of Values
1	String(fixed length)	Length of string	1 to 65,400 characters
2	String(variable length)	Length + 10 bytes	0 to 2 billion characters
3	Date	8 bytes	January 1, 100 to December 31, 9999
4	Boolean	2 bytes	True or False
5	Object	4 bytes	Any embedded object

6	Variant(numeric)	16 bytes	Any value as large as Double
7	Variant(text)	Length+22 bytes	Same as variable-length string

## **Operators in VB.NET**

Operators are symbols (characters or keywords) that specify operations to be performed on one or two operands. Operators that take one operand are called unary operators. Operators that take two operands are called binary operators. Unary Visual basic supports many operators that are described in the following sections.

1. Arithmetic Operators
2. Relational Operators
3. concatenation Operators
4. Bitwise Operators
5. Logical Operators
6. Unary Operators

### **1 Arithmetic Operators**

The arithmetic operators perform the standard arithmetic operations on numeric values. Computer performs mathematical calculations much faster than humans. However, computer itself will not be able to perform any mathematical calculations without receiving instructions from the user. It is necessary to instruct the computer to perform mathematical calculations such as addition, subtraction, multiplication, division and other kinds of arithmetic operations. In order for VB2010 to carry out arithmetic calculations, we need to write code that involves the use of various arithmetic operators. The VB2010 arithmetic operators are very similar to the normal arithmetic operators, only with slight variations. The plus and minus operators are the same while the multiplication operator uses the \* symbol and the division operator uses the / symbol. The list of VB2010 arithmetic operators are given below.

<b>S.No</b>	<b>Operator</b>	<b>Function</b>	<b>Example</b>
1	+	Addition	5+2=7
2	--	Subtraction	8-1=7
3	^	Exponential	2^4=16
4	*	Multiplication	4*4=16
5	/	Division	12/4=3
6	Mod	Modulus (return the remainder from an integer division)	17 Mod 4=1    165 mod 10=5
7	\	Integer Division (discards the	19\4=4

		decimal places)	
--	--	-----------------	--

## 2. Relational Operators

The relational operators perform some comparison between two operands and return a Boolean value indicating whether the operands satisfy the comparison. The relational operators supported by Visual Basic .NET are:

S.No	Operators	Usage
1	=	Equality
2	<>	Inequality
3	<	Less than
4	>	Greater than
5	>=	Greater than or equal to
6	<=	Less than or equal to
7	TypeOf...Is	To check the instance of the class
8	Is	To Check the reference of the object.
9	Like	To check the patterns of operands

The operators such as equality, inequality, less than Greater then, Greater then or equal to and less than or equal to are almost used in every programming languages. The TypeOf...Is operator is defined to take a reference as its first parameter and the name of a type as its second parameter. The result is True if the reference refers to an object that is type-compatible with the given type-name; False if the reference is Nothing or if it refers to an object that is not type-compatible with the given type name. Use the TypeOf...Is operator to determine whether a given object is an instance of a given class, is an instance of a class that is derived from a given class or exposes a given interface. The Is operator is defined for all reference types. The result is true if the references refer to the same object; false if not.

## 3 Concatenation Operators

Concatenation operators are used to concatenate multiple strings into a single string. There are two concatenation operators available in visual basic as shown in the table below.

S.No	Operators	Usage
1	+	String Concatenation
2	&	String Concatenation

## 4 Logical / Bitwise Operators

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

S.No	Operators	Usage
1	Not	Negation
2	And	Conjunction
3	AndAlso	Conjunction
4	Or	Disjunction
5	OrElse	Disjunction
6	Xor	Disjunction

### 5 Unary Operators

Operators that take one operand are called unary operators. VB.Net provides the set of unary operators as shown the table below.

S.No	Operators
1	Unary Plus (+)
2	Unary Minus(-)
3	Not (Logical)
4	AddressOf

The unary plus operator takes any numeric operand. It's not of much practical use because the value of the operation is equal to the value of the operand. The unary minus operator takes any numeric operand (except as noted later). The value of the operation is the negative of the value of the operand. In other words, the result is calculated by subtracting the operand from zero. If the operand type is Short, Integer, or Long, and the value of the operand is the maximum negative value for that type, then applying the unary minus operator will cause a `System.OverflowException` error, as in the following code fragment. The logical negation operator takes a Boolean operand. The result is the logical negation of the operand. That is, if the operand is `False`, the result of the operation is `true`, and vice versa. The `AddressOf` operator returns a reference to a method. The Unary

### Constants

If the value of the variable is not changed during the execution of the program, then the variable is called as constant. For example, take a student's information of a class, `student-name` is a variable, because it will change to all the students. But the `class-name` is a constant and it will not be changed for the particular class students. The syntax for declaring constant is

### Const Constant Name As Data Type = Value

Constants are different from variables in the sense that their values do not change during the running of the program. The format to declare a constant is

#### Example

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

```
Const I as integer=10
```

```
Dim j as integer = 20
```

```
I=i+1
```

```
J=j+2
```

```
Textbox1.text=I
```

```
Textbox2.text=j
```

```
End Sub
```

In this example `i=i+1` is impossible, since `i` is declared as constant which should not be changed during the execution of the program.

#### Arrays

A variable is used to store the value which may be used for further reference. In a simple variable only one value can be stored at a time. So how to store multiple values in the single variable?. The solution is Arrays. Array is a collection of elements or data items. All the elements must be same data type and they are stored in consecutive memory locations. It is a programming construct that stores data and allows user to access them by numeric index or subscript. Arrays in Visual Basic .NET inherit from the Array class in the System namespace. All arrays in VB are zero based, meaning, the index of the first element is zero and they are numbered sequentially. The user must specify the number of array elements by indicating the upper limit or size of the array. The upper limit is the number that specifies the index of the last element of the array. Arrays are declared using Dim, ReDim, Static, Private, Public and Protected keywords. An array can have one dimension or more than one (multidimensional arrays). The dimensionality of an array refers to the number of subscripts used to identify an individual element. In Visual Basic the user can specify up to 32 dimensions. Arrays do not have fixed size in Visual Basic. The syntax for declaring an array is

```
Dim|Public|Private ArrayName(Subscript) As DataType
```

ArrayName is the name of the array. Subscript is the dimensions of the array.

DataType is any valid data type. Dim, Public, and Private declare the array and its scope.

Using Dim in a procedure will make the array only available from within that procedure.

Using it in the General Declarations section will make it available to all procedures in



that module. Private has the same effect and should be used only at the modular level. Using Public will make the array available throughout the project. Let us see a simple example by creating an array of size 5 to store student names.

```
Dim student (4) as String
student(0)="Anil"
student(1)="Binoy"
student(0)="Cheema"
student(0)="Dhoni"
student(0)="Farkhat"
```

Array is created with name as student. Size of the array is five. First name is stored in index of 0, second in 1 and so on. To retrieve the elements from the array the index can be used as shown below.

```
MessageBox..Show(student(0))
```

When the statement is executed, message box will appear that displays the first name from the array student. Instead of message box textbox can be used to display the result with the following code

```
TextBox1.text = student (1)
```

Program to store the student name in the array and to print the name in message box.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim student(4) As String
    student(0) = "Anil"
    student(1) = "Binoy"
    student(2) = "Cheema"
    student(3) = "Dhoni"
    student(4) = "Farkhat"
    MessageBox.Show(student(0))
End Sub
```



It is possible to declare an array without specifying the number of elements. The user must provide values for each element when initializing the array.

```
Dim Test () as Integer
Test=New Integer (){1,2,3,4,5}
```

It is possible to change the size of an array after creating them. The ReDim statement assigns a completely new array object to the specified array variable. ReDim statement is used to change the number of elements in an array.

```
Dim Myarray(5) as Integer
ReDim Myarray(10) as Integer
```

When the Redim statement is used to reinitialize the array, all the data contained in the array will be lost. If the user wants to preserve existing data when reinitializing an array then Preserve keyword should be used as shown below.

```
Dim Myarray(5) as Integer
ReDim Preserve Myarray(10) as integer
```

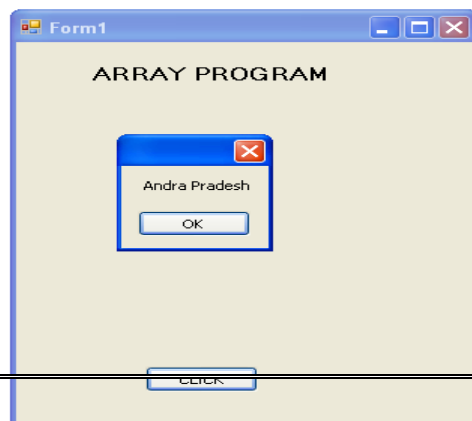
Program to store different states in India using array and displaying them using message box. Write the following code in the form load event.

```
Private Sub Form1_Load(ByVal sender As System. Object, ByVal e As
System.EventArgs) Handles MyBase.Load
Dim states (5) as String
Dim counter as Integer
States(0)="Andra Pradesh"
States(1)="Madhya Pradesh"
States(2)="Kerala"
States(3)="Tamil Nadu"
States(4)="Andra Pradesh"
States(5)="Hmachal Pradesh"
For
```

```
Next
End
```

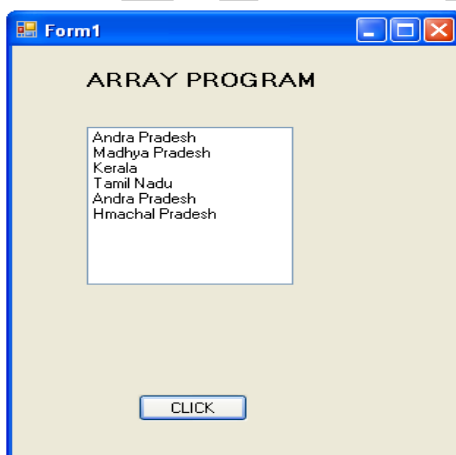
```
counter = 0 to states.count-1
MessageBox.Show(states(counter))
```

```
Sub
```



**Example** Program to store different states in India using array and displaying them using ListBox. Write the following code in the button click event.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    Dim states(5) As String
    Dim counter As Integer
    states(0) = "Andra Pradesh"
    states(1) = "Madhya Pradesh"
    states(2) = "Kerala"
    states(3) = "Tamil Nadu"
    states(4) = "Andra Pradesh"
    states(5) = "Hmachal Pradesh"
    For counter = 0 To states.Count - 1
        ListBox1.Items.Add(states(counter))
    Next
End Sub
```



## 2 Multi Dimensional Arrays

A multidimensional array is also known as array of arrays. A multidimensional array is the one in which each element of the array is an array itself. It is similar to tables of a database where each row is the collection of other secondary columns. If the secondary elements do not contain a collection of other

elements, it is called a 2-dimensional array, the most common type of multidimensional array, otherwise it is called an **n**-dimensional array where **n** is the depth of the chain of arrays.

There are two kinds of multidimensional arrays supported by the .NET framework: Rectangular arrays and Jagged arrays.

### a) Rectangular arrays

Rectangular arrays may be single-dimensional or multidimensional but always have a rectangular shape. Rectangular means that the length of each sub array in the same dimension is the same length. Array types are reference types, so the declaration of an array variable merely sets aside space for the reference to the array. Array instances are actually created via array initializes and array creation expressions

```
Dim rectArray(4, 2) As Integer  
Dim rectArray(,) As Integer = {{1, 2, 3}, {12, 13, 14}, {11, 10, 9}}
```

### Example

Program to create Rectangular array and print the array using console application of Visual Basic.Net

```
Public Sub RectangularArray()  
For i As Integer = 0 To rect.GetUpperBound(0)  
For j As Integer = 0 To rect.GetUpperBound(1)  
Console.Write(rect(i, j) & ControlChars.Tab)  
Next  
Console.WriteLine()  
Next  
End Sub
```

In the above example the rectangular array called rectarray is declared. It consists of 6 rows starting from 0 to 5 and 6 columns starting from 0 to 5. The next line of code simply prints the content of the array to the console.

**Note:** The elements of the array are not explicitly initialized. So all the elements by default will be 0 of Integer type.

### b) Jagged Arrays

Jagged Array is an array of arrays in which the length of each array can differ. A Very good example for Jagged arrays is creating a table in which the number of columns differ

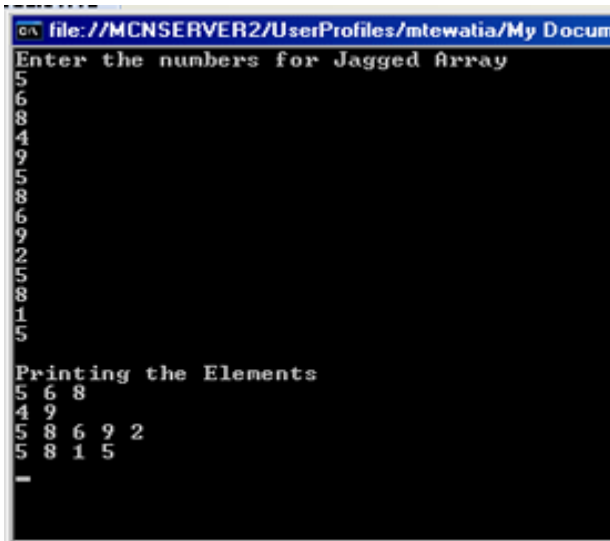
in each row. Say, if row1 has 3 columns, row2 has 3 columns then row3 can have 4 columns, row4 can have 5 columns and so on. The following code demonstrates jagged arrays. It is the another type of multidimensional array.

```
Dim colors(2,0) as String
colors(0)=New String(){ "Red","blue","Green"}
colors(1)=New String(){ "Yellow","Purple","Green","Violet"}
colors(2)=New String(){ "Red","Black","White","Grey","Aqua"}
```

**Example**

Program to demonstrate Jagged Array. The result is display using console application.

```
Public Class MyClass
    Public Shared Sub Main()
        Dim JaggedArray As Integer()() = New Integer(3)() {}
        JaggedArray(0) = New Integer(2) {}
        JaggedArray(1) = New Integer(1) {}
        JaggedArray(2) = New Integer(4) {}
        JaggedArray(3) = New Integer(3) {}
        Console.WriteLine("Enter the numbers for Jagged Array")
        For i As Integer = 0 To JaggedArray.Length - 1
            For x As Integer = 0 To JaggedArray(i).Length - 1
                Dim st As [String] = Console.ReadLine()
                Dim num As Integer = Int32.Parse(st)
                JaggedArray(i)(x) = num
            Next
        Next
        Console.WriteLine("")
        Console.WriteLine("Printing the Elements")
        For x As Integer = 0 To JaggedArray.Length - 1
            For y As Integer = 0 To JaggedArray(x).Length - 1
                Console.Write(JaggedArray(x)(y))
                Console.Write(vbNullChar)
            Next
            Console.WriteLine("")
        Next
        Console.ReadLine()
    End Sub
End Class
Output Window
```

A screenshot of a Windows application window titled 'file://MCNSERVER2/UserProfiles/mtewatia/My Docum'. The application has a black background with white text. It prompts the user to 'Enter the numbers for Jagged Array'. The user has entered the following numbers: 5, 6, 8, 4, 9, 5, 8, 6, 9, 2, 5, 8, 1, 5. Below the input, the application displays 'Printing the Elements' followed by the same sequence of numbers: 5 6 8, 4 9, 5 8 6 9 2, 5 8 1 5. A large, faint watermark 'K' is visible in the background of the screenshot.

```
file://MCNSERVER2/UserProfiles/mtewatia/My Docum
Enter the numbers for Jagged Array
5
6
8
4
9
5
8
6
9
2
5
8
1
5

Printing the Elements
5 6 8
4 9
5 8 6 9 2
5 8 1 5
```

## Dynamic Arrays

A dynamic array is an array which is used to specify the size at the run time. So the programmer can fix the size based upon the requirement. It allows the user to keep the number of elements in the array unspecified at the declaration time. Moreover, once declared, the number of elements can be altered later based on the requirement. It is an advantage over the static array. The wastage of memory space can be eliminated. Dim statement with empty parentheses can be used to declare a dynamic array. Dynamic arrays can be dimensioned or redimensioned as you need them with the ReDim statement. Preserve keyword can be used to preserve the data in an existing array when the user change the size of the last dimension.

```
Dim DynaStrings() As String
ReDim DynaStrings(10)
DynaStrings(0) = "String 0"
ReDim DynaStrings(100)
DynaStrings(50) = "String 50"
```

The upper bound of an array can be identified using the UBound function, which makes it easy to loop over all the elements in an array using a For loop. There are some limitations on dynamic arrays such as

- The dynamic part of the array must be of unpacked nature. A packed array can not be dynamic.
- The unpacked (and dynamic) part of the array must be one dimensional. More than one dimension is not allowed for dynamic arrays.

## Control Arrays

Control Arrays are arrays of controls sharing a common event handler. That is, you need to write code for only one event, which can handle other controls' events.

For example if you consider an application like calculator, where on the click event of the buttons from 0 to 9, you want to append the text to the visible text. So when you write code for all individual buttons, it is a time consuming process. Because we need the same piece of code for all these buttons, we can create one event handler to handle the events raised by all these buttons.

In Visual Basic 6, this was fairly simple. You have to copy and paste the control and confirm 'Yes' when asked, whether to create a control array. You can see the first control automatically gets an index of zero and the following controls get the index incremented by one from the last control. And if you double click on the buttons, you can see all these controls have same event handler, however you can notice a new argument, which is passed to the click event, named Index. This index property is the one, which will tell you which button is clicked (If you want to know, which one clicked). To create this while runtime, you can copy one control at design time and use this button to create other buttons. You can load a control and remove a control dynamically in VB6.

In Dot net the creation of control arrays are easier than the previous versions and are not created by copying and pasting, instead simply add the control events to the Handles list. You can give any name to this procedure.

Example

```
Private Sub ClickButton(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) Handles Button1.Click, _  
    Button2.Click, Button3.Enter  
    Dim btn As Button  
    btn = CType(sender, Button)  
    MsgBox(btn.Text)  
End Sub
```

In the above example ClickButton procedure is handling the click event of Button1 and Button2, whereas Enter event of the Button3. In order to check the control that is pressed, you need to convert the sender to the respective type. The CType function converts it into a button type, so that you can access the attributes of the event raised by the control.

## Handling Mouse Events in the Form

All Windows applications are event-driven. For example, nothing happens in Word until you click on a button, select a menu option, or type some text. Each of these actions is an



event. We can handle mouse events such as mouse pointer movements in Forms. The mouse events supported by VB .NET are as follows:

- **MouseDown:** This event happens when the mouse pointer is over the form/control and is pressed
- **MouseEnter:** This event happens when the mouse pointer enters the form/control
- **MouseUp:** This event happens when the mouse pointer is over the form/control and the mouse button is released
- **MouseLeave:** This event happens when the mouse pointer leaves the form/control
- **MouseMove:** This event happens when the mouse pointer is moved over the form/control
- **MouseWheel:** This event happens when the mouse wheel moves while the form/control has focus
- **MouseHover:** This event happens when the mouse pointer hovers over the form/control

The properties of the MouseEventArgs objects that can be passed to the mouse event handler are as follows:

**Button:** Specifies that the mouse button was pressed

**Clicks:** Specifies number of times the mouse button is pressed and released

**X:** The X-coordinate of the mouse click

**Y:** The Y-coordinate of the mouse click

**Delta:** Specifies a count of the number of detents (rotation of mouse wheel) the mouse wheel has rotated

### VB.NET String Class

(i)The Len Function:

The length function returns an integer value which is the length of a phrase or a sentence, including the empty spaces. The format is

**Len ("Phrase")**

or example, Len (VisualBasic) = 11 and Len (welcome to VB tutorial) = 22

The Len function can also return the number of digits or memory locations of a number that is stored in the computer. For example,

```
Private sub Form_Activate ( )
```

```
    X=sqr (16)
```

```
    Y=1234
```

```
    Z#=10#
```

```
    Print Len(x), Len(y), and Len (z)
```

**End Sub**

will produce the output 1, 4 , 8. The reason why the last value is 8 is because z# is a double precision number and so it is allocated more memory spaces.

(ii) The Right Function

The Right function extracts the right portion of a phrase. The format is

**Right ("Phrase", n)**

Where n is the starting position from the right of the phrase where the portion of the phrase is going to be extracted. For example,

Right("Visual Basic", 4) = asic

(iii) The Left Function

The Left\$ function extract the left portion of a phrase. The format is

**Left("Phrase", n)**

Where n is the starting position from the left of the phrase where the portion of the phrase is going to be extracted. For example,

Left ("Visual Basic", 4) = Visu

(iv) The Ltrim Function

The Ltrim function trims the empty spaces of the left portion of the phrase. The format is

**Ltrim("Phrase")**

.For example,

Ltrim (" Visual Basic", 4)= Visual basic

(v) The Rtrim Function

The Rtrim function trims the empty spaces of the right portion of the phrase. The format is

**Rtrim("Phrase")**

.For example,

Rtrim ("Visual Basic ", 4) = Visual basic

(vi) The Trim function

The Ttrim function trims the empty spaces on both side of the phrase. The format is

**Trim("Phrase")**

.For example,

Trim (“ Visual Basic ”) = Visual basic

(viii) The Mid Function

The **Mid** function extracts a substring from the original phrase or string. It takes the following format:

**Mid(phrase, position, n)**

Where position is the starting position of the phrase from which the extraction process will start and n is the number of characters to be extracted. For example,

Mid(“Visual Basic”, 3, 6) = ual Bas

(ix) The InStr function

The **InStr** function looks for a phrase that is embedded within the original phrase and returns the starting position of the embedded phrase. The format is

**Instr (n, original phase, embedded phrase)**

Where n is the position where the Instr function will begin to look for the embedded phrase. For example

Instr(1, “Visual Basic”, ” Basic”)=8

(x) The Ucase and the Lcase functions

The **Ucase** function converts all the characters of a string to capital letters. On the other hand, the **Lcase** function converts all the characters of a string to small letters. For example,

Ucase(“Visual Basic”) =VISUAL BASiC

Lcase(“Visual Basic”) =visual basic

(xi) The Str and Val functions

The **Str** is the function that converts a number to a string while the **Val** function converts a string to a number. The two functions are important when we need to perform mathematical operations.

(xii) The Chr and the Asc functions

The **Chr** function returns the string that corresponds to an ASCII code while the **Asc** function converts an ASCII character or symbol to the corresponding ASCII code. ASCII stands for “American Standard Code for Information Interchange”.

Altogether there are 255 ASCII codes and as many ASCII characters. Some of the characters may not be displayed as they may represent some actions such as the pressing of a key or produce a beep sound. The format of the Chr function is

**Chr(charcode)**

and the format of the Asc function is

**Asc(Character)**

The following are some examples:

Chr(65)=A, Chr(122)=z, Chr(37)=% , Asc("B")=66, Asc("&")=38

### Subroutines and Functions

A Procedure is a unit of code outside of the main execution code. But it can be executed by an invoking statement in the main execution code. There are 3 aspects about procedures:

1. Defining a procedure.
2. Invoking a procedure.
3. Exchanging data between the main execution code and a procedure.

VB offers two types of procedures:

1. Function Procedure - A procedure that returns a value explicitly.
2. Sub Procedure - A procedure that does not return any value explicitly.

### Defining and Invoking Function Procedures

A "Function" statement defines a function procedure with the following syntax:

```
Function function_name(argument_list)
    statement_block
    function_name = return_value
End Function
```

where "function\_name" is the name of the function, "argument\_list" a list of variables used to pass data into and/or out of the function, and "return\_value" is the value to be returned explicitly to the invoking statements. Of course, "argument\_list" is optional. Assigning the return value to the function name is also optional. If not given, default value will be returned to the invoking statements. But this is not recommended.

Invoking a function procedure is simple, no need of any special statements. Just use the function name with an argument list in any expression:

... *function\_name(argument\_list)* ...

This will cause the system to:

- Stop evaluating the expression.
- Map data or variables based on the argument list.
- Execute the entire statement block defined inside the function.
- Take the value returned in the function name.
- Continue to evaluate the expression.

If you want terminate a function procedure early, you can use the "Exit" statement:

### MsgBox() Function

Msgbox () function displays a message in a dialog box, waits for the user to click a button, and returns an Integer indicating which button is clicked by the user. Its syntax is MsgBox(prompt [, buttons] [, title])

The description of arguments in the function is as follows:

- *Prompt* Required. It is a string expression that will appear as the message in the dialog box.
- *Buttons* Optional. It is a numerical expression that sums the values used to specify the number and type of buttons to display, the icon style to use, the identity of the default button, and the modality of the message box. The default value for *buttons* is 0 when this argument is omitted.
- *Title* Optional. It is a string expression that will appear in the title bar of the dialog box. The application name is placed in the title bar when this argument is omitted.

The key *buttons* settings are:

Constant	Value	Description
<code>vbOKOnly</code>	0	Display <b>OK</b> button only.
<code>vbOKCancel</code>	1	Display <b>OK</b> and <b>Cancel</b> buttons.
<code>vbAbortRetryIgnore</code>	2	Display <b>Abort</b> , <b>Retry</b> , and <b>Ignore</b> buttons.
<code>vbYesNoCancel</code>	3	Display <b>Yes</b> , <b>No</b> , and <b>Cancel</b> buttons.
<code>vbYesNo</code>	4	Display <b>Yes</b> and <b>No</b> buttons.
<code>vbRetryCancel</code>	5	Display <b>Retry</b> and <b>Cancel</b> buttons.
<code>vbCritical</code>	16	Display <b>Critical Message</b> icon.
<code>vbApplicationModal</code>	0	Application modal; the user must respond to the message box before continuing work in the current application.
<code>vbSystemModal</code>	4096	System modal; all applications are suspended until the user responds to the message box.

The Return Values can be one of the values in the list below:

Constant	Value	Description
<code>vbOK</code>	1	OK
<code>vbCancel</code>	2	Cancel
<code>vbAbort</code>	3	Abort
<code>vbRetry</code>	4	Retry
<code>vbIgnore</code>	5	Ignore
<code>vbYes</code>	6	Yes
<code>vbNo</code>	7	No

An example of the `MsgBox ()` function is as follows:

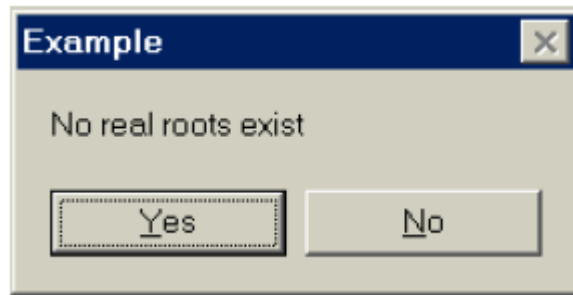
```
Response = MsgBox ("No real roots exist", VbYesNo, _
"Example")
```

```
If response = vbYes then
```

```
Exit Sub
```

```
End if
```

The `MsgBox()` function asks the user to respond by clicking on the *Yes* or *No* button. If the user clicks the *Yes* button, the value of 6 (or *VbYes*) will be returned and assigned to the variable *response*. The next statement will check if the return value is *VbYes*. If it is *VbYes*, then the program executes the *Exit Sub* statement and exits the procedure.



### The InputBox( ) Function

The InputBox( ) function will display a message box where the user can enter a value or a message in the form of text. The format is

**myMessage=InputBox(Prompt, Title, default\_text, x-position, y-position)**

myMessage is a variant data type but typically it is declared as string, which accept the message input by the users. The arguments are explained as follows:

- Prompt - The message displayed normally as a question asked.
- Title - The title of the Input Box.
- default-text - The default text that appears in the input field where users can use it as his intended input or he may change to the message he wish to key in.
- x-position and y-position - the position or the coordinate of the input box.

### Example

```
Private Sub OK_Click()
```

```
Dim userMsg As String
```

```
userMsg = InputBox("What is your message?", "Message Entry Form", "Enter your  
message here", 500, 700)
```

```
If userMsg <> "" Then
```

```
message.Caption = userMsg
```

```
Else
```

```
message.Caption = "No Message"
```

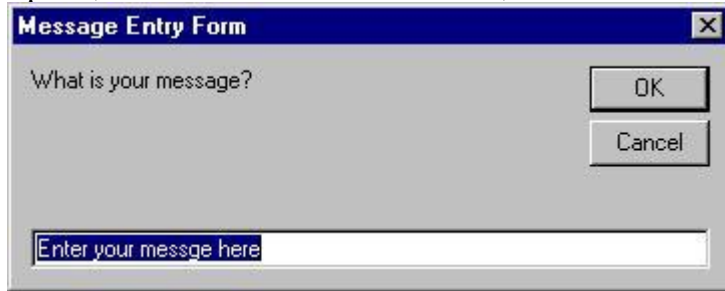
```
End If
```

```
End Sub
```

When a user click the OK button, the input box as shown in Figure 10.5 will appear. After user entering the message and click OK, the message will be displayed on the



caption, if he click Cancel, "No message" will be displayed.



### Controlling Structures

Normally the statements of a program will executed sequentially, one by one from the main ( ) function. Using the control statements can alter the sequence of execution. These statements also can be used to take some decisions, repeating the process number of times etc. To effectively control the VB2008 program flows, if control structure can be used together with the conditional operators and logical operators.

### Decision Making Statements

The conditional control statements are used to check some condition and then transfer the control based on the condition result. There are few control statements available and they are

1. If....Then statement
2. If....Then... Else
3. If....Then....Else If
4. Select Case

### If....Then Statement

This is the simplest control structure which asks the computer to perform a certain action specified by the VB expression if the condition is true. However, when the condition is false, no action will be performed. The general format for the if...then.. Statement is

If condition Then

Statements

End If

### Program to check whether the given number is greater than 50 or not

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

```
Dim number1 As Integer
```

```
number1 = TextBox1.Text  
If number1 > 50 Then  
Msgbox" The number is greater than 50"  
End If  
End Sub  
End Sub
```

This program displays the message box if the condition is satisfied. If the user enters the number smaller than 50 then the user will not receive any output. This indicates that if only the condition is satisfied then the statement will be executed. It is the drawback of simple If statement.

### **If....Then...Else Statement**

If....Then statement that is used above is not very useful in programming and it does not provide choices for the users. In order to provide a choice, If....Then...Else Statement can be used. This control structure will ask the computer to perform a certain action specified by the VB expression if the condition is true. And when the condition is false an alternative action will be executed. The general format for the if...then.. Else statement is

```
If condition Then  
    Statement (1)  
Else  
    Statement (2)  
End If
```

The above program can be rewritten with the use of If Then Else Statement  
Private Sub Button1\_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles Button1.Click

```
Dim number1 As Integer  
number1 = TextBox1.Text  
If number1 > 50 Then  
Msgbox" The number is greater than 50"  
Else  
Msgbox " The number is smaller than 50"  
End If  
End Sub  
End Sub
```

In the previous program, when user inputs the value smaller than 50 then no output will be displayed. But in this program, even if the input is smaller than 50 the message box will be displayed as "The number is smaller than 50". This is possible only with the use If Then else statement.

### **If....Then...Else If Statement**

If there are more than two alternative choices, using just If....Then....Else statement will not be enough. In order to provide more choices, we can use the If....Then...ElseIf Statement. The general format for the if...then.. Else statement is

If condition Then

Statement (1)

ElseIf condition Then

Statement (2)

ElseIf condition Then

Statement (3).

.

.

.

Else

Statement (n)

End If

**This is a program to print the Grade of a student based on the average of marks.**

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

```
Dim Mark As Integer
```

```
Dim Grade as String
```

```
Mark = TextBox1.Text
```

```
If Mark >=80 Then
```

```
Grade="A"
```

```
ElseIf Mark>=60 and Mark<80 then
```

```
Grade="B"
```

```
ElseIf Mark>=40 and Mark<60 then
```

```
Grade="C"
```

```
Else
```

```
Grade="D"
```

```
End If
```

```
End Sub
```

### **Select Case Statement**

Another way to control the program flow is to use the Select Case control structure. The Select Case control structure is slightly different from the If....Else If control structure. The main difference is that the Select Case control structure only make decision on one expression or dimension, while the If ...Else If statement control structure may evaluate only one expression, each If....Else If statement may also compute entirely different dimensions. Select Case is preferred when there exist many different conditions because using If...Then..ElseIf statements might become too messy. The syntax for select case structure is shown below.

Select Case test expression

Case expression list 1

Block of one or more statements

Case expression list 2

Block of one or more Statements

Case expression list 3

Block of one or more statements

Case expression list 4

Case Else

Block of one or more VB Statements

End Select

In the select case statement it is possible to use the IS Keyword along with the comparison operators For Example

**Select Case mark**

**Case Is >= 85**

**Textbox1.Text= "Excellence**

It is possible to use Select Case statement with numbers, as well, and the To word comes in very handy here. If you were checking a variable to see if the number that was in the variable fell within a certain range, you could use something like this:

**Select Case mark**

**Case 0 to 40**

**Textbox1.Text = "Fail"**

Here, a variable called mark is being tested. It's being checked to see if it falls between certain values. If it does, then a message box is displayed.

## Looping Structures

The simple statements that had discussed in the previous section are used to execute the statements only once. If suppose a programmer needs to execute the same statements multiple times. Then some other statement should be used. The solution for this is a looping statement. . For example, to print the string "Visual basic" five times. The same statement should be used five times in the coding. But, imagine if the no. of time increases to print 1000 times or N times. By using the looping statements a statement or set of statements can be executed repeatedly.

Visual Basic 2010 allows a procedure to be repeated as many times as long as the processor and memory could support. This is generally called looping. In Visual Basic 2010, different looping statements are available such as Do Loop, For Loops etc.

### Do Looping Statements

Looping is a very useful feature of Visual Basic because it makes repetitive works easier.

Do Loop statements provide a way to continue iterating while one or more conditions are true? Many types of Do Loops exists in Visual Basic.Net such as

1. Do...While Statement
2. Do...Loop While Statement
3. Do...Until Statement
4. Do...Loop Until

### **Do... While Statement**

The Do loop executes a block of statements either until a condition becomes true or while a condition remains true. The condition can be tested at the beginning or at the end of each iteration. If the test is performed at the end of each iteration, the block of statements is guaranteed to execute at least once. The Do loop can also be written without any conditions, in which case it executes repeatedly until and unless an Exit Do statement is executed within the body of the loop. The general form of Do...While Statement is

Do while condition  
    Block of one or more statements  
Loop

### **Program to demonstrate the use of Do while Loop**

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles Button1.Click  
    Dim counter as Integer  
    Counter =1  
        Do while counter <=100  
            TextBox1.Text=counter  
            Counter +=1  
        Loop  
End Sub
```

This program prints the numbers from 1 to 100 in the textbox. Once it reach 100, the loop terminates.

### **Do...Loop While Statement**

It is another format of Do statement. Here the statements are executed while the condition is satisfied. The format is given below.

Do  
    Block of one or more statements  
Loop While condition

**Program to demonstrate the use of Do Loop While statement. The code that is used in the previous example is rewritten here**

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles Button1.Click  
Counter=1  
    Do  
        TextBox1.Text=counter  
        counter+=1  
    Loop While counter>100  
End Sub
```

### **Do Until Statement**

Another format of Do Loops is Do ... Until statement. There is no much difference between the two looping statements. The format is given below

```
Do Until condition  
    Block of one or more statements  
Loop
```

**Program to demonstrate the use of Do Until statement.**

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles Button1.Click  
Do Until number > 5  
    MsgBox number  
    number = number + 1  
Loop  
  
End Sub
```

The above program displays the numbers from 1 to 5 in the message box when user clicks the button that is placed in the form.

### **Do Loop Until Statement**

The format for this statement is  
Do

```
    Block of one or more statements  
Loop Until condition
```

**Program to demonstrate the use of Do Loop Until statement**

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

```
Do
MsgBox number
number = number + 1
Loop Until number >= 5
End Sub
```

### For Loop Statement

If the programmer wants to execute a block of statements to certain number of times in VB.NET program, then For Loop can be used. The For loop requires the user to provide a top index, a bottom index, and also allows an optional step. The For loop is a powerful and expressive way to iterate through numbers.

### For...Next Statement

The For loop executes a block of statements a specified number of times. The number of iterations is controlled by a loop variable, which is initialized to a certain value by the For statement, then is incremented for each iteration of the loop. The statements in the body of the loop are repeatedly executed until the loop variable exceeds a given upper bound.

```
For variable = expression To expression [ Step expression ]
Statements
Next [ variable_list ]
```

The loop variable can be of any numeric type. The variable is set equal to the value of the first expression before entering the first iteration of the loop body. Prior to executing each iteration of the loop, the loop variable is compared with the value of the second expression. If the value of the loop variable is greater than the expression (or less than the expression if the step expression is negative), the loop exits and execution continues with the first statement following the Next statement

The step expression is a numeric value that is added to the loop variable between loop iterations. If the Step clause is omitted, the step expression is taken to be 1. The Next statement marks the end of the loop body. The Next keyword can either appear by itself in the statement or be followed by the name of the loop variable. If For statements are nested, a single Next statement can terminate the bodies of multiple loops. For example:

Program to print the numbers from 1 to 10 using For Loop

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

```
Dim i as Integer
For i=1 to 10
Msgbox i
```

```
Next  
End Sub
```

This program will print the numbers from 1 to 10 in the message box.

```
Program to print the numbers from 1 to 100 using For Loop and Step  
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles Button1.Click  
Dim i, sum As Integer  
For i=1 to 100 step 10  
sum+=i  
Msgbox sum  
Next  
End Sub
```

The program will calculate the sum of the numbers as follows:  
sum=0+10+20+30+40+.....

### For...Each...Next Statement

The For Each statement is similar to the For statement, except that the loop variable need not be numeric, and successive iterations do not increment the loop variable. Instead, the loop variable takes successive values from a collection of values. Here is the syntax

```
For Each variable In expression  
Statements  
Next [ variable ]
```

The loop variable can be of any type. The expression must be a reference to an object that exposes the IEnumerable interface

```
Program to print the numbers from 1 to 100 using For Loop and Step  
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles Button1.Click  
Dim d As Integer  
For d = 1 To 100 step 10  
Textbox1.Text= d  
Next d  
End Sub
```

### Possible Questions

#### Unit II

1. Explain the different data types with example



2. Explain different operators in Vb.Net with example
3. What is an array? Explain its types
4. Explain message and input box with example
5. Explain different control flow statements by giving example

KAHE

**KARPAGAM ACADEMY OF HIGHER EDUCATION**  
**COIMBATORE - 21**

**DEPARTMENT OF COMPUTER SCIENCE, CA & IT**

**CLASS : III B.Sc COMPUTER TECHNOLOGY**

**BATCH : 2016-2019**

**Part -A Online Examinations**

**I One Marks**

**SUBJECT: . NET Programming**

**SUBJECT CODE: 16CTU502A**

**UNIT-II**

S.No.	Question	Option 1	Option 2	Option 3	Option 4	Answer
1	Toolbar items are part of _____ collection	items	Buttons	propertie	Opions	Buttons
2	A _____ is a place to store the code we write	class	module	method	subroutin	module
3	What is the statement used to declare variable?	loc	dim	global	redim	dim
4	The data type of the variable is defined by using the ----- clause	in	where	as	is	as
5	A composite data type is of ----- types	3	4	5	2	2
6	Constants are declared using the keyword	constant	const	consta	fixed	const
7	An abstract class contains	abstract	non-	friend	overloadi	abstract
8	The storage size for Byte data types is -----	2	4	1	8	1
9	Resizing can be done by using ----- statement	Dim	ReDim	Int	Float	ReDim
10	To ensure that the existing contents of an array are not lost -----	ReDim	Dim	Preserve	New	Preserve
11	The ----- class provided by the .NET framework serves as the base class	Hash	Stack	Queue	Array	Array
12	----- is an array of arrays in which the length of each array can differ	onedimen	rectangul	jagged	sorted	jagged
13	----- loop is used in a situation to execute every single element or item	For	Do	For Each	While	For Each
14	The ----- Function in VB.NET can be used to make the computer emit a	Stack	Beep	Sound	Exit	Beep
15	----- are arrays of controls sharing a common event handler	Arrays	Control	rectangul	jagged	Control
16	The String data type comes from the ----- class	System.S	System	System.F	System.A	System.S
17	_____ is nothing but the name that is used for naming the variables,	Variable	Identifier	Constants	Datatypes	Identifier
18	The ----- function in String Class will insert a String in a specified	Length()	Insert()	Length()	Format()	Insert()
19	----- method is create a new String object with the same content	CopyTo()	Copy()	Format()	Compare	Copy()
20	The ----- function returns an array of String containing the substrings	Length()	Length()	Split()	Format()	Split()
21	The ----- function remeove an item from a specified position	Add	Insert()	Remove	Remove	Remove

22	----- stores a Key Value pair type collection of data	AyyayLis	HashTabl	Stack	Queue	HashTabl
23	What is the maximum no of dimension that an array can have in VB.NET	3	5	32	unlimited	32
24	Which of the following when turned on do not allow to use any variable	Option	Option	Option	Option	Option
25	Which of the following methods can be used to add items to an	Insert	collection	top	Add	Add
26	Parameters to methods in VB.NET are declared by default as -----	ByVal	ByRef	Val	Ref	ByVal
27	Which of the following Access Specifies and scope are used with	Private	Protected	Protected	All	All
28	Which of the following does not denote a arithmetic operator allowed in	Mod	/	*	~	~
29	Which of the following denote the method used for compatible type	TypeCov	Type()	CTyp()	CType()	CType()
30	Which of the following does not denote a data type in VB.Net?	Boolean	Float	Decimal	Byte	Float
31	The ---- event happens when the mouse pointer hovers over the	MouseW	MouseUp	MouseDown	MouseHo	MouseHo
32	----- specifies number of times the mouse button is pressed and released	Button	Click	Delta	X	Click
33	The format used for Date is -----	{0:D}	{0:T}	{0:DD}	{0:Dy}	{0:D}
34	The format used for Time is -----	{0:D}	{0:T}	{0:TT}	{0:TTY}	{0:T}
35	The ---- method Copies a specified number of characters from a specified	CopyTo()	Copy()	Format()	Compare	CopyTo()
36	The ----- method in the VB.NET String Class check the specified	Compare	Exists	Contains	Found	Contains
37	A procedure may return ----- values	0	1	0 or 1	more	0 or 1
38	Procedures that returns a value are called -----	subroutin	sub units	parameter	functions	functions
39	----- scope restricts access to the procedure to only code in the	public	private	protected	Friend	private
40	The first word in the procedure declaration is always the -----	function	Keyword	Keyword	Scope	Scope
41	----- is a key word used to declare a procedure that doesnot return a	Function	Sub	Scope	Private	Sub
42	In the function 'Public Function fname(ByVal str as String) As Integer'	Void	String	Integer	Any	Integer
43	A calling procedure passes data to the parameters by way of -----	objects	argument	strings	numbers	argument
44	To create a procedure as an entry point in code, you must name the	Main	Sub	Entry	Start	Main
45	Which of the following can be called by value?	Class	Module	Assembly	Function	Function
46	Which of the following cannot occur multiple number of times in a	Entry	class	functions	module	Entry
47	Variable of ----- type can store any type of data	Variant	Decimal	Object	Boolean	Object
48	This data type can be used for currency values	Currency	Dollar	Object	Decimal	Decimal
49	Size of integer data type is ----- bits	8	16	24	32	32

50	Which of the given data types used to represent integer numbers	long	short	byte	All the	All
51	----- is the operator used for string concatenation	Cat	Str	^	&	&
52	The order in which the operators in an expression are evaluated is known	operator	operator	associativ	operator	operator
53	And , Or , Not, Xor are called _____ operators	Boolean	Relationa	comparisi	String	Boolean
54	_____ function is used to retrieve only the month part of the date	DateDiff(	DatePart(	DateInter	Date.Mo	DatePart(
55	Which function returns the system's current date and time	DateTim	DateTim	DateTim	DateTim	DateTim
56	In Select Case _____ Case is used to define codes that executes, if the	Default	Otherwis	Else	False	Else
57	While using GoTo, ____ has to be defines to specify the location to jump	Variable	Index	Code	Pointer	Code
58	What statement is used to close a loop started with For statement?	Close	End For	Loop	Next	Next
59	What statement is used to terminate a Do..Loop without evaluating the	End Do	Loop	Exit	Exit Do	Exit Do
60	_____ errors are called Exceptions	compile	build	runtime	None	runtime
61	To halt execution at a specific statement in code, use _____.	Stop	Break	Wait ()	End	Break

### **UNIT III**

Working with WPF: Introduction: Understanding Windows Graphics – WPF: A Higher Level API – The architecture of WPF. XAML: Basics, properties and events in XAML – loading and compiling – Layout. Classic controls: The Control class – content controls – text controls – list controls – Range based controls.

## **INTRODUCTION TO WPF**

When .NET was introduced, it was the powerful language for software development. There was a whole new way to write web applications (ASP.NET), a whole new way to connect to databases (ADO.NET), new type safe languages (C# and VB .NET), and a managed runtime (the CLR). Not least among these new technologies was Windows Forms, a library of classes for building Windows applications. Although Windows Forms is a mature and full-featured toolkit, which haven't changed much in the past ten years. Most significantly, Windows Forms relies on the Windows API to create the visual appearance of standard user interface elements such as buttons, text boxes, check boxes, and so on. As a result, these ingredients are essentially uncustomizable.

For example, if you want to create a stylish glow button you need to create a custom control and paint every aspect of the button (in all its different states) using a lower-level drawing model. Even worse, ordinary windows are carved up into distinct regions, with each control getting its own piece of real estate. As a result, there's no good way for the painting in one control (for example, the glow effect behind a button) to spread into the area owned by another control. And don't even think about introducing animated effects such as spinning text, shimmering buttons, shrinking windows, or live previews because you'll have to paint every detail by hand.

The Windows Presentation Foundation (WPF) changes all this by introducing a new model with entirely different technology. Although WPF includes the standard controls you're familiar with, it draws every text, border, and background fill itself. As a result, WPF can provide much more powerful features that let you alter the way any piece of screen content is rendered. Using these features, you can restyle common controls such as buttons, often without writing any code. The Windows Presentation Foundation (WPF) is an entirely new graphical display system for Windows. WPF is designed for .NET, influenced by modern display technologies such as HTML and Flash, and hardware-accelerated. It's also the most radical change to hit Windows user interfaces since Windows 95.

## **UNDERSTANDING WINDOWS GRAPHICS**

A standard Windows application relies on two parts of the Windows operating system to create its user interface:

- User32 provides the familiar Windows look and feel for elements such as windows, buttons, text boxes, and so on.
- GDI/GDI+ provides drawing support for rendering shapes, text, and images at the cost of additional complexity

In order to overcome certain limitations of User32 and GDI/GDI+, Microsoft Introduced a new graphics engine called DirectX.

## DirectX: The New Graphics Engine

DirectX introduced as an error-prone toolkit for creating games on the Windows platform. Its design mandate was speed, and so Microsoft worked closely with video card vendors to give DirectX the hardware acceleration needed for complex textures, special effects such as partial transparency, and three-dimensional graphics. Over the years since it was first introduced (shortly after Windows 95), DirectX has matured. It's now an integral part of Windows, with support for all modern video cards. Because of its raw complexity, DirectX is almost never used in traditional types of Windows applications such as business software.

In WPF, the underlying graphics technology isn't GDI/GDI+. Instead, it's DirectX. Remarkably, WPF applications use DirectX no matter what type of user interface you create. That means that whether you're designing complex three-dimensional graphics or just drawing buttons and plain text, all the drawing work travels through the DirectX pipeline. As a result, even the most basic business applications can use rich effects from DirectX. You also benefit from hardware acceleration, which simply means DirectX hands off as much work as possible to the GPU (graphics processing unit), which is the dedicated processor on the video card.

## Hardware Acceleration and WPF

The video cards differ in their support for specialized rendering features and optimizations. When programming with DirectX, that's a significant headache. With WPF, it's a much smaller concern, because WPF has the ability to perform everything it does using software calculations rather than relying on built-in support from the video card. Having a high-powered video card is not an absolute guarantee that you'll get fast, hardware accelerated performance in WPF. Software also plays a significant role. For example, WPF can't provide hardware acceleration to video cards that are using out-of-date drivers

WPF also provides better performance under the Windows Vista operating system, where it can take advantage of the new Windows Vista Display Driver Model (WDDM). WDDM offers several important enhancements beyond the Windows XP Display Driver Model (XPDM). Most importantly, WDDM allows several GPU operations to be scheduled at once, and it allows video card memory to be paged to normal system memory if you exceed what's available on the video card. WPF offers some sort of hardware acceleration to all WDDM (Windows Vista) drivers and to XPDM (Windows XP) drivers that were created after November 2004, which is when Microsoft released new driver development guidelines.

## WPF: A HIGHER-LEVEL API

WPF not only offers hardware acceleration through DirectX, but includes a basket of high-level services designed for application programmers. Here's a list with some of the most dramatic changes that WPF made to the Windows programming world:

1. **A web-like layout model.** Rather than fix controls in place with specific coordinates, WPF emphasizes flexible flow layout that arranges controls based on their content. The result is a user interface that can adapt to show highly dynamic content or different languages.

2. **A rich drawing model.** Rather than painting pixels, in WPF you deal with primitives— basic shapes, blocks of text, and other graphical ingredients. You also have new features, such as true transparent controls, the ability to stack multiple layers with different opacities, and native 3-D support.
3. **A rich text model.** After years of substandard text handling with feeble controls such as the classic Label, WPF finally gives Windows applications the ability to display rich, styled text anywhere in a user interface. You can even combine text with lists, floating figures, and other user interface elements. And if you need to display large amounts of text, you can use advanced document display features such as wrapping, columns, and justification to improve readability.
4. **Animation as a first-class programming concept.** Yes, you could use a timer to force a form to repaint itself. But in WPF, animation is built in part of the framework. You define animations with declarative tags, and WPF puts them into action automatically.
5. **Support for audio and video media.** Previous user interface toolkits, such as Windows Forms, were surprisingly limited when dealing with multimedia. But WPF includes support for playing any audio or video file supported by Windows Media Player, and it allows you to play more than one media file at once. Even more impressively, it gives you the tools to integrate video content into the rest of your user interface, allowing you to pull off exotic tricks such as placing a video window on a spinning 3-D cube.
6. **Styles and templates.** Styles allow you to standardize formatting and reuse it throughout your application. Templates allow you to change the way any element is rendered, even a core control such as the button. It's never been easier to build modern skinned interfaces.
7. **Commands.** Most users realize that it doesn't matter whether they trigger the Open command through a menu or a toolbar; the end result is the same. Now that abstraction is available to your code, you can define an application command in one place and link it to multiple controls.
8. **Declarative user interface.** Although you can construct a WPF window with code, Visual Studio takes a different approach. It serializes each window's content to a set of XML tags in a XAML document. The advantage is that your user interface is completely separated from your code, and graphic designers can use professional tools to edit your XAML files and refine your application's front end. (XAML is short for Extensible Application Markup Language).
9. **Page-based applications.** Using WPF, you can build a browser-like application that lets you move through a collection of pages, complete with forward and back navigation buttons. WPF handles the confusing details, such as the page history. You can even deploy your project as a browser-based application that runs right inside Internet Explorer.

### ***Resolution Independence***

Traditional Windows applications are bound by certain assumptions about resolution. Developers usually assume a standard monitor resolution (such as 1024 by 768 pixels), design their windows with that in mind, and try to ensure reasonable resizing behavior for smaller and larger dimensions.

The problem is that the user interface in traditional Windows applications isn't scalable. As a result, if you use a high monitor resolution that crams pixels in more densely, your application windows become smaller and more difficult to read. This is particularly a problem with newer monitors that have high pixel densities and run at correspondingly high resolutions. For example, it's common to find consumer monitors (particularly on laptops) that have pixel densities of 120 dpi or 144 dpi (dots per

inch), rather than the more traditional 96 dpi. At their native resolution, these displays pack the pixels in much more tightly, creating eye-squintingly small controls and text.

### ***WPF Units***

A WPF window and all the elements inside it are measured using device-independent units. A single device-independent unit is defined as 1/96 of an inch. To understand what this means in practice, you'll need to consider an example.

Imagine that you create a small button in WPF that's 96 by 96 units in size. If you're using the standard Windows DPI setting (96 dpi), each device-independent unit corresponds to one real, physical pixel. That's because WPF uses this calculation:

$$[\text{Physical Unit Size}] = [\text{Device-Independent Unit Size}] * [\text{System DPI}]$$

$$= 1/96 \text{ inch} * 96 \text{ dpi}$$

$$= 1 \text{ pixel}$$

Essentially, WPF assumes it takes 96 pixels to make an inch because Windows tells it that through the system DPI setting. However, the reality depends on your display device.

For example, consider a 20-inch LCD monitor with a maximum resolution of 1600 by 1200 pixels. Using a dash of Pythagoras, you can calculate the pixel density for this monitor, as shown here:

$$[\text{Screen DPI}] = \frac{\sqrt{1600^2 + 1200^2} \text{ pixels}}{19 \text{ inches}}$$

$$= 100 \text{ dpi}$$

In this case, the pixel density works out to 100 dpi, which is slightly higher than what Windows assumes. As a result, on this monitor a 96-by-96-pixel button will be slightly smaller than 1 inch.

On the other hand, consider a 15-inch LCD monitor with a resolution of 1024 by 768. Here, the pixel density drops to about 85 dpi, so the 96-by-96 pixel button appears slightly larger than 1 inch.

In both these cases, if you reduce the screen size (say, by switching to 800 by 600 resolutions), the button (and every other screen element) will appear proportionately larger. That's because the system DPI setting remains at 96 dpi. In other words, Windows continues to assume it takes 96 pixels to make an inch, even though at a lower resolution it takes far fewer pixels.

### ***System DPI***

So far, the WPF button example works exactly the same as any other user interface element in any other type of Windows application. The difference is the result if you change the system DPI setting. In the previous generation of Windows, this feature was sometimes called large fonts. That's because the system DPI affects the system font size, but often leaves other details unchanged.



This is where WPF is different. WPF respects the system DPI setting natively and effortlessly. For example, if you change the system DPI setting to 120 dpi (a common choice for users of large high-resolution screens), WPF assumes that it needs 120 pixels to fill an inch of space. WPF uses the following calculation to figure out how it should translate its logical units to physical device pixels:

$$\begin{aligned} \text{[Physical Unit Size]} &= \text{[Device-Independent Unit Size]} \div \text{[System DPI]} \\ &= 1/96 \text{ inch} \div 120 \text{ dpi} \\ &= 1.25 \text{ pixels} \end{aligned}$$

The steps for adjusting the system DPI depend on the operating system. In Windows XP, you follow these steps:

1. Right-click your desktop and choose Display.
2. Choose the Settings tab and click Advanced.
3. On the General tab, choose Normal Size (96 dpi), or Large Size (120 dpi). These are the two recommended options for Windows XP, because custom DPI settings are less likely to be supported by older programs. To try out a custom DPI setting, choose Custom Setting. You can then specify a specific percentage value. (For example, 175% scales the standard 96 dpi to 168 dpi.)

Here's what to do to change system DPI in Windows Vista:

1. Right-click your desktop and choose Personalize.
2. In the list of links on the left, choose Adjust Font Size (DPI).
3. Choose between 96 or 120 dpi. Or click Custom DPI to use a custom DPI setting. You can then specify a percentage value, as shown in Figure 1-1. (For example, 175% scales the standard 96 dpi to 168 dpi.) In addition, when using a custom DPI setting, you have an option named Use Windows XP Style DPI Scaling, which is described in the sidebar "DPI Scaling with Windows Vista."

### ***Bitmap and Vector Graphics***

When you work with ordinary controls, you can take WPF's resolution independence for granted. WPF takes care of making sure that everything has the right size automatically. However, if you plan to incorporate images into your application you can't be quite as casual. For example, in traditional Windows applications, developers use tiny bitmaps for toolbar commands. In a WPF application, this approach is not ideal because the bitmap may display artifacts (becoming blurry) as it's scaled up or down according to the system DPI. Instead, when designing a WPF user interface even the smallest icon is generally implemented as a vector graphic. Vector graphics are defined as a set of shapes, and as such they can be easily scaled to any size.

### **The Evolution of WPF**

Although WPF is a relatively new technology, it already exists in two versions:

- **WPF 3.0.** The first version of WPF was released with two other new technologies: *Windows Communication Foundation (WCF)* and *Windows Workflow Foundation (WF)*. Together, these three technologies were called the .NET Framework 3.0.
- **WPF 3.5.** A year later, a new version of WPF was released as part of the .NET Framework 3.5. The new features in WPF are mostly minor refinements. Some of these bug fixes and performance improvements are available to .NET Framework 3.0 applications through the .NET Framework 3.0 Service Pack 1.

From a developer standpoint, the most significant difference between WPF 3.0 and 3.5 is design-time support. The .NET Framework 3.0 was released without a corresponding version of Visual Studio.

Developers could get basic support for Visual Studio 2005 by installing a free Community Technology Preview (CTP). Although these extensions made it possible to create and develop WPF applications in Visual Studio 2005, they didn't provide a drag-and-drop designer for WPF windows.

The .NET Framework 3.5 was released in conjunction with Visual Studio 2008, and as a result, it offers much better design-time support for building WPF applications.

### ***New Features in WPF 3.5***

Aside from bug fixes, performance tune-ups, and better design support, WPF 3.5 introduces the following enhancements.

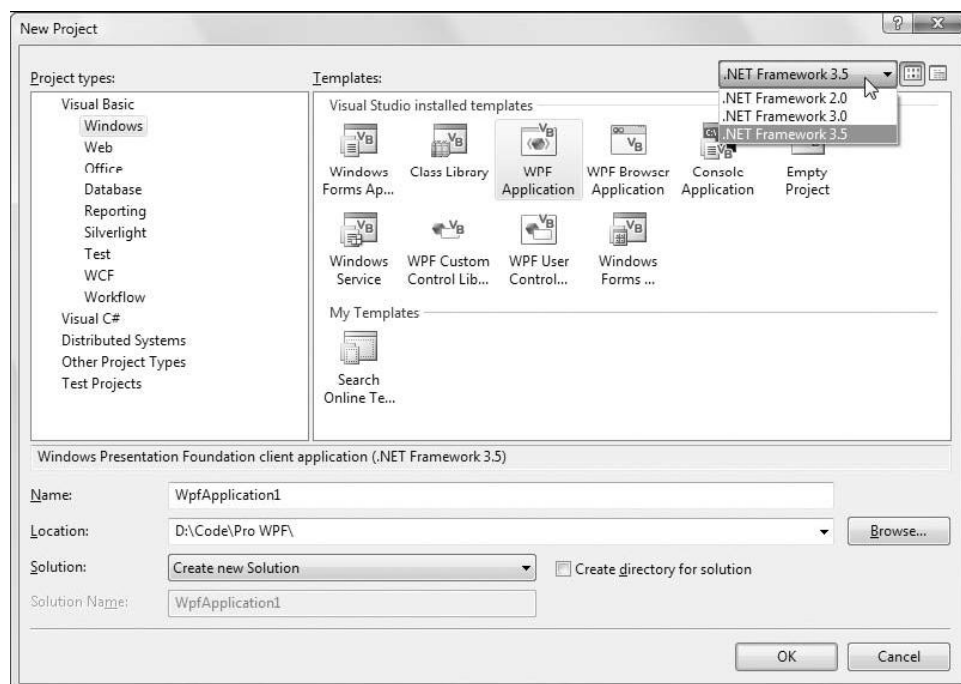
- **Firefox support for XBAPs.** It's now possible to run WPF browser-hosted applications (known as XBAPs) in Firefox as well as in Internet Explorer.
- **Data binding support for LINQ.** LINQ is a set of language extensions that allow developers to write queries. These queries can pull data out of various data sources, including in-memory collections, XML files, and databases, all without requiring a line of low-level code.
- **Data binding support for IDataErrorInfo.** The IDataErrorInfo interface is a key for business developers who want to build rich data objects with built-in validation. Now, the data binding infrastructure can catch these validation errors and display them in the user interface.
- **Support for placing interactive controls (such as buttons) inside a RichTextBox control.** This feature previously required an obscure workaround. It now works through a simple property
- **Support for placing 2-D elements on 3-D surfaces.** This feature previously required a separate download. Now, it's incorporated into the framework, along with better support for 3-D objects that can raise mouse and keyboard events.
- **An add-in model.** The add-in model allows an application to host third-party components in a limited security context. Technically, this feature isn't WPF-specific, because it can be used in any .NET 3.5 applications.

### ***Multitargeting***

Previous versions of Visual Studio were tightly coupled to specific versions of .NET. You used Visual Studio .NET to create .NET 1.0 applications, Visual Studio .NET 2003 to create .NET 1.1 applications, and Visual Studio 2005 to create .NET 2.0 applications. Visual Studio 2008 partially removes this restriction. It allows you to create applications that are specifically designed to work with .NET 2.0, .NET 3.0, or .NET 3.5.

Although it's obviously not possible to create a WPF application with .NET 2.0, both .NET 3.0 and .NET 3.5 have WPF support. You may choose to target .NET 3.0 for slightly broader compatibility (because .NET 3.0 applications can run on both the .NET 3.0 and .NET 3.5 runtimes). Or, you may choose to target .NET 3.5 to get access to newer features in WPF or in the .NET platform itself. (One common reason for targeting .NET 3.5 is to support LINQ, the set of technologies that allow .NET languages to access different data sources using tightly integrated query syntax.)

When you create a new project in Visual Studio (by choosing File → New → Project), you can choose the version of the .NET Framework that you're targeting from a drop-down list in the top-right corner of the New Project dialog box (see Figure 1-2). You can also change the version you're targeting at any point afterward. Just double-click the My Project node in the Solution Explorer, choose the Compile tab, click the Advanced Compile Options button, and change the selection in the Target Framework list.



To really understand how the Visual Studio multitargeting system works, you need to know a bit more about how .NET 3.5 is structured. Essentially, .NET 3.5 is built out of three separate pieces—a copy of the original .NET 2.0 assemblies, a copy of the assemblies that were added in .NET 3.0 (for WPF, WCF, and WF), and the new assemblies that were added in .NET 3.5 (for LINQ and a number of miscellaneous features). However, when you create and test an application in Visual Studio, you are always using the .NET 3.5 assemblies. When you choose to target an earlier version of .NET, Visual Studio simply uses a subset of the .NET 3.5 assemblies.

For example, when you choose to target .NET 3.0, you effectively configure Visual Studio to use a *portion* of .NET 3.5—just those assemblies that were available in .NET 2.0 and .NET 3.0. There's a potential stumbling block in this system. Although these assemblies are treated as though they haven't changed in .NET 3.5, they aren't completely identical to the .NET 2.0 versions. For example, they may include performance tweaks, bug fixes, and (very rarely) a new public member in a class. For that reason, if you build an assembly that targets an earlier version of .NET, you should still test it with that version of .NET to make absolutely sure there are no backward compatibility quirks.

### ***Windows Forms Lives On***

WPF is the platform for the future of Windows user interface development. However, it won't displace Windows Forms overnight. Windows Forms is in many ways the culmination of the display technology built on GDI/GDI+ and User32. It's more mature than WPF and still includes features that haven't made their way into the WPF toolkit (such as the WebBrowser control, the DataGridView control, and the HelpProvider component).

So which platform should you choose when you begin designing a new Windows application? If you're starting from the ground up, WPF is an ideal choice and it offers the best prospects for future enhancements and longevity. Similarly, if you need one of the features that WPF provides and Windows Forms does not—such as 3-D drawing or page-based applications—it makes sense to make the shift. On

the other hand, if you have a considerable investment in a Windows Forms–based business application, there’s no need to recode your application for WPF. The Windows Forms platform will continue to be supported for years to come.

Perhaps the best part of the story is the fact that Microsoft has invested considerable effort in building an interoperability layer between WPF and Windows Forms (which plays a similar role to the interoperability layer that allows .NET applications to continue to use legacy COM components). In Chapter 25, you’ll learn how to use this support to host Windows Forms controls inside a WPF application, and vice versa. WPF offers similarly robust support for integrating with older Win32-style applications.

### ***DirectX Also Lives On***

There’s one area where WPF isn’t a good fit: when creating applications with demanding realtime graphics, such as complex physics-based simulators or cutting-edge action games. If you want the best possible video performance for these types of applications, you’ll need to program at a much lower level and use raw DirectX. You can download the managed .NET libraries for DirectX programming at <http://msdn.microsoft.com/directx>.

### ***Silverlight***

Like the .NET Framework itself, WPF is a *Windows-centric technology*. That means that WPF applications can only be used on computers running the Windows operating system (specifically, Windows XP or Windows Vista). Browser-based WPF applications are similarly limited—they can run only on Windows computers, although they support both the Internet Explorer and Firefox browsers.

These restrictions won’t change—after all, part of Microsoft’s goal with WPF is to take advantage of the rich capabilities of Windows computers and its investment in technologies such as DirectX. However, there is a separate technology named Silverlight that’s designed to take a subset of the WPF platform, host it in any modern browser using a plug-in (including Firefox, Opera, and Safari), and open it up to other operating systems (such as Linux and Mac OS). This is an ambitious project that’s attracted considerable developer interest.

To make matters more interesting, Silverlight currently exists in two versions:

- **Silverlight 1.0** This first release includes 2-D drawing features, animation, and media playback features that are similar to those in WPF. However, Silverlight 1.0 has no support for the .NET Framework or the C# and Visual Basic languages—instead, you must use JavaScript code.
- **Silverlight 2.0** This second release adds a pared-down version of the .NET Framework, complete with a miniature CLR that’s hosted by the browser plug-in and a small subset of essential .NET Framework classes. Because Silverlight 2.0 allows you to write code in a .NET language such as C# and Visual Basic, it’s a far more compelling technology than Silverlight 1.0. However, at the time of this writing it’s still in beta.

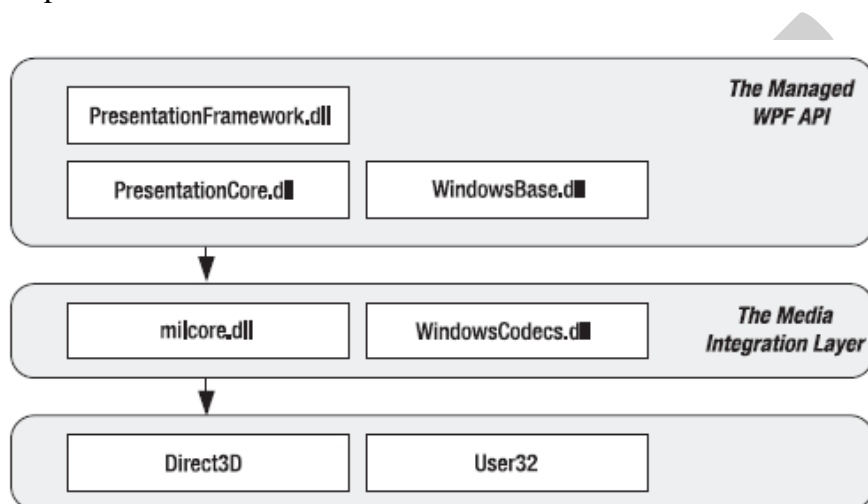
Although both Silverlight 1.0 and Silverlight 2.0 are based on WPF and incorporate many of its conventions (such as the XAML markup you’ll learn about in the next chapter), they leave out certain feature areas. For example, neither version supports true three-dimensional drawing or rich document display. New features may appear in future Silverlight releases, but the more complex ones might never make the leap.

The ultimate goal of Silverlight is to provide a powerful developer-oriented competitor for Adobe Flash. However, Flash has a key advantage—it’s used throughout the Web, and the Flash plug-in is

installed just about everywhere. In order to entice developers to switch to a new, less-established technology, Microsoft will need to make sure Silverlight has next-generation features, rock-solid compatibility, and unrivaled design support.

### THE ARCHITECTURE OF WPF

WPF uses a multilayered architecture. At the top, your application interacts with a high-level set of services that are completely written in managed C# code. The actual work of translating .NET objects into Direct3D textures and triangles happens behind the scenes, using a lowerlevel unmanaged component called milcore.dll.



**Architecture of WPF**

- **PresentationFramework.dll** holds the top-level WPF types, including those that represent windows, panels, and other types of controls. It also implements higher-level programming abstractions such as styles. Most of the classes are used directly come from this assembly.
- **PresentationCore.dll** holds base types, such as `UIElement` and `Visual`, from which all shapes and controls derive
- **WindowsBase.dll** holds even more basic ingredients that have the potential to be reused outside of WPF, such as `DispatcherObject` and `DependencyObject`, which introduces the plumbing for dependency properties
- **milcore.dll** is the core of the WPF rendering system and the foundation of the Media Integration Layer (MIL). Its composition engine translates visual elements into the triangle and textures that Direct3D expects. Although milcore.dll is considered a part of WPF, it's also an essential system component for Windows Vista. In fact, the Desktop Window Manager (DWM) in Windows Vista uses milcore.dll to render the desktop.
- **WindowsCodecs.dll** is a low-level API that provides imaging support (for example, processing, displaying, and scaling bitmaps and JPEGs).
- **Direct3D** is the low-level API through which all the graphics in a WPF are rendered.
- **User32** is used to determine what program gets what real estate. As a result, it's still involved in WPF, but it plays no part in rendering common controls.



## **XAML**

XAML (short for Extensible Application Markup Language, and pronounced “zammel”) is a markup language used to instantiate .NET objects. Although XAML is a technology that can be applied to many different problem domains, its primary role in life is to construct WPF user interfaces. In other words, XAML documents define the arrangement of panels, buttons, and controls that make up the windows in a WPF application.

With traditional display technologies, there’s no easy way to separate the graphical content from the code. The key problem with Windows Forms application is that every form you create is defined entirely in VB code. As you drop controls onto the design surface and configure them, Visual Studio quietly adjusts the code in the corresponding form class. Sadly, graphic designers don’t have any tools that can work with VB code.

## **UNDERSTANDING XAML**

Developers realized long ago that the most efficient way to tackle complex, graphically rich applications is to separate the graphical portion from the underlying code. That way, artists can own the graphics and developers can own the code. Both pieces can be designed and refined separately, without any versioning headaches.

### **Graphical User Interfaces before WPF**

With traditional display technologies, there’s no easy way to separate the graphical content from the code. The key problem with Windows Forms application is that every form you create is defined entirely in VB code. As you drop controls onto the design surface and configure them, Visual Studio quietly adjusts the code in the corresponding form class. Sadly, graphic designers don’t have any tools that can work with VB code. Instead, artists are forced to take their content and export it to a bitmap format. These bitmaps can then be used to skin windows, buttons, and other controls. This approach works well for straightforward interfaces that don’t change much over time, but it’s extremely limiting in other scenarios. Some of its problems include the following:

- Each graphical element (background, button, and so on) needs to be exported as a separate bitmap. That limits the ability to combine bitmaps and use dynamic effects such as antialiasing, transparency, and shadows.
- A fair bit of user interface logic needs to be embedded in the code by the developer. This includes button sizes, positioning, mouse-over effects, and animations. The graphic designer can’t control any of these details.
- There’s no intrinsic connection between the different graphical elements, so it’s easy to end up with an unmatched set of images. Tracking all these items adds complexity.
- Bitmaps can’t be resized without compromising their quality. For that reason, a bitmap-based user interface is resolution-dependent. That means it can’t accommodate large monitors and high-resolution displays, which is a major violation of the WPF design philosophy.

## **XAML BASICS**

The XAML standard is quite straightforward once you understand a few ground rules:

- Every element in a XAML document maps to an instance of a .NET class. The name of the element matches the name of the class exactly. For example, the element `<Button>` instructs WPF to create a Button object.
- As with any XML document, you can nest one element inside another. As you'll see, XAML gives every class the flexibility to decide how it handles this situation. However, nesting is usually a way to express containment—in other words, if you find a Button element inside a Grid element, your user interface probably includes a Grid that contains a Button inside.
- You can set the properties of each class through attributes. However, in some situations an attribute isn't powerful enough to handle the job. In these cases, you'll use nested tags with a special syntax

Before continuing, take a look at this bare-bones XAML document, which represents anew blank window (as created by Visual Studio). The lines have been numbered for easy reference:

```
1 <Window x:Class="Window1"
2 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4 Title="Window1" Height="300" Width="300">
5
6 <Grid>
7 </Grid>
8 </Window>
```

This document includes only two elements—the top-level Window element, which represents the entire window, and the Grid, in which you can place all your controls. Although you could use any top-level element, WPF applications rely on just a few:

- Window
- Page (which is similar to Window, but used for navigable applications)
- Application (which defines application resources and startup settings)

As in all XML documents, there can only be one top-level element. In the previous example, that means that as soon as you close the Window element with the `</Window>` tag, you end the document. No more content can follow.

Looking at the start tag for the Window element you'll find several interesting attributes, including a class name and two XML namespaces (described in the following sections). You'll also find the three properties shown here:

```
4 Title="Window1" Height="300" Width="300">
```

Each attribute corresponds to a separate property of the Window class. All in all, this tells WPF to create a window with the caption Window1 and to make it 300 by 300 units large.

## **XAML Namespaces**

Clearly, it's not enough to supply just a class name. The XAML parser also needs to know the .NET namespace where this class is located. For example, the Window class could exist in several places—it might refer to the `system.Windows.Window` class, or it could refer to a Window class in a third-party component, or one you've defined in your application. To figure out which class you really want, the XAML parser examines the XML namespace that's applied to the element. Here's how it works. In the sample document shown earlier, two namespaces are defined:

```
2 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
```

```
3 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

The `xmlns` attribute is a specialized attribute in the world of XML that's reserved for declaring namespaces. This snippet of markup declares two namespaces that you'll find in every WPF XAML document you create:

- `http://schemas.microsoft.com/winfx/2006/xaml/presentation` is the core WPF namespace. It encompasses all the WPF classes, including the controls you use to build user interfaces. In this example, this namespace is declared without a namespace prefix, so it becomes the default namespace for the entire document. In other words, every element is automatically placed in this namespace unless you specify otherwise.
- `http://schemas.microsoft.com/winfx/2006/xaml` is the XAML namespace. It includes various XAML utility features that allow you to influence how your document is interpreted. This namespace is mapped to the prefix `x`. That means you can apply it by placing the namespace prefix before the element name (as in `<x:ElementName>`).

As you can see, the XML namespace name doesn't match any particular .NET namespace. There are a couple of reasons the creators of XAML chose this design. By convention, XML namespaces are often URIs (as they are here). These URIs look like they point to a location on the Web, but they don't. The URI format is used because it makes it unlikely that different organizations will inadvertently create different XML-based languages with the same namespace. Because the domain `schemas.microsoft.com` is owned by Microsoft, only Microsoft will use it in an XML namespace name.

The other reason that there isn't a one-to-one mapping between the XML namespaces used in XAML and .NET namespaces is because it would significantly complicate your XAML documents. The problem here is that WPF encompasses well over a dozen namespaces (all of which start with `System.Windows`). If each .NET namespace had a different XML namespace, you'd need to specify the right namespace for each and every control you use, which quickly gets messy. Instead, the creators of WPF chose to combine all of these .NET namespaces into a single XML namespace. This works because within the different .NET namespaces that are a part of WPF, there aren't any classes that have the same name. The namespace information allows the XAML parser to find the right class. For example, when it looks at the Window and Grid elements, it sees that they are placed in the default WPF namespace. It then searches the corresponding .NET namespaces, until it finds `System.Windows`. Window and `system.Windows.Controls.Grid`.



### **The Code-Behind Class**

XAML allows you to construct a user interface, but in order to make a functioning application you need a way to connect the event handlers that contain your application code. XAML makes this easy using the Class attribute that's shown here:

```
<Window x:Class="Window1"
```

The x namespace prefix places the Class attribute in the XAML namespace, which means this is a more general part of the XAML language. In fact, the Class attribute tells the XAML parser to generate a new class with the specified name. That class derives from the class that's named by the XML element. In other words, this example creates a new class named Window1, which derives from the base Window class. The Window1 class is generated automatically at compile time. But here's where things get interesting. You can supply a piece of the Window1 class that will be merged into the automatically generated portion. The piece you specify is the perfect container for your event handling code.

When you compile your application, the XAML that defines your user interface (such as Window1.xaml) is translated into CLR type declaration that is merged with the logic in your code-behind class file (such as Window1.xaml.vb) to form one single unit. Initially, the code behind class that Visual Studio creates is empty:

```
Class Window1  
End Class
```

### **The InitializeComponent() Method**

Currently, the Window1 class code doesn't include any code. However, there's one detail that's completely hidden—the default constructor. If you could see the constructor that Visual Studio generates at compile time, it would look like this:

```
Public Sub New()  
InitializeComponent()  
End Sub
```

The InitializeComponent() method is another piece of automatically generated code that doesn't appear in your class. It loads the BAML (the compiled XAML) from your assembly and uses it to build your user interface. As it parses the BAML, it creates each control object, sets its properties, and attaches any event handlers. At this point, you're probably wondering why it's important to understand a detail that Visual Studio hides from you completely. The reason becomes apparent if you create a custom constructor for your window. In this case, you need to explicitly call the InitializeComponent() method. If you don't, the window won't be initialized, and none of your controls will appear.

### **Naming Elements**

There's one more detail to consider. In your code-behind class, you'll often want to manipulate controls programmatically. For example, you might want to read or change properties or attach and detach event handlers on the fly. To make this possible, the control must include a XAML Name attribute. In the previous example, the Grid control does not include a Name attribute, so you won't be able to manipulate it in your code-behind file. Here's how you can attach a name to the Grid:

```
6 <Grid x:Name="grid1">
```

7 </Grid>

You can make this change by hand in the XAML document, or you can select the grid in the Visual Studio designer and set the Name property using the Properties window. Either way, the Name attribute tells the XAML parser to add a field like this to the automatically generated portion of the Window1 class:

```
Friend WithEvents grid1 As System.Windows.Controls.Grid
```

Now you can interact with the grid in your Window1 class code by using the name grid1:

```
MessageBox.Show(String.Format("The grid is {0}x{1} units in size.", _  
grid1.ActualWidth, grid1.ActualHeight))
```

This technique doesn't add much for the simple grid example, but it becomes much more important when you need to read values in input controls such as text boxes and list boxes. The Name property shown previously is part of the XAML language, and it's used to help integrate your code-behind class. Somewhat confusingly, many classes define their own Name property. (One example is the base FrameworkElement class from which all WPF elements derive.) XAML parsers have a clever way of handling this. You can set either the XAML Name property (using the x: prefix) or the Name property that belongs to the actual element (by leaving out the prefix). Either way, the result is the same—the name you specify is used in the automatically generated code file and it's used to set the Name property. That means the following markup is equivalent to what you've already seen:

```
<Grid Name="grid1">  
</Grid>
```

This bit of magic only works if the class that includes the Name property decorates itself with the RuntimeNameProperty attribute. The RuntimeNameProperty indicates which property should be treated as the name for instances of that type. (Obviously, it's usually the property that's named Name.) The FrameworkElement class includes the RuntimeNameProperty attribute, so there's no problem.

## **PROPERTIES AND EVENTS IN XAML**

XAML isn't limited to the classes that are a part of WPF. You can use XAML to create an instance of any class that meets a few ground rules

### **Simple Properties and Type Converters**

As you've already seen, the attributes of an element set the properties of the corresponding object. For example, the text boxes in the eight ball example configure the alignment, margin, and font:

```
<TextBox Name="txtQuestion"  
VerticalAlignment="Stretch" HorizontalAlignment="Stretch"  
FontFamily="Verdana" FontSize="24" Foreground="Green" ... >
```

In order for this to work, the System.Windows.Controls.TextBox class must provide the following properties: VerticalAlignment, HorizontalAlignment, FontFamily, FontSize, and Foreground. To make this system work, the XAML parser needs to perform a bit more work than you might initially realize. The value in an XML attribute is always a plain text string. However, object properties can be any .NET

type. In the previous example, there are two properties that use enumerations (VerticalAlignment and HorizontalAlignment), one string (FontFamily), one integer (FontSize), and one Brush object (Foreground). In order to bridge the gap between string values and nonstring properties, the XAML parser needs to perform a conversion. The conversion is performed by type converters, a basic piece of .NET infrastructure that's existed since .NET 1.0.

Essentially, a type converted has one role in life—it provides utility methods that can convert a specific .NET data type to and from any other .NET type, such as a string representation in this case. The XAML parser follows two steps to find a type converter:

- It examines the property declaration, looking for a TypeConverter attribute. (If present, the TypeConverter attribute indicates what class can perform the conversion.) For example, when you use a property such as Foreground, .NET checks the declaration of the Foreground property.
- If there's no TypeConverter attribute on the property declaration, the XAML parser checks the class declaration of the corresponding data type. For example, the Foreground property uses a Brush object. The Brush class (and its derivatives) use the BrushConverter because the Brush class is decorated with the TypeConverter(Get- Type(BrushConverter)) attribute declaration.

If there's no associated type converter on the property declaration or the class declaration, the XAML parser generates an error. This system is simple but flexible. If you set a type converter at the class level, that converter applies to every property that uses that class. On the other hand, if you want to fine-tune the way type conversion works for a particular property, you can use the TypeConverter attribute on the property declaration instead. It's technically possible to use type converters in code, but the syntax is a bit convoluted. It's almost always better to set a property directly—not only is it faster, it also avoids potential errors from mistyping strings, which won't be caught until runtime. (This problem doesn't affect XAML, because the XAML is parsed and validated at compile time.) Of course, before you can set the properties on a WPF element, you need to know a bit more about the basic WPF properties and data types.

### **Complex Properties**

As handy as type converters are, they aren't practical for all scenarios. For example, some properties are full-fledged objects with their own set of properties. Although it's possible to create a string representation that the type converter could use, that syntax might be difficult to use and prone to error. Fortunately, XAML provides another option: property-element syntax. With propertyelement syntax, you add a child element with a name in the form Parent.PropertyName. For example, the Grid has a Background property that allows you to supply a brush that's used to paint the area behind the controls. If you want to use a complex brush—one more advanced than a solid color fill—you'll need to add a child tag named Grid.Background, as shown here: <Grid Name="grid1"> <Grid.Background>

Any set of XAML tags can be replaced with a set of code statements that performs the same task. The tags shown previously, which fill the background with a gradient of your choice, are equivalent to the following code:

```
Dim brush As New LinearGradientBrush()  
Dim gradientStop1 As New GradientStop()
```

```
gradientStop1.Offset = 0
gradientStop1.Color = Colors.Red
brush.GradientStops.Add(gradientStop1)
Dim gradientStop2 As New GradientStop()
gradientStop2.Offset = 0.5
gradientStop2.Color = Colors.Indigo
brush.GradientStops.Add(gradientStop2)
Dim gradientStop3 As New GradientStop()
gradientStop3.Offset = 1
gradientStop3.Color = Colors.Violet
brush.GradientStops.Add(gradientStop3)
grid1.Background = brush
```

### Attached Properties

Along with ordinary properties, XAML also includes the concept of attached properties—properties that may apply to several controls but are defined in a different class. In WPF, attached properties are frequently used to control layout.

Here's how it works. Every control has its own set of intrinsic properties. (For example, a text box has a specific font, text color, and text content as dictated by properties such as Font-Family, Foreground, and Text.) When you place a control inside a container it gains additional features, depending on the type of container. (For example, if you place a text box inside a grid, you need to be able to choose the grid cell where it's positioned.) These additional details are set using attached properties.

Attached properties always use a two-part name in this form: DefiningType.Property- Name. This two-part naming syntax allows the XAML parser to distinguish between a normal property and an attached property. In the eight ball example, attached properties allow the individual controls to place themselves on separate rows in the (invisible) grid:

```
<TextBox ... Grid.Row="0">
[Place question here.]
</TextBox>
<Button ... Grid.Row="1">
Ask the Eight Ball
</Button>
<TextBox ... Grid.Row="2">
[Answer will appear here.]
</TextBox>
```

Attached properties aren't really properties at all. They're actually translated into method calls. The XAML parser calls the shared method that has this form: DefiningType.SetPropertyName(). For example, in the previous XAML snippet, the defining type is the Grid class, and the property is Row, so the parser calls Grid.SetRow(). When calling SetPropertyNames(), the parser passes two parameters: the object that's being modified, and the property value that's specified. For example, when you set the Grid.Row property on the TextBox control, the XAML parser executes this code:

```
Grid.SetRow(txtQuestion, 0)
```

This pattern (calling a shared method of the defining type) is a convenience that conceals what's really taking place. To the casual eye, this code implies that the row number is stored in the Grid object. However, the row number is actually stored in the object that it applies to—in this case, the TextBox object. This sleight of hand works because the TextBox derives from the DependencyObject base class, as do all WPF controls. And as you'll learn in Chapter 6, the DependencyObject is designed to store a virtually unlimited collection of dependency properties. (The attached properties that were discussed earlier are a special type of dependency property.) In fact, the Grid.SetRow() method is actually a shortcut that's equivalent to calling DependencyObject.SetValue() method, as shown here:

```
xtQuestion.SetValue(Grid.RowProperty, 0)
```

Attached properties are a core ingredient of WPF. They act as an all-purpose extensibility system. For example, by defining the Row property as an attached property, you guarantee that it's usable with any control. The other option, making it a part of a base class such as FrameworkElement, complicates life. Not only would it clutter the public interface with properties that only have meaning in certain circumstances (in this case, when an element is being used inside a Grid), it also makes it impossible to add new types of containers that require new properties.

### **Nesting Elements**

XAML documents are arranged as a heavily nested tree of elements. In the current example, a Window element contains a Grid element, which contains TextBox and Button elements. XAML allows each element to decide how it deals with nested elements. This interaction is mediated through one of four mechanisms that are evaluated in this order:

- If the parent implements IList, the parser calls IList.Add() and passes in the child.
- If the parent implements IDictionary, the parser calls IDictionary.Add() and passes in the child. When using a dictionary collection, you must also set the x:Key attribute to give a key name to each item.
- If the parent is decorated with the ContentProperty attribute, the parser uses the child to set that property.

### **Special Characters and White space**

XAML is bound by the rules of XML. For example, XML pays special attention to a few specific characters, such as & and < and >. If you try to use these values to set the content of an element, you'll run into trouble because the XAML parser assumes you're trying to do something else—such as create a nested element.

For example, imagine you want to create a button that contains the text <Click Me>. The following markup won't work:

```
<Button ... >  
<Click Me>  
</Button>
```

The problem here is that it looks like you're trying to create an element named Click with an attribute named Me. The solution is to replace the offending characters with entity references—specific codes that the XAML parser will interpret correctly. Table 2-1 lists the character entities you might choose to use.



Note that the quotation mark character entity is only required when setting values using an attribute because the quotation mark indicates the beginning and ending of an attribute value.

Special Character	Character Entity
Less than (<)	&lt;
Greater than (>)	&gt;
Ampersand (&)	&amp;
Quotation mark (")	&quot;

## LAYOUT

In WPF, shape layout use different *containers*. Each container has its own layout logic—some stack elements, others arrange them in a grid of invisible cells.

### UNDERSTANDING LAYOUT IN WPF

WPF introduces a new layout system, developers can now create resolution-independent, size-independent interfaces that scale well on different monitors, adjust themselves when content changes, and handle the transition to other languages effortlessly.

A WPF window can hold only a single element. To fit in more than one element and create a more practical user interface, you need to place a container in your window and then add other elements to that container.

In WPF, layout is determined by the container that you use. Although there are several containers to choose from, the “ideal” WPF window follows a few key principles:

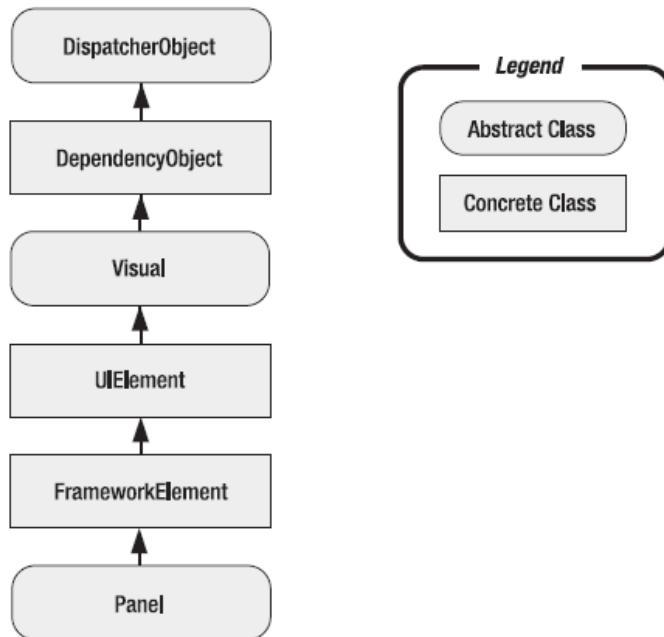
- **Elements (like controls) should not be explicitly sized.** Instead, they grow to fit their content. For example, a button expands as you add more text. You can limit controls to acceptable sizes by setting a maximum and minimum size.
- **Elements do not indicate their position with screen coordinates.** Instead, they are arranged by their container based on their size, order, and (optionally) other information that’s specific to the layout container. If you need to add whitespace between elements, you use the `Margin` property.
- **Layout containers “share” the available space among their children.** They attempt to give each element its preferred size (based on its content) if the space is available. They can also distribute extra space to one or more children.
- **Layout containers can be nested.** A typical user interface begins with the `Grid`, WPF’s most capable container, and contains other layout containers that arrange smaller groups of elements, such as captioned text boxes, items in a list, icons on a toolbar, a column of buttons, and so on.

### The Layout Process

WPF layout takes place in two stages: a *measure* stage and an *arrange* stage. In the measure stage, the container loops through its child elements and asks them to provide their preferred size. In the arrange stage, the container places the child elements in the appropriate position.

### The Layout Containers

All the WPF layout containers are panels that derive from the abstract `System.Windows.Controls.Panel` class. The `Panel` class adds a small set of members, including the three public properties.



### Core Layout Panels

Name	Description
StackPanel	Places elements in a horizontal or vertical stack. This layout container is typically used for small sections of a larger, more complex window.
WrapPanel	Places elements in a series of wrapped lines. In horizontal orientation, the WrapPanel lays items out in a row from left to right and then onto subsequent lines. In vertical orientation, the WrapPanel lays out items in a top-to-bottom column and then uses additional columns to fit the remaining items.
DockPanel	Aligns elements against an entire edge of the container.
Grid	Arranges elements in rows and columns according to an invisible table. This is one of the most flexible and commonly used layout containers.
UniformGrid	Places elements in an invisible table but forces all cells to have the same size. This layout container is used infrequently.
Canvas	Allows elements to be positioned absolutely using fixed coordinates. This layout container is the most similar to traditional Windows Forms, but it doesn't provide anchoring or docking features. As a result, it's an unsuitable choice for a resizable window unless you're willing to do a fair bit of work.

### SIMPLE LAYOUT WITH THE STACKPANEL

The StackPanel is one of the simplest layout containers. It simply stacks its children in a single row or column.

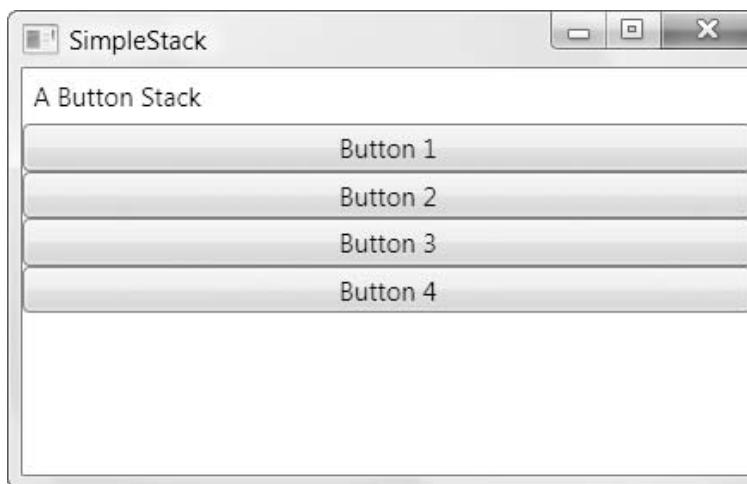
For example, consider this window, which contains a stack of three buttons:

```

<Window x:Class="SimpleStack"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  
```

Title="Layout" Height="223" Width="354" >

```
<StackPanel>
  <Label>A Button Stack</Label>
  <Button>Button 1</Button>
  <Button>Button 2</Button>
  <Button>Button 3</Button>
  <Button>Button 4</Button>
</StackPanel>
</Window>
```



By default, a StackPanel arranges elements from top to bottom, making each one as tall as necessary to display its content. The StackPanel can also be used to arrange elements horizontally by the Orientation property:

```
<StackPanel Orientation="Horizontal">
```

is

setting

## Layout Properties

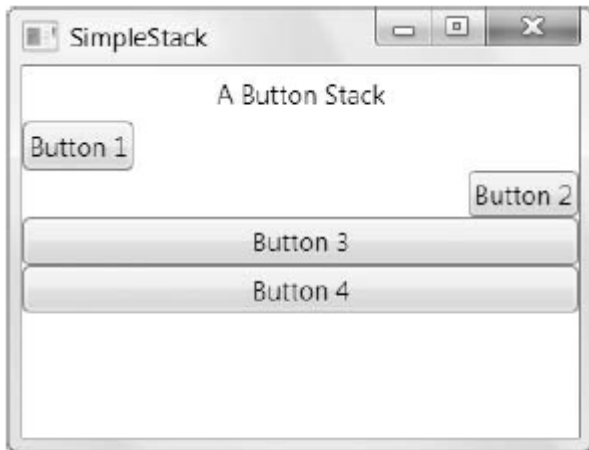
Name	Description
HorizontalAlignment	Determines how a child is positioned inside a layout container when there's extra horizontal space available. You can choose Center, Left, Right, or Stretch.
VerticalAlignment	Determines how a child is positioned inside a layout container when there's extra vertical space available. You can choose Center, Top, Bottom, or Stretch.
Margin	Adds a bit of breathing room around an element. The Margin property is an instance of the System.Windows.Thickness structure, with separate components for the top, bottom, left, and right edges.
MinWidth and MinHeight	Sets the minimum dimensions of an element. If an element is too large for its layout container, it will be cropped to fit.
MaxWidth and MaxHeight	Sets the maximum dimensions of an element. If the container has more room available, the element won't be enlarged beyond these bounds, even if the HorizontalAlignment and VerticalAlignment properties are set to Stretch.
Width and Height	Explicitly sets the size of an element. This setting overrides a Stretch value for the HorizontalAlignment or VerticalAlignment properties. However, this size won't be honored if it's outside of the bounds set by the MinWidth, MinHeight, MaxWidth, and MaxHeight.

## Alignment

In this example—a StackPanel with vertical orientation—the VerticalAlignment property has no effect because each element is given as much height as it needs and no more. However, the HorizontalAlignment *is* important. It determines where each element is placed in its row. By default HorizontalAlignment is Left for a label and Stretch for a Button.



```
<StackPanel>
  <Label HorizontalAlignment="Center">A Button Stack</Label>
  <Button HorizontalAlignment="Left">Button 1</Button>
  <Button HorizontalAlignment="Right">Button 2</Button>
  <Button>Button 3</Button>
  <Button>Button 4</Button>
</StackPanel>
```



### Margin

A well-designed window doesn't just contain elements, it also includes a bit of extra space in between the elements. To introduce this extra space and make the StackPanel less cramped, set control margins.

When setting margins, you can set a single width for all sides, like this:

```
<Button Margin="5">Button 3</Button>
```

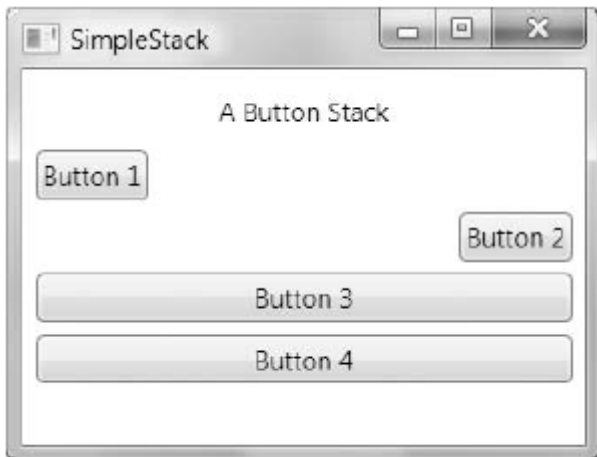
Alternatively, you can set different margins for each side of a control in the order *left, top, right, bottom*:

```
<Button Margin="5,10,5,10">Button 3</Button>
```

In code, margins are set using the Thickness structure:

```
cmd.Margin = New Thickness(5)
```

```
<StackPanel Margin="3">
  <Label Margin="3" HorizontalAlignment="Center">
    A Button Stack</Label>
  <Button Margin="3" HorizontalAlignment="Left">Button 1</Button>
  <Button Margin="3" HorizontalAlignment="Right">Button 2</Button>
  <Button Margin="3">Button 3</Button>
  <Button Margin="3">Button 4</Button>
</StackPanel>
```



### Minimum, Maximum, and Explicit Sizes

Finally, every element includes Height and Width properties that allow you to give it an explicit size. maximum and minimum size properties can be used to lock your control into the right range.

When the StackPanel sizes a button, it considers several pieces of information:

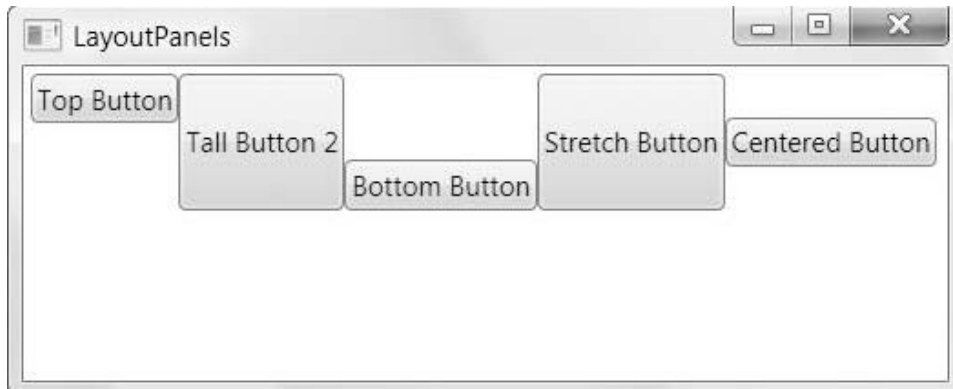
- **The minimum size.** Each button will always be at least as large as the minimum size.
- **The maximum size.** Each button will always be smaller than the maximum size (unless you've incorrectly set the maximum size to be smaller than the minimum size).
- **The content.** If the content inside the button requires a greater width, the StackPanel will attempt to enlarge the button. (You can find out the size that the button wants by examining the DesiredSize property, which returns the minimum width or the content width, whichever is greater.)
- **The size of the container.** If the minimum width is larger than the width of the StackPanel, a portion of the button will be cut off. Otherwise, the button will not be allowed to grow wider than the StackPanel, even if it can't fit all its text on the button surface.
- **The horizontal alignment.** Because the button uses a HorizontalAlignment of Stretch (the default), the StackPanel will attempt to enlarge the button to fill the full width of the StackPanel.

### THE WRAPPANEL AND DOCKPANEL

#### The WrapPanel

The WrapPanel lays out controls in the available space, one line or column at a time. By default, the WrapPanel.Orientation property is set to Horizontal; controls are arranged from left to right, and then on subsequent rows. However, use Vertical to place elements in multiple columns. Here's an example that defines a series of buttons with different alignments and places them into the WrapPanel:

```
<WrapPanel Margin="3">
  <Button VerticalAlignment="Top">Top Button</Button>
  <Button MinHeight="60">Tall Button 2</Button>
  <Button VerticalAlignment="Bottom">Bottom Button</Button>
  <Button>Stretch Button</Button>
  <Button VerticalAlignment="Center">Centered Button</Button>
</WrapPanel>
```

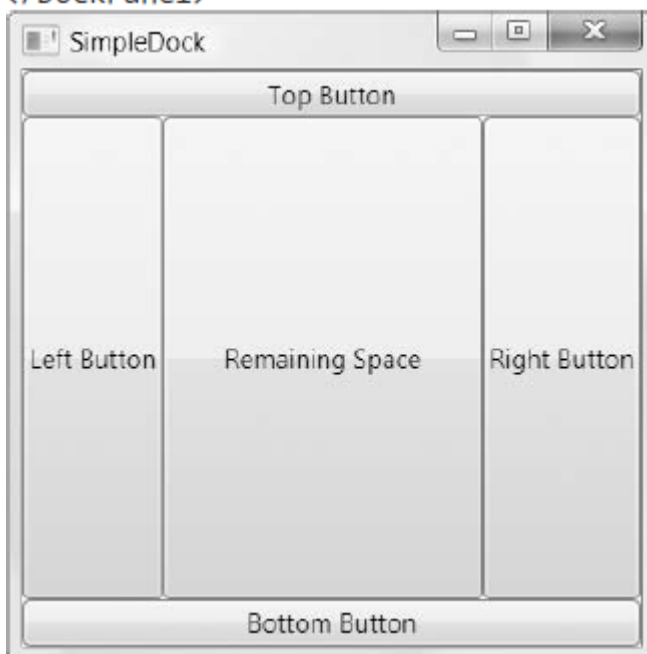


### THE DOCKPANEL

The DockPanel is a more interesting layout option. It stretches controls against one of its outside edges. The easiest way to visualize this is to think of the toolbars that sit at the top of many Windows applications. These toolbars are docked to the top of the window.

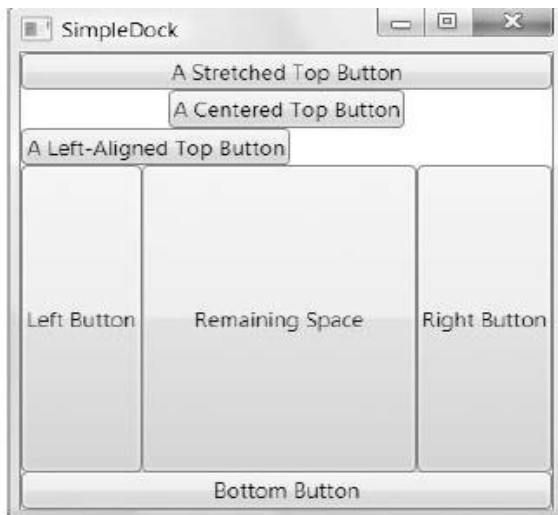
Here's an example that puts one button on every side of a DockPanel. This example also sets the LastChildFill to True, which tells the DockPanel to give the remaining space to the last element.

```
<DockPanel LastChildFill="True">
  <Button DockPanel.Dock="Top">Top Button</Button>
  <Button DockPanel.Dock="Bottom">Bottom Button</Button>
  <Button DockPanel.Dock="Left">Left Button</Button>
  <Button DockPanel.Dock="Right">Right Button</Button>
  <Button>Remaining Space</Button>
</DockPanel>
```



The user can dock several elements against the same side. In this case, the elements simply stack up against the side in the order they're declared in your markup.

```
<DockPanel LastChildFill="True">
  <Button DockPanel.Dock="Top">A Stretched Top Button</Button>
  <Button DockPanel.Dock="Top" HorizontalAlignment="Center">
    A Centered Top Button</Button>
  <Button DockPanel.Dock="Top" HorizontalAlignment="Left">
    A Left-Aligned Top Button</Button>
  <Button DockPanel.Dock="Bottom">Bottom Button</Button>
  <Button DockPanel.Dock="Left">Left Button</Button>
  <Button DockPanel.Dock="Right">Right Button</Button>
  <Button>Remaining Space</Button>
</DockPanel>
```



### **Nesting Layout Containers**

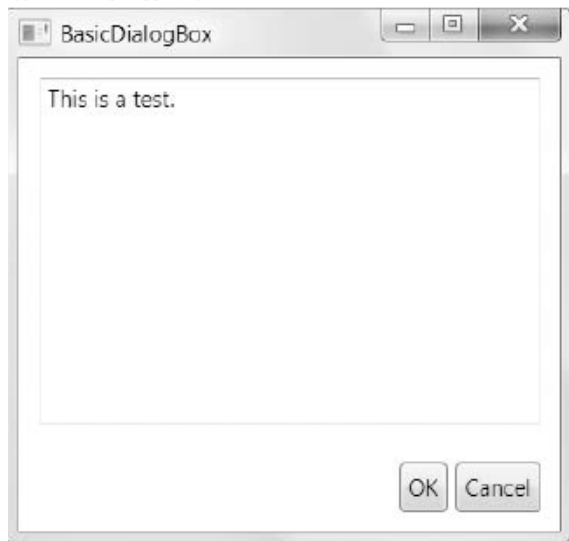
Use a DockPanel to place different StackPanel and WrapPanel containers in the appropriate regions of a window. For example, imagine you want to create a standard dialog box with OK and Cancel buttons in the bottom right-hand corner, and a large content region in the rest of the window.

There are several ways to model this interface with WPF, but the easiest option that uses the panels you've seen so far is as follows:

1. Create a horizontal StackPanel to wrap the OK and Cancel buttons together.
2. Place the StackPanel in a DockPanel and use that to dock it to the bottom of the window.
3. Set DockPanel.LastChildFill to True, so you can use the rest of the window to fill in other content. You can add another layout control here, or just an ordinary TextBox control (as in this example).
4. Set the margin properties to give the right amount of whitespace.

Here's the final markup:

```
<DockPanel LastChildFill="True">
  <StackPanel DockPanel.Dock="Bottom" HorizontalAlignment="Right"
    Orientation="Horizontal">
    <Button Margin="10,10,2,10" Padding="3">OK</Button>
    <Button Margin="2,10,10,10" Padding="3">Cancel</Button>
  </StackPanel>
  <TextBox DockPanel.Dock="Top" Margin="10">This is a test.</TextBox>
</DockPanel>
```



## THE GRID

The Grid is the most powerful layout container in WPF. The Grid is also an ideal tool for carving your window into smaller regions that user can manage with other panels. In fact, the Grid is so useful that when adding a new XAML document for a window in Visual Studio, it automatically adds the Grid tags as the first-level container, nested inside the root Window element. The Grid separates elements into an invisible grid of rows and columns.

Creating a Grid-based layout is a two-step process. First, choose the number of columns and rows that you want. Next, assign the appropriate row and column to each contained element, thereby placing it in just the right spot. Create grids and rows by filling the Grid.ColumnDefinitions and Grid.RowDefinitions collections with objects.

For example, if you decide you need two rows and three columns, you'd add the following tags:

```
<Grid ShowGridLines="True">
  <Grid.RowDefinitions>
    <RowDefinition></RowDefinition>
    <RowDefinition></RowDefinition>
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition></ColumnDefinition>
    <ColumnDefinition></ColumnDefinition>
    <ColumnDefinition></ColumnDefinition>
  </Grid.ColumnDefinitions>
```

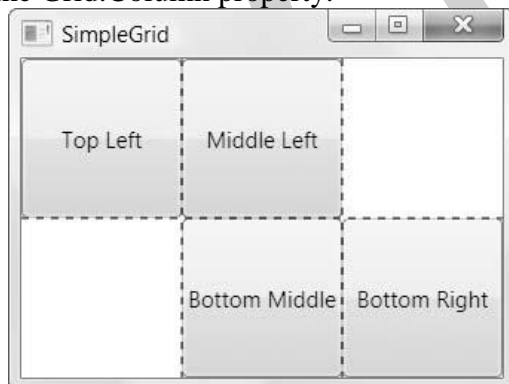
```
...
</Grid>
```

To place individual elements into a cell, use the attached Row and Column properties. Both these properties take 0-based index numbers. For example, create a partially filled grid of buttons:

```
<Grid ShowGridLines="True">
  ...

  <Button Grid.Row="0" Grid.Column="0">Top Left</Button>
  <Button Grid.Row="0" Grid.Column="1">Middle Left</Button>
  <Button Grid.Row="1" Grid.Column="2">Bottom Right</Button>
  <Button Grid.Row="1" Grid.Column="1">Bottom Middle</Button>
</Grid>
```

If user doesn't specify the Grid.Row property, the Grid assumes that it's 0. The same behavior applies to the Grid.Column property.



### Fine-Tuning Rows and Columns

The user can change the way each row and column is sized. The Grid supports three sizing strategies:

- **Absolute sizes.** Choose the exact size using device-independent units. This is the least useful strategy because it's not flexible enough to deal with changing content size, changing container size, or localization.
- **Automatic sizes.** Each row or column is given exactly the amount of space it needs, and no more. This is one of the most useful sizing modes.



- **Proportional sizes.** Space is divided between a group of rows or columns. This is the standard setting for all rows and columns.

You set the sizing mode using the Width property of the ColumnDefinition object or the Height property of the RowDefinition object to a number. For example, here's how you set an absolute width of 100 device-independent units:

```
<ColumnDefinition Width="100"></ColumnDefinition>
```

To use automatic sizing, you use a value of Auto:

```
<ColumnDefinition Width="Auto"></ColumnDefinition>
```

Finally, to use proportional sizing, you use an asterisk (\*):

```
<ColumnDefinition Width="*"></ColumnDefinition>
```

### Spanning Rows and Columns

Elements are placed in cells using the Row and Column attached properties. Use two more attached properties to make an element stretch over several cells: RowSpan and ColumnSpan. These properties take the number of rows or columns that the element should occupy.

For example, this button will take all the space that's available in the first and second cell of the first row:

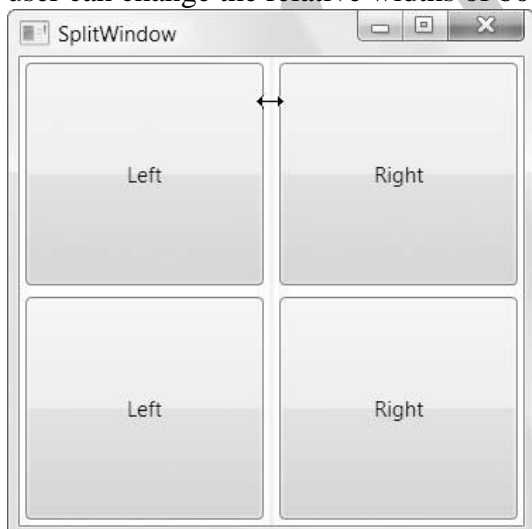
```
<Button Grid.Row="0" Grid.Column="0" Grid.RowSpan="2">Span Button</Button>
```

And this button will stretch over four cells in total by spanning two columns and two rows:

```
<Button Grid.Row="0" Grid.Column="0" Grid.RowSpan="2" Grid.ColumnSpan="2">  
Span Button</Button>
```

### Split Windows

In WPF, splitter bars are represented by the GridSplitter class and are a feature of the Grid. By adding a GridSplitter to a Grid, user were able to resize rows or columns. By dragging the splitter bar, the user can change the relative widths of both columns.



### Shared Size Groups

A Grid contains a collection of rows and columns, which are sized explicitly, proportionately, or based on the size of their children. There's one other way to size a row or a column—to match the size of another row or column. This works through a feature called *shared size groups*. The goal of shared size groups is to keep separate portions of your user interface consistent. For example, you might want to size one column to fit its content and size another column to match that size exactly. Creating a shared group



is easy. You simply need to set the `SharedSizeGroup` property on both columns, using a matching string. In the current example, both columns use a group named `TextLabel`:

```
<Grid Margin="3" Background="LightYellow" ShowGridLines="True">
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="Auto" SharedSizeGroup="TextLabel"></ColumnDefinition>
    <ColumnDefinition Width="Auto"></ColumnDefinition>
    <ColumnDefinition></ColumnDefinition>
  </Grid.ColumnDefinitions>

  <Label Margin="5">A very long bit of text</Label>
  <Label Grid.Column="1" Margin="5">More text</Label>
  <TextBox Grid.Column="2" Margin="5">A text box</TextBox>
</Grid>

...
<Grid Margin="3" Background="LightYellow" ShowGridLines="True">
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="Auto" SharedSizeGroup="TextLabel"></ColumnDefinition>
    <ColumnDefinition></ColumnDefinition>
  </Grid.ColumnDefinitions>

  <Label Margin="5">Short</Label>
  <TextBox Grid.Column="1" Margin="5">A text box</TextBox>
</Grid>
```

## THE UNIFORMGRID

There is a grid that breaks all the rules you've learned about so far: the `UniformGrid`. Unlike the `Grid`, the `UniformGrid` doesn't require (or even support) predefined columns and rows. Instead, you simply set the `Rows` and `Columns` properties to set its size. Each cell is always the same size because the available space is divided equally. Finally, elements are placed into the appropriate cell based on the order in which you define them. There are no attached `Row` and `Column` properties, and no blank cells.

Here's an example that fills a `UniformGrid` with four buttons:

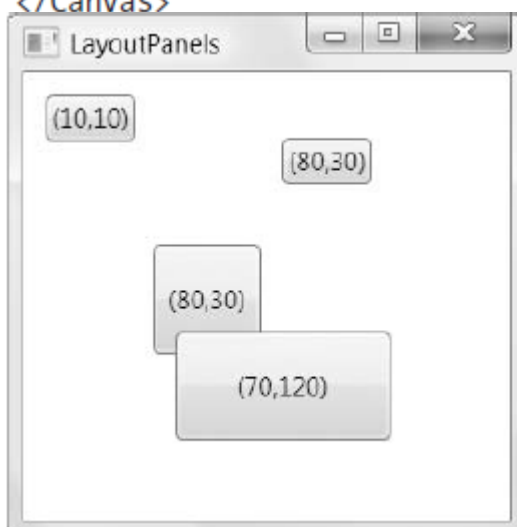
```
<UniformGrid Rows="2" Columns="2">
  <Button>Top Left</Button>
  <Button>Top Right</Button>
  <Button>Bottom Left</Button>
  <Button>Bottom Right</Button>
</UniformGrid>
```

## COORDINATE-BASED LAYOUT WITH THE CANVAS

It allows you to place elements using exact coordinates, which is a poor choice for designing rich data-driven forms and standard dialog boxes. To position an element on the `Canvas`, you set the attached `Canvas.Left` and `Canvas.Top` properties. `Canvas.Left` sets the number of units between the left edge of your element and the left edge of the `Canvas`. `Canvas.Top` sets the number of units between the top of your element and the top of the `Canvas`.

Here's a simple Canvas that includes four buttons:

```
<Canvas>
  <Button Canvas.Left="10" Canvas.Top="10">(10,10)</Button>
  <Button Canvas.Left="120" Canvas.Top="30">(120,30)</Button>
  <Button Canvas.Left="60" Canvas.Top="80" Width="50" Height="50">
    (60,80)</Button>
  <Button Canvas.Left="70" Canvas.Top="120" Width="100" Height="50">
    (70,120)</Button>
</Canvas>
```



If you resize the window, the Canvas stretches to fill the available space, but none of the controls in the Canvas moves or changes size. Like any other layout container, the Canvas can be nested inside a user interface.

### **Z-Order**

If there is more than one overlapping element, set the attached Canvas.ZIndex property to control how they are layered. Ordinarily, all the elements you add have the same ZIndex—0. When elements have the same ZIndex, they're displayed in the same order that they exist in Canvas.Children collection, which is based on the order that they're defined in the XAML markup. Elements declared later in the markup—such as button (70,120)—are displayed overtop of elements that are declared earlier—such as button (120,30).

```
<Button Canvas.Left="60" Canvas.Top="80" Canvas.ZIndex="1" Width="50" Height="50">
  (60,80)</Button>
<Button Canvas.Left="70" Canvas.Top="120" Width="100" Height="50">
  (70,120)</Button>
```

### **The InkCanvas**

WPF also includes an InkCanvas element that's similar to the Canvas in some respects (and wholly different in others). Like the Canvas, the InkCanvas defines four attached properties that you can apply to child elements for coordinate-based positioning (Top, Left, Bottom, and Right). However, the

underlying plumbing is quite a bit different—in fact, the InkCanvas doesn't derive from Canvas, or even from the base Panel class. Instead, it derives directly from FrameworkElement.

The primary purpose of the InkCanvas is to allow *stylus* input. The stylus is the penlike input device that's used in tablet PCs. However, the InkCanvas works with the mouse in the same way as it works with the stylus. Thus, a user can draw lines or select and manipulate elements in the InkCanvas using the mouse. The InkCanvas actually holds two collections of child content. The familiar Children collection holds arbitrary elements, just as with the Canvas. Each element can be positioned based on the Top, Left, Bottom, and Right properties. The Strokes collection holds System.Windows.Ink.Stroke objects, which represent graphical input that the user has drawn in the InkCanvas. Each line or curve that the user draws becomes a separate Stroke object. The strokes are drawn at runtime by the user.

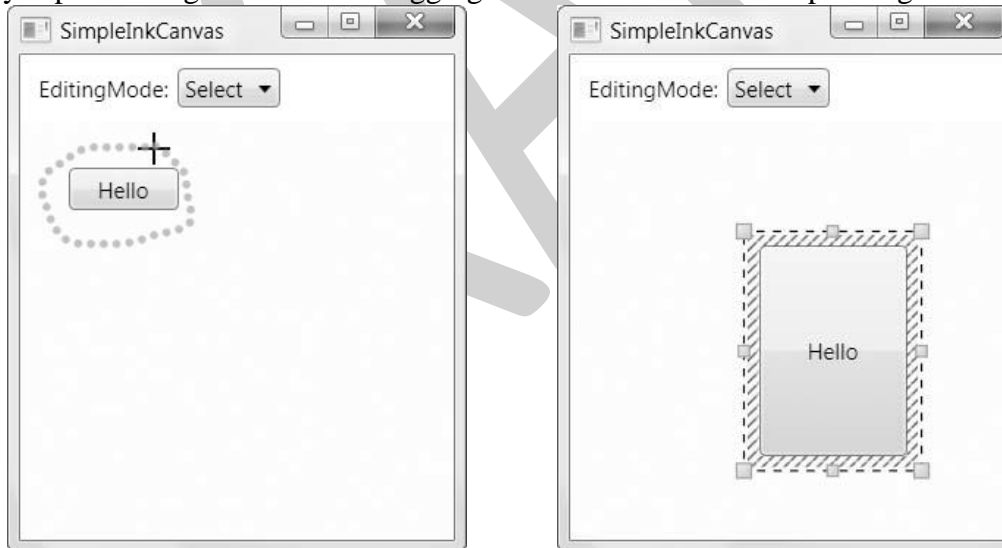
```
<InkCanvas Name="inkCanvas" Background="LightYellow"
  EditingMode="Ink">
  <Image Source="office.jpg" InkCanvas.Top="10" InkCanvas.Left="10"
    Width="287" Height="319"></Image>
</InkCanvas>
```



The InkCanvas can be used in some significantly different ways, depending on the value you set for the InkCanvas.EditingMode property.

Name	Description
Ink	The InkCanvas allows the user to draw annotations. This is the default mode. When the user draws with the mouse or stylus, a stroke is drawn.
GestureOnly	The InkCanvas doesn't allow the user to draw stroke annotations but pays attention to specific predefined <i>gestures</i> (such as dragging the stylus in one direction, or scratching out content). The full list of recognized gestures is listed by the System.Windows.Ink.ApplicationGesture enumeration.
InkAndGesture	The InkCanvas allows the user to draw stroke annotations and also recognizes predefined gestures.
EraseByStroke	The InkCanvas erases a stroke when it's clicked. If the user has a stylus, he can switch to this mode by using the back end of the stylus. (You can determine the current mode using the read-only ActiveEditingMode property, and you can change the mode used for the back end of the stylus by changing the EditingModeInverted property.)
EraseByPoint	The InkCanvas erases a portion of a stroke (a point in a stroke) when that portion is clicked.
Select	The InkCanvas allows the user to select elements that are stored in the Children collection. To select an element, the user must click it or drag a selection "lasso" around it. Once an element is selected, it can be moved, resized, or deleted.
None	The InkCanvas ignores mouse and stylus input.

The InkCanvas raises events when the editing mode changes (ActiveEditingModeChanged), a gesture is detected in GestureOnly or InkAndGesture mode (Gesture), a stroke is drawn (StrokeCollected), a stroke is erased (StrokeErasing and StrokeErased), and an element is selected or changed in Select mode (SelectionChanging, SelectionChanged, SelectionMoving, SelectionMoved, SelectionResizing, and SelectionResized). The events that end in *ing* represent an action that is about to take place but can be canceled by setting the Cancel property of the EventArgs object. In Select mode, the InkCanvas provides a fairly capable design surface for dragging content around and manipulating it.



## CLASSIC CONTROLS

The most fundamental WPF controls, includes basic controls such as labels, buttons, and text boxes.

## THE CONTROL CLASS

WPF windows are filled with elements, but only some of these elements are *controls*. Controls are *user-interactive* elements—elements that can take focus and receive input from the keyboard or mouse. All controls derive from the `System.Windows.Control` class, which adds a bit of basic infrastructure:

- The ability to set the alignment of content inside the control
- The ability to set the tab order
- Support for painting a background, foreground, and border
- Support for formatting the size and font of text content.

### Background and Foreground Brushes

All controls include the concept of a background and foreground. In WPF, set the color of these two areas (but not the content) using the Background and Foreground properties.

#### Setting Colors in Code

Imagine you want to set a blue surface area inside a button named `cmd`. Here's the code that does the trick:

```
cmd.Background = New SolidColorBrush(Colors.AliceBlue)
```

This code creates a new `SolidColorBrush` using a ready-made color via a shared property of the handy `Colors` class. It then sets the brush as the background brush for the button, which causes its background to be painted a light shade of blue.

User can also create a `Color` object by supplying the R, G, B values (red, green, and blue). Each one of these values is a number from 0 to 255:

```
Dim red As Integer = 0
```

```
Dim green As Integer = 255
```

```
Dim blue As Integer = 0
```

```
cmd.Foreground = New SolidColorBrush(Color.FromRgb(red, green, blue))
```

#### Setting Colors in XAML

When you set the background or foreground in XAML, you can use a helpful shortcut. Rather than define a `Brush` object, you can supply a color name or color value. The WPF parser will automatically create a `SolidColorBrush` object using the color you specify, and it will use that brush object for the foreground or background. Here's an example that uses a color name:

```
<Button Background="Red">A Button</Button>
```

It's equivalent to this more verbose syntax:

```
<Button>A Button
  <Button.Background>
    <SolidColorBrush Color="Red" />
  </Button.Background>
</Button>
```

#### Transparency

WPF supports true transparency. There are two ways to make an element partly transparent:

- **Set the Opacity property.** *Opacity* is a fractional value from 0 to 1, where 1 is completely solid (the default) and 0 is completely transparent. The *Opacity* property is defined in the `UIElement` class (and the base `Brush` class), so it applies to all elements.
- **Use a semitransparent color.** Any color that has an alpha value less than 255 is semitransparent.



```
<StackPanel Margin="5">
    <StackPanel.Background>
        <ImageBrush ImageSource="celestial.jpg" Opacity="0.7"/>
    </StackPanel.Background>

    <Button Foreground="White" FontSize="16" Margin="10"
        BorderBrush="White" Background="#60AA4030"
        Padding="20">A Semi-Transparent Button</Button>
    <Label Margin="10" FontSize="18" FontWeight="Bold" Foreground="White">
        Some Label Text</Label>
    <TextBox Margin="10" Background="#AAAAAAA" Foreground="White"
        BorderBrush="White">A semi-transparent text box</TextBox>

    <Button Margin="10" Padding="25" BorderBrush="White">
        <Button.Background>
            <ImageBrush ImageSource="happyface.jpg" Opacity="0.6"
                TileMode="Tile" Viewport="0,0,0.1,0.3"/>
        </Button.Background>

        <StackPanel>
            <TextBlock Foreground="#75FFFFFF" TextAlignment="Center"
                FontSize="30" FontWeight="Bold" TextWrapping="Wrap">
                Semi-Transparent Layers</TextBlock>
        </StackPanel>
    </Button>
</StackPanel>
```

## Fonts

The Control class defines a small set of font-related properties that determine how text appears in a control.

Name	Description
FontStyle	The angling of the text, as represented as a FontStyle object.
FontWeight	The heaviness of text, as represented as a FontWeight object. Bold is the most obvious of these, but some typefaces provide other variations such as Heavy, Light, ExtraBold, and so on.
FontStretch	The amount that text is stretched or compressed, as represented by a FontStretch object.

When choosing a font, you must supply the full family name, as shown here:

```
<Button Name="cmd" FontFamily="Times New Roman" FontSize="18">A Button</Button>
```

It's much the same in code:

```
cmd.FontFamily = "Times New Roman"
```

```
cmd.FontSize = "18"
```

### ***Text Decorations and Typography***

Some elements also support more advanced text manipulation through the `TextDecorations` and `Typography` properties. It provides just four decorations, each of which allows you to add some sort of line to your text. They include `Baseline`, `OverLine`, `Strikethrough`, and `Underline`. The `Typography` property is more advanced—it lets you access specialized typeface variants that only some fonts will provide. Examples include different number alignments, ligatures (connections between adjacent letters), and small caps.

```
<TextBlock TextDecorations="Underline">Underlined text</TextBlock>
```

### ***Font Substitution***

WPF does give you a little flexibility with a font fallback system. You can set `FontFamily` to a comma-separated list of font options. WPF will then move through the list in order, trying to find one of the fonts you've indicated. Here's an example that attempts to use `Technical Italic` font but falls back to `Comic Sans MS` or `Arial` if that isn't available:

```
<Button FontFamily="Technical Italic, Comic Sans MS, Arial">A Button</Button>
```

Incidentally, you can get a list of all the fonts that are installed on the current computer using the shared `SystemFontFamilies` collection of the `System.Windows.Media.Fonts` class. Here's an example that uses it to add fonts to a list box:

```
For Each fontFamily As FontFamily In Fonts.SystemFontFamilies
```

```
    lstFonts.Items.Add(fontFamily.Source)
```

```
Next
```

### ***Mouse Cursors***

A common task in any application is to adjust the mouse cursor to show when the application is busy or to indicate how different controls work. You can set the mouse pointer for any element using the `Cursor` property, which is inherited from the `FrameworkElement` class. Every cursor is represented by a `System.Windows.Input.Cursor` object. The easiest way to get a `Cursor` object is to use the shared properties of the `Cursors` class (from the `System.Windows.Input` namespace). They include all the standard Windows cursors, such as the hourglass, the hand, resizing arrows, and so on. Here's an example that sets the hourglass for the current window:

```
Me.Cursor = Cursors.Wait
```

Now when you move the mouse over the current window, the mouse pointer changes to the familiar hourglass icon (in Windows XP) or the swirl (in Windows Vista).

## **CONTENT CONTROLS**

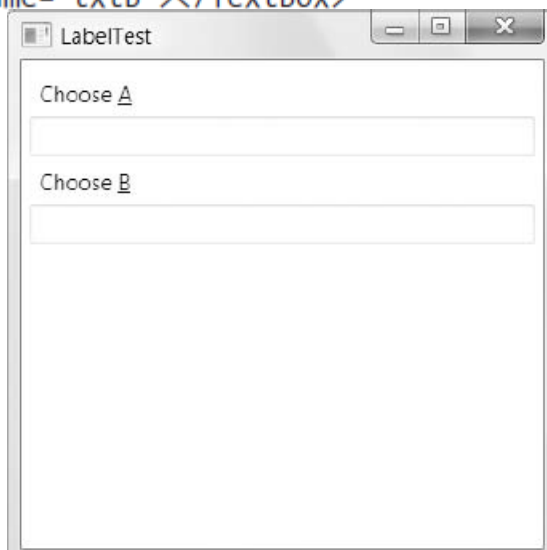
These include the well-worn `Label`, `Button`, `CheckBox`, and `RadioButton`.

### ***Labels***

The simplest of all content controls is the `Label` control. Like any other content control, it accepts any single piece of content you want to place inside. But what distinguishes the `Label` control is its support for *mnemonics*—essentially, shortcut keys that set the focus to a linked control. To support this functionality, the `Label` control adds a single property, named `Target`. To set the `Target` property, you need to use a binding expression that points to another control. Here's the syntax you must use:



```
<Label Target="{Binding ElementName=txtA}">Choose _A</Label>
<TextBox Name="txtA"></TextBox>
<Label Target="{Binding ElementName=txtB}">Choose _B</Label>
<TextBox Name="txtB"></TextBox>
```



## Buttons

WPF recognizes three types of button controls: the familiar Button, the CheckBox, and the RadioButton. All of these controls are content controls that derive from ButtonBase. The ButtonBase class includes only a few members. It defines the Click event and adds support for commands, which allow you to wire buttons to higher-level application tasks.

The ButtonBase class adds a ClickMode property, which determines when a button fires its Click event in response to mouse actions. The default value is ClickMode.Release, which means the Click event fires when the mouse is clicked and released. However, you can also choose to fire the Click event mouse when the mouse button is first pressed (ClickMode.Press) or, oddly enough, whenever the mouse moves over the button and pauses there (ClickMode.Hover).

### The Button

The Button class represents the ever-present Windows push button. It adds just two writeable properties, IsCancel and IsDefault:

- **When IsCancel is True**, this button is designated as the cancel button for a window. If you press the Escape key while positioned anywhere on the current window, this button is triggered.
- **When IsDefault is True**, this button is designated as the default button (also known as the accept button). If Enter key is pressed, this button is triggered.

### The ToggleButton and RepeatButton

Alongside Button, three more classes derive from ButtonBase. These include the following:

- GridViewColumnHeader, which represents the clickable header of a column when you use a grid-based ListView.
- RepeatButton, which fires Click events continuously, as long as the button is held down. Ordinary buttons fire one Click event per user click.

- **ToggleButton**, which represents a button that has two states (pushed or unpushed). When you click a **ToggleButton**, it stays in its pushed state until you click it again to release it. This is sometimes described as “sticky click” behavior.

### **The CheckBox**

Both the **CheckBox** and the **RadioButton** are buttons of a different sort. They derive from **ToggleButton**, which means they can be switched on or off by the user, hence their “toggle” behavior. In the case of the **CheckBox**, switching the control “on” means placing a check mark in it.

To assign a null value in WPF markup, you need to use the null markup extension, as shown here:

```
<CheckBox IsChecked="{x:Null}">A check box in indeterminate state</CheckBox>
```

The **ToggleButton** class also defines three events that fire when the check box enters specific states: **Checked**, **Unchecked**, and **Indeterminate**.

### **The RadioButton**

The **RadioButton** also derives from **ToggleButton** and uses the same **IsChecked** property and the same **Checked**, **Unchecked**, and **Indeterminate** events. Along with these, the **RadioButton** adds a single property named **GroupName**, which allows you to control how radio buttons are placed into groups. Ordinarily, radio buttons are grouped by their container. That means if you place three **RadioButton** controls in a single **StackPanel**, they form a group from which you can select just one of the three. On the other hand, if you place a combination of radio buttons in two separate **StackPanel** controls, you have two independent groups.

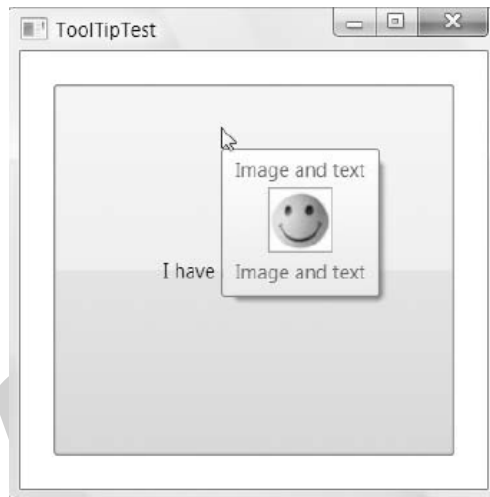
```
<StackPanel>
    <GroupBox Margin="5">
        <StackPanel>
            <RadioButton>Group 1</RadioButton>
            <RadioButton>Group 1</RadioButton>
            <RadioButton>Group 1</RadioButton>
            <RadioButton Margin="0,10,0,0" GroupName="Group2">Group 2</RadioButton>
        </StackPanel>
    </GroupBox>
    <GroupBox Margin="5">
        <StackPanel>
            <RadioButton>Group 3</RadioButton>
            <RadioButton>Group 3</RadioButton>
            <RadioButton>Group 3</RadioButton>
            <RadioButton Margin="0,10,0,0" GroupName="Group2">Group 2</RadioButton>
        </StackPanel>
    </GroupBox>
</StackPanel>
```

Here, there are two containers holding radio buttons, but three groups. The final radio button at the bottom of each group box is part of a third group.

### **Tooltips**

WPF has a flexible model for *tooltips*. The easiest way to show a tooltip doesn’t involve using the **ToolTip** class directly. Instead, set the **ToolTip** property of the element. The **ToolTip** property is defined in the **FrameworkElement** class, so it’s available on anything in a WPF window. For example, here’s a button that has a basic tooltip:

```
<Button ToolTip="This is my tooltip">I have a tooltip</Button>  
<Button>  
    <Button.ToolTip>  
        <StackPanel>  
            <TextBlock Margin="3" >Image and text</TextBlock>  
            <Image Source="happyface.jpg" Stretch="None" />  
            <TextBlock Margin="3" >Image and text</TextBlock>  
        </StackPanel>  
    </Button.ToolTip>  
    <Button.Content>I have a fancy tooltip</Button.Content>  
</Button>
```



### *Setting ToolTip Properties*

**Class: III B. Sc. [CT]**

**Course Name: .NET Programming**

**Course Code: 16CTU502A**

**Unit: III (Working with WPF)**

**Batch- 2016-2019**

Name	Description
HasDropShadow	Determines whether the tooltip has a diffuse black drop shadow that makes it stand out from the window underneath.
Placement	Determines how the tooltip is positioned, using one of the values from the PlacementMode enumeration. The default value is Mouse, which means that the top-left corner of the tooltip is placed relative to the current mouse position. (The actual position of the tooltip may be offset from this starting point based on the HorizontalOffset and VerticalOffset properties.) Other possibilities allow you to place the tooltip using absolute screen coordinates or place it relative to some element (which you indicate using the PlacementTarget property).
HorizontalOffset and VerticalOffset	Allows you to nudge the tooltip into the exact position you want. You can use positive or negative values.
PlacementTarget	Allows you to place a tooltip relative to another element. In order to use this property, the Placement property must be set to Left, Right, Top, Bottom, or Center. (This is the edge of the element to which the tooltip is aligned.)
PlacementRectangle	Allows you to offset the position of the tooltip. This works in much the same way as the HorizontalOffset and VerticalOffset properties. This property doesn't have an effect if Placement property is set to Mouse.

Name	Description
CustomPopupPlacementCallback	Allows you to position a tooltip dynamically using code. If the Placement property is set to Custom, this property identifies the method that will be called by the ToolTip to get the position where the ToolTip should be placed. Your callback method receives three pieces of information—popupSize (the size of the ToolTip), targetSize (the size of the PlacementTarget, if it's used), and offset (a point that's created based on HorizontalOffset and VerticalOffset properties). The method returns a CustomPopupPlacement object that tells WPF where to place the tooltip.
StaysOpen	Has no effect in practice. The intended purpose of this property is to allow you to create a tooltip that remains open until the user clicks somewhere else. However, the ToolTipService.ShowDuration property overrides the StaysOpen property. As a result, tooltips always disappear after a configurable amount of time (usually about 5 seconds) or when the user moves the mouse away. If you want to create a tooltip-like window that stays open indefinitely, the easiest approach is to use the Popup control.

Using the ToolTip properties, the following markup creates a tooltip that has no drop shadow but uses a transparent red background that lets the underlying window (and controls) show through:

```

<Button>
  <Button.ToolTip>
    <ToolTip Background="#60AA4030" Foreground="White"
      HasDropShadow="False" >
      <StackPanel>
        <TextBlock Margin="3" >Image and text</TextBlock>
        <Image Source="happyface.jpg" Stretch="None" />
        <TextBlock Margin="3" >Image and text</TextBlock>
      </StackPanel>
    </ToolTip>
  </Button.ToolTip>
  <Button.Content>I have a fancy tooltip</Button.Content>
</Button>

```

Here are some strategies you can use to place a tooltip:

- **Based on the current position of the mouse.** This is the standard behavior, which relies on Placement being set to Mouse. The top-left corner of the tooltip box is lined up with the bottom-left corner of the invisible “bounding box” around the mouse pointer.
- **Based on the position of the moused-over element.** Set the Placement property to Left, Right, Top, Bottom, or Center, depending on the edge of the element. The top-left corner of the tooltip box will be lined up with that edge.
- **Based on the position of another element (or the window).** Set the Placement property in the same way you would if you were lining the tooltip up with the current element. (Use the value Left, Right, Top, Bottom, or Center.) Then choose the element by setting the PlacementTarget property. Remember to use the {Binding ElementName=Name} syntax to identify the element you want to use.
- **With an offset.** Use any of the strategies described previously, but set the HorizontalOffset and VerticalOffset properties to add a little extra space.
- **Using absolute coordinates.** Set Placement to Absolute and use the HorizontalOffset and VerticalOffset properties (or the PlacementRectangle) to set some space between the tooltip and the top-left corner of the window.
- **Using a calculation at runtime.** Set Placement to Custom. Set the CustomPopup PlacementCallback property to point to a method that created.

### Setting ToolTipService Properties

There are some tooltip properties that can't be configured using the properties of the ToolTip class. In this case, use a different class, which is named ToolTipService. ToolTipService allows to configure the time delays associated with the display of a tooltip. All the properties of the ToolTipService class are attached properties, so set them directly in the control tag, as shown here:

```

<Button ToolTipService.InitialShowDelay="1">
  ...
</Button>

```



The ToolTipService class defines many of the same properties as ToolTip. This allows to use a simpler syntax when dealing with text-only tooltips. Rather than adding a nested ToolTip element, set everything using attributes:

```
<Button ToolTip="This tooltip is aligned with the bottom edge"
        ToolTipService.Placement="Bottom">I have a tooltip</Button>
```

Name	Description
InitialShowDelay	Sets the delay (in milliseconds) before this tooltip is shown when the mouse hovers over the element.
ShowDuration	Sets the amount of time (in milliseconds) that this tooltip is shown before it disappears, if the user does not move the mouse.
BetweenShowDelay	Sets a time window (in milliseconds) during which the user can move between tooltips without experiencing the InitialShowDelay. For example, if BetweenShowDelay is 5000, the user has five seconds to move to another control that has a tooltip. If the user moves to another control within that time period, the new tooltip is shown immediately. If the user takes longer, the BetweenShowDelay window expires, and the InitialShowDelay kicks into action. In this case, the second tooltip isn't shown until after the InitialShowDelay period.
ToolTip	Sets the content for the tooltip. Setting ToolTipService.ToolTip is equivalent to setting the FrameworkElement.ToolTip property of an element.
HasDropShadow	Determines whether the tooltip has a diffuse black drop shadow that makes it stand out from the window underneath.
ShowOnDisabled	Determines the tooltip behavior when the associated element is disabled. If True, the tooltip will appear for disabled elements (elements that have their IsEnabled property set to False). The default is False, in which case the tooltip appears only if the associated element is enabled.
Placement, PlacementTarget, PlacementRectangle, and VerticalOffset	Allows you to control the placement of the tooltip. These properties work in the same way as the matching properties of the ToolTipHorizontalOffset class.
IsEnabled and IsOpen	Allows you to control the tooltip in code. IsEnabled allows you to temporarily disable a ToolTip, and IsOpen allows you to programmatically show or hide a tooltip (or just check whether the tooltip is open).

### ***The Popup***

The Popup control has a great deal in common with the ToolTip, although neither one derives from the other. Like the ToolTip, the Popup can hold a single piece of content, which can include any WPF element. The differences between the Popup and ToolTip are more important. They include the following:

- The Popup is never shown automatically. Set the IsOpen property for it to appear.
- By default, the Popup.StaysOpen property is set to True, and the Popup does not disappear until explicitly set its IsOpen property to False. If you set StaysOpen to False, the Popup disappears when the user clicks somewhere else.
- The Popup provides a PopupAnimation property that lets you control how it comes into view when you set IsOpen to True. Your options include None (the default), Fade (the opacity of the popup gradually increases), Scroll (the popup slides in from the upperleft corner of the window,

space permitting), and Slide (the popup slides down into place, space permitting). In order for any of these animations to work, you must also set the `AllowsTransparency` property to `True`.

- The Popup can accept focus. Thus, you can place user-interactive controls in it, such as a Button. This functionality is one of the key reasons to use the Popup instead of the ToolTip.
- The Popup control is defined in the `System.Windows.Controls.Primitives` namespace because it is most commonly used as a building block for more complex controls.

## **TEXT CONTROLS**

WPF includes three text-entry controls: `TextBox`, `RichTextBox`, and `PasswordBox`. The `PasswordBox` derives directly from `Control`. The `TextBox` and `RichTextBox` controls go through another level and derive from `TextBoxBase`.

### **Multiple Lines of Text**

Ordinarily, the `TextBox` control stores a single line of text. (You can limit the allowed number of characters by setting the `MaxLength` property.) However, there are many cases to create a multiline text box for dealing with large amounts of content. In this case, set the `TextWrapping` property to `Wrap` or `WrapWithOverflow`. `Wrap` always breaks at the edge of the control, even if it means severing an extremely long word in two. `WrapWithOverflow` allows some lines to stretch beyond the right edge if the line-break algorithm.

### **Text Selection**

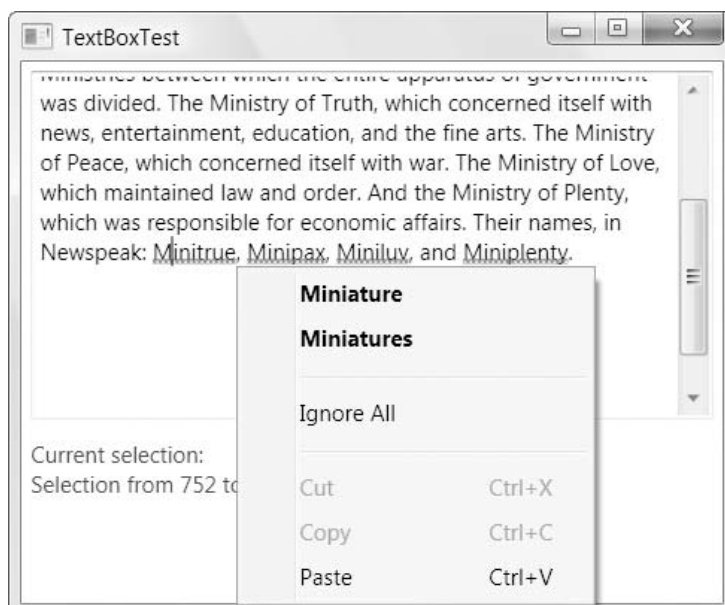
The `TextBox` class also gives the ability to determine or change the currently selected text programmatically, using the `SelectionStart`, `SelectionLength`, and `SelectedText` properties. `SelectionStart` identifies the zero-based position where the selection begins. For example, if you set this property to 10, the first selected character is the 11th character in the text box. The `SelectionLength` indicates the total number of selected characters. (A value of 0 indicates no selected characters.) Finally, the `SelectedText` property allows you to quickly examine or change the selected text in the text box.

### **Miscellaneous TextBox Features**

The `TextBox` includes a few more specialized frills. The most interesting is the spelling-checker feature, which underlines unrecognized words with a red squiggly line. The user can rightclick an unrecognized word and choose from a list of possibilities. To turn on this spelling-checker functionality for the `TextBox` control, set the `SpellCheck.IsEnabled` dependency property, as shown here:

```
<TextBox SpellCheck.IsEnabled="True">...</TextBox>
```





### The PasswordBox

The PasswordBox looks like a TextBox, but it displays a string of circle symbols to mask the characters it shows. (choose a different mask character by setting the PasswordChar property.) Additionally, the PasswordBox does not support the clipboard, so you can't copy the text inside.

### LIST CONTROLS

WPF includes many controls that wrap a collection of items, ranging from the simple ListBox and ComboBox. The class hierarchy that leads from ItemsControls is a bit tangled. One major branch is the *selectors*, which includes the ListBox, the ComboBox, and the TabControl. These controls derive from Selector and have properties that let you track down the currently selected item (SelectedItem) or its position (SelectedIndex).

### The ListBox

The ListBox and ComboBox class represent two common staples of Windows design— variable-length lists that allow the user to select an item. To add items to the ListBox, you can nest ListBoxItem elements inside the ListBox element. For example, here's a ListBox that contains a list of colors:

```
<ListBox>
  <ListBoxItem>Green</ListBoxItem>
  <ListBoxItem>Blue</ListBoxItem>
  <ListBoxItem>Yellow</ListBoxItem>
  <ListBoxItem>Red</ListBoxItem>
</ListBox>
```

For example, if you decided you want to create a list with images, you could create markup like this:

```
<ListBox>
  <ListBoxItem>
    <Image Source="happyface.jpg"></Image>
  </ListBoxItem>
  <ListBoxItem>
    <Image Source="happyface.jpg"></Image>
  </ListBoxItem>
</ListBox>
```

Here's a more ambitious example that uses nested StackPanel objects to combine text and image content:

```
<ListBox>
  <StackPanel Orientation="Horizontal">
    <Image Source="happyface.jpg" Width="30" Height="30"></Image>
    <Label VerticalContentAlignment="Center">A happy face</Label>
  </StackPanel>
  <StackPanel Orientation="Horizontal">
    <Image Source="redx.jpg" Width="30" Height="30"></Image>
    <Label VerticalContentAlignment="Center">A warning sign</Label>
  </StackPanel>
  <StackPanel Orientation="Horizontal">
    <Image Source="happyface.jpg" Width="30" Height="30"></Image>
    <Label VerticalContentAlignment="Center">A happy face</Label>
  </StackPanel>
</ListBox>
```



### The ComboBox

The ComboBox is similar to the ListBox control. It holds a collection of ComboBoxItem objects, which are created either implicitly or explicitly. As with the ListBoxItem, the ComboBoxItem is a content control that can contain any nested element. The ComboBox control uses a drop-down list, which means only one item can be selected at a time.

If you want to allow the user to type in text in the combo box to select an item, you must set the IsEditable property to True.

### RANGE-BASED CONTROLS

WPF includes three controls that use the concept of a *range*. These controls take a numeric value that falls in between a specific minimum and maximum value. These controls— ScrollBar, ProgressBar, and Slider—all derive from the RangeBase class (which itself derives from the Control class).

Name	Description
Value	This is the current value of the control (which must fall between the minimum and maximum). By default, it starts at 0.
Maximum	This is the upper limit (the largest allowed value).
Minimum	This is the lower limit (the smallest allowed value).
SmallChange	This is the amount the Value property is adjusted up or down for a “small change.” The meaning of a small change depends on the control (and may not be used at all). For the ScrollBar and Slider, this is the amount the value changes when you use the arrow keys. For the ScrollBar, you can also use the arrow buttons at either end of the bar.

LargeChange	This is the amount the Value property is adjusted up or down for a “large change.” The meaning of a large change depends on the control (and may not be used at all). For the ScrollBar and Slider, this is the amount the value changes when you use the Page Up and Page Down keys or when you click the bar on either side of the thumb (which indicates the current position).
-------------	--

### The ScrollBar

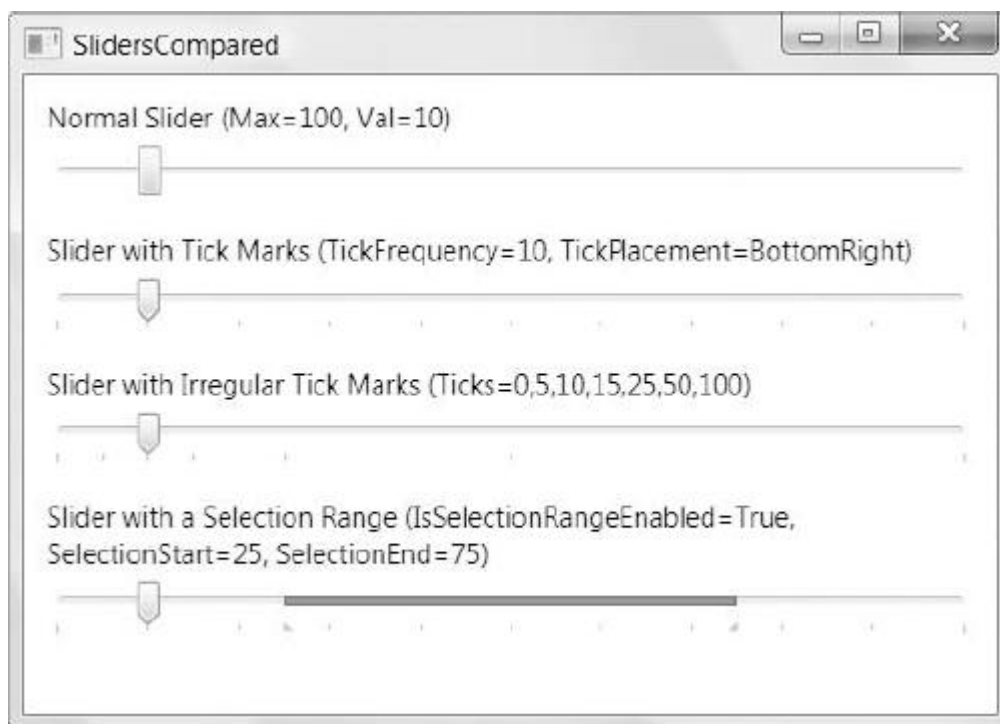
The ScrollViewer control provides a convenient way to enable scrolling of content in Windows Presentation Foundation (WPF) applications. There are two predefined elements that enable scrolling in WPF applications: ScrollBar and ScrollViewer. The ScrollViewer control encapsulates horizontal and vertical ScrollBar elements and a content container (such as a Panel element) in order to display other visible elements in a scrollable area. You must build a custom object in order to use the ScrollBar element for content scrolling. However, you can use the ScrollViewer element by itself because it is a composite control that encapsulates ScrollBar functionality.

The ScrollViewer control responds to both mouse and keyboard commands, and defines numerous methods with which to scroll content by predetermined increments. User can use the ScrollChanged event to detect a change in a ScrollViewer state. A ScrollViewer can only have one child, typically a Panel element that can host a Children collection of elements. The Content property defines the sole child of the ScrollViewer.

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      WindowTitle="ScrollViewer Sample">
    <ScrollViewer HorizontalScrollBarVisibility="Auto">
        <StackPanel VerticalAlignment="Top" HorizontalAlignment="Left">
            <TextBlock TextWrapping="Wrap" Margin="0,0,0,20">Scrolling is enabled
when it is necessary.
            Resize the window, making it larger and smaller.</TextBlock>
            <Rectangle Fill="Red" Width="500" Height="500"></Rectangle>
        </StackPanel>
    </ScrollViewer>
</Page>
```

### The Slider

The Slider is a specialized control that’s occasionally useful—for example, you might use it to set numeric values in situations where the number itself isn’t particularly significant. For example, it makes sense to set the volume in a media player by dragging the thumb in a slider bar from side to side. The general position of the thumb indicates the relative loudness (normal, quiet, and loud)



Name	Description
Orientation	Switches between a vertical and a horizontal slider.
Delay and Interval	Controls how fast the thumb moves along the track when you click and hold down either side of the slider. Both are millisecond values. The Delay is the time before the thumb moves one (small change) unit after you click, and the Interval is the time before it moves again if you continue holding the mouse button down.
TickPlacement	Determines where the tick marks appear. (Tick marks are notches that appear near the bar to help you visualize the scale.) By default, the TickPlacement is set to None, and no tick marks appear. If you have a horizontal slider, you can place the tick marks above (TopLeft) or below (BottomRight) the track. With a vertical slider, you can place them on the left (TopLeft) and right (BottomRight). (The TickPlacement names are a bit confusing because two values cover four possibilities, depending on the orientation of the slider.)
TickFrequency	Sets the interval in between ticks, which determines how many ticks appear. For example, you could place them every 5 numeric units, every 10, and so on.
Ticks	If you want to place ticks in specific, irregular positions, you can use the Ticks collection. Simply add one number (as a Double) to this collection for each tick mark. For example, you could place ticks at the positions 1, 1.5, 2, and 10 on the scale by adding these numbers.
IsSnapToTickEnabled	If True, when you move the slider, it automatically snaps into place, jumping to the nearest tick mark. The default is False.
IsSelectionRangeEnabled	If True, you can use a selection range to shade in a portion of the slider bar. You set the position selection range using the SelectionStart and SelectionEnd properties. The selection range has no intrinsic meaning, but you can use it for whatever purpose makes sense. For example, media players sometimes use a shaded background bar to indicate the download progress for a media file.

## The ProgressBar

The ProgressBar indicates the progress of a long-running task. Unlike the slider, the ProgressBar isn't user interactive. Instead, it's up to the code to periodically increment the Value property.

```
<ProgressBar Height="18" Width="200" IsIndeterminate="True"></ProgressBar>
```

When setting `IsIndeterminate`, no longer use the `Minimum`, `Maximum`, and `Value` properties. Instead, this `ProgressBar` shows a periodic green pulse that travels from left to right, which the universal Windows convention is indicating that there's work in progress. This sort of indicator makes particular sense in an application's status bar. For example, you could use it to indicate that you're contacting a remote server for information.

KAHE

**KARPAGAM ACADEMY OF HIGHER EDUCATION  
COIMBATORE - 21**

**DEPARTMENT OF COMPUTER SCIENCE, CA & IT**

**CLASS : III B.Sc COMPUTER TECHNOLOGY**

**BATCH : 2016-2019**

**Part -A Online Examinations**

**I One Marks**

**SUBJECT: . NET Programming**

**SUBJECT CODE: 16CTU502A**

**UNIT-III**

S.No.	QUESTION	Option 1	Option 2	Option 3	Option 4	Answer
1	WPF Stands for	Windows Presentati	Windows Program	Windows Presentati	Windows Procedur	Windows Presentati
2	_____ is designed for .NET, influenced by modern display technologies such as HTML and Flash, and hardware-acceleration.	WPF	WCM	WFM	WWM	WPF
3	_____ provides the familiar Windows look and feel for elements such as windows, buttons and text boxes	User32	GDI	GDI+	GUI	User32
4	_____ GDI/GDI+ provides drawing support for rendering shapes, text, and images at the cost of additional complexity	GDI/GDI+	CDI/CDI+	GUI/GUI+	CUII/CUI+	GDI/GDI+
5	DirectX introduced as an error-prone toolkit for creating games on the _____ platform	Linux	Unix	Windows	Red Hat	Windows
6	A WPF window and all the elements inside it are measured using _____	dependen t units	independ ent units	independ ent units	dependen t units	independ ent units
7	A single device-independent unit is defined as _____ of an inch	one by 96	one by 186	one by 248	one by 32	one by 96
8	The First Versions of WPF is _____	2	2.5	3	3.5	3
9	What is the name of the protocol used by .NET to marshal object requests across the Web?	HTTP	WSDL	TCP/IP	SOAP	SOAP
10	What does XAML stand for?	eXtraordi nary	eXtensibl e	eXtended Applicati	eXtreme Applicati	eXtensibl e



11	_____ are designed specifically to run in browser over the web	Web Forms	Windows Forms	XML Files	Component	Web Forms
12	_____ dynamically builds Web-based client server applications.	XML	VB	ASP.NET	None	ASP.NET
13	_____ holds base types, such as UIElement and Visual, from which all shapes and controls derive	PresentationCore.dll	Windows Base.dll	milcore.dll	Windows Codecs.dll	PresentationCore.dll
14	_____ is the low-level API through which all the graphics in a WPF are rendered.	Direct3D	User32	GDI	DirectX	Direct3D
15	XAML documents are arranged as _____ Structure	List	Tree	Stack	Queue	Tree
16	The _____ lays out controls in the available space, one line or column at a time.	WrapPanel	StackPanel	DockPanel	DoublePanel	WrapPanel
17	By default, the WrapPanel.Orientation property is set to _____	Horizontal	Vertical	Semi Vertical	Rectangular	Horizontal
18	Which property determines whether a control is displayed to the user?	Hide	Show	Visible	Enabled	Visible
19	The value for Standard Monitor Resolution is _____	768 pixels	868 pixels	800 pixels	968 pixels	768 pixels
20	_____ defines a set of portable instructions that are independent of any specific CPU.	CSIL	MSIL	Native code	Managed Code	MSIL
21	Which of the Following is not a .NET Supported Language	VB	C#	J#	FORTRAN	FORTRAN
22	The current status of a Web Forms page and its controls is called the	viewstate	webstatus	round trips	request/response	viewstate
23	.NET programs are translated using _____	compiler	interpreter	both	none	compiler
24	_____ is the function of the CLR	memory managem	security	garbage collection	All the above	All the above

25	Which is not a common control event?	Click	SingleClick	DoubleClick	MouseDown	SingleClick
26	XAML documents are arranged as a heavily _____ of elements.	Nested Tree	Binary Tree	TPR Tree	BBX Tree	Nested Tree
27	Data types standardized across the CLR are called _____	Standard Data	Common Type	Common Language	System Data	Common Type
28	The CTS equivalent of Integer data type is	System.Int32	System.Int16	System.Int64	System.Int24	System.Int32
29	The Size of 'System.Double' data type is	8 bits	2 bytes	4 bytes	8 bytes	8 bytes
30	_____ panel stretches controls against one of its outside edges	WrapPanel	StackPanel	DockPanel	DoublePanel	DockPanel
31	The _____ is the most powerful layout container in WPF.	Layout	Grid	Flexgrid	Datagrid	Grid
32	_____ an ideal tool for carving your window into smaller regions that user can manage with other panels	WrapPanel	StackPanel	DockPanel	Grid	Grid
33	_____ allows user to resize rows or columns in WPF Applications	Grid Splitter	Grid Extractor	Grid Merger	Grid Arranger	Grid Splitter
34	Which of the following is not a Text Control	TextBox	RichTextBox	PasswordBox	ListBox	ListBox
35	ScrollBar, ProgressBar, and Slider are derive from the _____ class	RangeBase	ControlBase	ListBase	QueryBase	RangeBase
36	scrolling of content in Windows Presentation Foundation (WPF) applications.	ScrollViewer	ImageViewer	ListViewer	GridViewer	ScrollViewer
37	_____Control provide an out-of-the-box spell checking functionality.	TextBox	ListBox	ComboBox	LabelBox	TextBox
38	The _____ control is a special type of TextBox designed to enter passwords	TextBox	ListBox	PictureBox	LabelBox	PictureBox

39	_____is the Base Class for All the Controls	Control Class	Base Class	ControlBase Class	BaseControl Class	Control Class
40	The default event of the TextBox is the _____	Click	TextChanged	KeyPress	GotFocus	TextChanged
41	The default event of Label is _____	Click	TextChanged	KeyPress	GotFocus	Click
42	The default event of ListBox is _____	IndexChanged	GotFocus	KeyPress	GotFocus	IndexChanged
43	_____ is a combination of a TextBox and a ListBox	ComboBox	ListBox	RichTextBox	LabelBox	ComboBox
44	_____method is used to clear the content of ListBox	Remove	Clear	Close	RemoveAt	Clear
45	_____ are those controls which contain other controls	GroupBox	Panel	Layout	Picture	Panel
46	_____ is used to delete specific item from the ListBox	RemoveAt	DeleteAt	Remove	ClearAt	RemoveAt
47	Which of The Control displays HyperLink	Label	LinkLabel	HyperLink Label	HyperText	LinkLabel

### UNIT IV

Objects and Collections: Understanding objects, properties, methods. Understanding collections. Files: Introduction – classification of files – processing files – handling files and folder using class – directory class – file class..
---

### Object Oriented Programming

Though VB2008 is very much similar to VB6 in terms of Interface and program structure, their underlying concepts are quite different. The main different is that VB2008 is a full Object Oriented Programming Language while VB6 may have OOP capabilities, it is not fully object oriented. In order to qualify as a fully object oriented programming language, it must have three core technologies namely encapsulation, inheritance and polymorphism. These three terms are explained below:

**Encapsulation** refers to the creation of self-contained modules that bind processing functions to the data. These user-defined data types are called classes. Each class contains data as well as a set of methods which manipulate the data. The data components of a class are called instance variables and one instance of a class is an object. For example, in a library system, a class could be member, and John and Sharon could be two instances (two objects) of the library class.

#### Inheritance

Classes are created according to hierarchies, and inheritance allows the structure and methods in one class to be passed down the hierarchy. That means less programming is required when adding functions to complex systems. If a step is added at the bottom of a hierarchy, then only the processing and data associated with that unique step needs to be added. Everything else about that step is inherited. The ability to reuse existing objects is considered a major advantage of object technology.

#### Polymorphism

Object-oriented programming allows procedures about objects to be created whose exact type is not known until runtime. For example, a screen cursor may change its shape from an arrow to a line depending on the program mode. The routine to move the cursor on screen in response to mouse movement would be written for "cursor," and polymorphism allows that cursor to take on whatever shape is required at runtime. It also allows new shapes to be easily integrated.

VB6 is not a full OOP in the sense that it does not have inheritance capabilities although it can make use of some benefits of inheritance. However, VB2008 is a fully functional Object Oriented Programming Language, just like other OOP such as C++ and Java. It is different from the earlier versions of VB because it focuses more on the data itself while the previous versions focus more on the actions. Previous versions of VB are known as procedural or functional programming language. Some other procedural programming languages are C, Pascal and Fortran.

VB2008 allows users to write programs that break down into modules. These modules will represent the real-world objects and are known as classes or types. An object can be created out of a class and it is known as an instance of the class. A class can also comprise subclass. For example, apple tree is a

**Class: III B. Sc. [CT]**

**Course Name: .NET Programming**

**Course Code: 16CTU502A**

**Unit: IV (Objects and Collections)**

**Batch- 2016-2019**

*subclass* of the *plant* class and the apple in your backyard is an instance of the apple tree class. Another example is student class is a subclass of the human class while your son John is an instance of the student class.

A class consists of data members as well as methods. In VB2008, the program structure to define a Human class can be written as follows:

Public Class Human

    'Data Members

    Private Name As String

    Private Birthdate As String

    Private Gender As String

    Private Age As Integer

    'Methods

    Overridable Sub ShowInfo( )

        MessageBox.Show(Name)

        MessageBox.Show(Birthdate)

        MessageBox.Show(Gender)

        MessageBox.Show(Age)

    End Sub

End Class;j

After you have created the human class, you can create a subclass that inherits the attributes or data from the human class. For example, you can create a students class that is a subclass of the human class. Under the student class, you don't have to define any data fields that are already defined under the human class, you only have to define the data fields that are different from an instance of the human class. For example, you may want to include StudentID and Address in the student class. The program code for the StudentClass is as follows:

Public Class Students

    Inherits Human

    Public StudentID as String

    Public Address As String

    Overrides Sub ShowInfo( )

        MessageBox.Show(Name)

        MessageBox.Show(StudentID)

        MessageBox.Show(Birthdate)

MessageBox.Show(Gender)

MessageBox.Show(Age)

MessageBox.Show(Address)

End Sub

We will discuss more on OOP in later lessons. In the next lesson, we will start learning simple programming techniques in VB2008

## VB.Net – COLLECTIONS

Collection classes are specialized classes for data storage and retrieval. These classes provide support for stacks, queues, lists, and hash tables. Most collection classes implement the same interfaces.

Collection classes serve various purposes, such as allocating memory dynamically to elements and accessing a list of items on the basis of an index, etc. These classes create collections of objects of the Object class, which is the base class for all data types in VB.Net.

### *Various Collection Classes and Their Usage*

The following are the various commonly used classes of the **System.Collection** namespace. Click the following links to check their details.

Class	Description and Usage
<a href="#"><u>ArrayList</u></a>	<p>It represents ordered collection of an object that can be <b>indexed</b> individually.</p> <p>It is basically an alternative to an array. However, unlike array, you can add and remove items from a list at a specified position using an <b>index</b> and the array resizes itself automatically. It also allows dynamic memory allocation, add, search and sort items in the list.</p>
<a href="#"><u>Hashtable</u></a>	<p>It uses a <b>key</b> to access the elements in the collection.</p> <p>A hash table is used when you need to access elements by using key, and you can identify a useful key value. Each item in the hash table has a <b>key/value</b> pair. The key is used to access the items in the collection.</p>
<a href="#"><u>SortedList</u></a>	<p>It uses a <b>key</b> as well as an <b>index</b> to access the items in a list.</p> <p>A sorted list is a combination of an array and a hash table. It contains a list of items that can be accessed using a key or an index. If you access items using an index, it is an ArrayList, and if you access items using a key, it is a Hashtable. The collection of items is always sorted by the key value.</p>

**Class: III B. Sc. [CT]**

**Course Name: .NET Programming**

**Course Code: 16CTU502A**

**Unit: IV (Objects and Collections)**

**Batch- 2016-2019**

<u>Stack</u>	It represents a <b>last-in, first out</b> collection of object. It is used when you need a last-in, first-out access of items. When you add an item in the list, it is called <b>pushing</b> the item, and when you remove it, it is called <b>popping</b> the item.
<u>Queue</u>	It represents a <b>first-in, first out</b> collection of object. It is used when you need a first-in, first-out access of items. When you add an item in the list, it is called <b>enqueue</b> , and when you remove an item, it is called <b>dequeue</b> .
<u>BitArray</u>	It represents an array of the <b>binary representation</b> using the values 1 and 0. It is used when you need to store the bits but do not know the number of bits in advance. You can access items from the BitArray collection by using an <b>integer index</b> , which starts from zero.

### **ArrayList Class**

It represents an ordered collection of an object that can be indexed individually. It is basically an alternative to an array. However, unlike array, you can add and remove items from a list at a specified position using an **index** and the array resizes itself automatically. It also allows dynamic memory allocation, adding, searching and sorting items in the list.

#### *Properties and Methods of the ArrayList Class*

The following table lists some of the commonly used **properties** of the **ArrayList** class:

<b>Property</b>	<b>Description</b>
Capacity	Gets or sets the number of elements that the ArrayList can contain.
Count	Gets the number of elements actually contained in the ArrayList.
IsFixedSize	Gets a value indicating whether the ArrayList has a fixed size.
IsReadOnly	Gets a value indicating whether the ArrayList is read-only.
Item	Gets or sets the element at the specified index.

The following table lists some of the commonly used **methods** of the **ArrayList** class:

<b>S.N.</b>	<b>Method Name &amp; Purpose</b>
1	<b>Public Overridable Function Add ( value As Object ) As Integer</b> Adds an object to the end of the ArrayList.
2	<b>Public Overridable Sub AddRange ( c As ICollection )</b> Adds the elements of an ICollection to the end of the ArrayList.
3	<b>Public Overridable Sub Clear</b> Removes all elements from the ArrayList.



4	<b>Public Overridable Function Contains ( item As Object ) As Boolean</b> Determines whether an element is in the ArrayList.
5	<b>Public Overridable Function GetRange ( index As Integer, count As Integer ) As ArrayList</b> Returns an ArrayList, which represents a subset of the elements in the source ArrayList.
6	<b>Public Overridable Function IndexOf ( value As Object ) As Integer</b> Returns the zero-based index of the first occurrence of a value in the ArrayList or in a portion of it.
7	<b>Public Overridable Sub Insert ( index As Integer, value As Object )</b> Inserts an element into the ArrayList at the specified index.
8	<b>Public Overridable Sub InsertRange ( index As Integer, c As ICollection )</b> Inserts the elements of a collection into the ArrayList at the specified index.
9	<b>Public Overridable Sub Remove ( obj As Object )</b> Removes the first occurrence of a specific object from the ArrayList.
10	<b>Public Overridable Sub RemoveAt ( index As Integer )</b> Removes the element at the specified index of the ArrayList.
11	<b>Public Overridable Sub RemoveRange ( index As Integer, count As Integer )</b> Removes a range of elements from the ArrayList.
12	<b>Public Overridable Sub Reverse</b> Reverses the order of the elements in the ArrayList.
13	<b>Public Overridable Sub SetRange ( index As Integer, c As ICollection )</b> Copies the elements of a collection over a range of elements in the ArrayList.
14	<b>Public Overridable Sub Sort</b> Sorts the elements in the ArrayList.
15	<b>Public Overridable Sub TrimToSize</b> Sets the capacity to the actual number of elements in the ArrayList.

*Example:*

The following example demonstrates the concept:

```
Sub Main()
    Dim al As ArrayList = New ArrayList()
    Dim i As Integer
    Console.WriteLine("Adding some numbers:")
    al.Add(45)
```

```
al.Add(78)
al.Add(33)
al.Add(56)
al.Add(12)
al.Add(23)
al.Add(9)
Console.WriteLine("Capacity: {0} ", al.Capacity)
Console.WriteLine("Count: {0}", al.Count)
Console.Write("Content: ")
For Each i In al
    Console.Write("{0} ", i)
Next i
Console.WriteLine()
Console.Write("Sorted Content: ")
al.Sort()
For Each i In al
    Console.Write("{0} ", i)
Next i
Console.WriteLine()
Console.ReadKey()
End Sub
End Module
```

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

```
Adding some numbers:
Capacity: 8
Count: 7
Content: 45 78 33 56 12 23 9
Content: 9 12 23 33 45 56 78
```

## Hashtable Class

The Hashtable class represents a collection of **key-and-value pairs** that are organized based on the hash code of the key. It uses the key to access the elements in the collection.

A hashtable is used when you need to access elements by using **key**, and you can identify a useful key value. Each item in the hashtable has a key/value pair. The key is used to access the items in the collection.

### *Properties and Methods of the Hashtable Class*

The following table lists some of the commonly used **properties** of the **Hashtable** class:

Property	Description
----------	-------------

**Class: III B. Sc. [CT]**

**Course Name: .NET Programming**

**Course Code: 16CTU502A**

**Unit: IV (Objects and Collections)**

**Batch- 2016-2019**

Count	Gets the number of key-and-value pairs contained in the Hashtable.
IsFixedSize	Gets a value indicating whether the Hashtable has a fixed size.
IsReadOnly	Gets a value indicating whether the Hashtable is read-only.
Item	Gets or sets the value associated with the specified key.
Keys	Gets an ICollection containing the keys in the Hashtable.
Values	Gets an ICollection containing the values in the Hashtable.

The following table lists some of the commonly used **methods** of the **Hashtable** class:

S.N	Method Name & Purpose
1	<b>Public Overridable Sub Add ( key As Object, value As Object )</b> Adds an element with the specified key and value into the Hashtable.
2	<b>Public Overridable Sub Clear</b> Removes all elements from the Hashtable.
3	<b>Public Overridable Function ContainsKey ( key As Object ) As Boolean</b> Determines whether the Hashtable contains a specific key.
4	<b>Public Overridable Function ContainsValue ( value As Object ) As Boolean</b> Determines whether the Hashtable contains a specific value.
5	<b>Public Overridable Sub Remove ( key As Object )</b> Removes the element with the specified key from the Hashtable.

*Example:*

The following example demonstrates the concept:

Module collections

Sub Main()

Dim ht As Hashtable = New Hashtable()

Dim k As String

ht.Add("001", "Zara Ali")

ht.Add("002", "Abida Rehman")

ht.Add("003", "Joe Holzner")

ht.Add("004", "Mausam Benazir Nur")

ht.Add("005", "M. Amlan")

ht.Add("006", "M. Arif")

ht.Add("007", "Ritesh Saikia")

If (ht.ContainsValue("Nuha Ali")) Then

Console.WriteLine("This student name is already in the list")

```
Else
    ht.Add("008", "Nuha Ali")
End If
' Get a collection of the keys.
Dim key As ICollection = ht.Keys
For Each k In key
    Console.WriteLine(" {0} : {1}", k, ht(k))
Next k
Console.ReadKey()
End Sub
End Module
```

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

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

### Stack Class

It represents a last-in, first-out collection of objects. It is used when you need a last-in, first-out access of items. When you add an item in the list, it is called pushing the item, and when you remove it, it is called popping the item.

#### *Properties and Methods of the Stack Class*

The following table lists some of the commonly used **properties** of the **Stack** class:

Property	Description
Count	Gets the number of elements contained in the Stack.

The following table lists some of the commonly used **methods** of the **Stack** class:

S.N	Method Name & Purpose
1	<b>Public Overridable Sub Clear</b> Removes all elements from the Stack.
2	<b>Public Overridable Function Contains ( obj As Object ) As Boolean</b> Determines whether an element is in the Stack.

**Class: III B. Sc. [CT]****Course Name: .NET Programming****Course Code: 16CTU502A****Unit: IV (Objects and Collections)****Batch- 2016-2019**

---

3	<b>Public Overridable Function Peek As Object</b> Returns the object at the top of the Stack without removing it.
4	<b>Public Overridable Function Pop As Object</b> Removes and returns the object at the top of the Stack.
5	<b>Public Overridable Sub Push ( obj As Object )</b> Inserts an object at the top of the Stack.
6	<b>Public Overridable Function ToArray As Object()</b> Copies the Stack to a new array.

*Example:*

The following example demonstrates use of Stack:

Module collections

```
Sub Main()  
    Dim st As Stack = New Stack()  
    st.Push("A")  
    st.Push("M")  
    st.Push("G")  
    st.Push("W")  
    Console.WriteLine("Current stack: ")  
    Dim c As Char  
    For Each c In st  
        Console.Write(c + " ")  
    Next c  
    Console.WriteLine()  
    st.Push("V")  
    st.Push("H")  
    Console.WriteLine("The next poppable value in stack: {0}", st.Peek())  
    Console.WriteLine("Current stack: ")  
    For Each c In st  
        Console.Write(c + " ")  
    Next c  
    Console.WriteLine()  
    Console.WriteLine("Removing values ")  
    st.Pop()  
    st.Pop()  
    st.Pop()  
    Console.WriteLine("Current stack: ")  
    For Each c In st  
        Console.Write(c + " ")  
    Next c  
    Console.ReadKey()
```

End Sub  
End Module

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

```
Current stack:
W G M A
The next poppable value in stack: H
Current stack:
H V W G M A
Removing values
Current stack:
G M A
```

### Queue

It represents a first-in, first-out collection of object. It is used when you need a first-in, first-out access of items. When you add an item in the list, it is called **enqueue**, and when you remove an item, it is called **dequeue**.

#### *Properties and Methods of the Queue Class*

The following table lists some of the commonly used **properties** of the **Queue** class:

Property	Description
Count	Gets the number of elements contained in the Queue.

The following table lists some of the commonly used **methods** of the **Queue** class:

S.N	Method Name & Purpose
1	<b>Public Overridable Sub Clear</b> Removes all elements from the Queue.
2	<b>Public Overridable Function Contains ( obj As Object ) As Boolean</b> Determines whether an element is in the Queue.
3	<b>Public Overridable Function Dequeue As Object</b> Removes and returns the object at the beginning of the Queue.
4	<b>Public Overridable Sub Enqueue ( obj As Object )</b> Adds an object to the end of the Queue.
5	<b>Public Overridable Function ToArray As Object()</b>

	Copies the Queue to a new array.
6	<b>Public Overridable Sub TrimToSize</b> Sets the capacity to the actual number of elements in the Queue.

*Example:*

The following example demonstrates use of Queue:

```
Module collections
Sub Main()
    Dim q As Queue = New Queue()
    q.Enqueue("A")
    q.Enqueue("M")
    q.Enqueue("G")
    q.Enqueue("W")
    Console.WriteLine("Current queue: ")
    Dim c As Char
    For Each c In q
        Console.Write(c + " ")
    Next c
    Console.WriteLine()
    q.Enqueue("V")
    q.Enqueue("H")
    Console.WriteLine("Current queue: ")
    For Each c In q
        Console.Write(c + " ")
    Next c
    Console.WriteLine()
    Console.WriteLine("Removing some values ")
    Dim ch As Char
    ch = q.Dequeue()
    Console.WriteLine("The removed value: {0}", ch)
    ch = q.Dequeue()
    Console.WriteLine("The removed value: {0}", ch)
    Console.ReadKey()
End Sub
End Module
```

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

```
Current queue:
A M G W
Current queue:
A M G W V H
```



**Class: III B. Sc. [CT]**

**Course Name: .NET Programming**

**Course Code: 16CTU502A**

**Unit: IV (Objects and Collections)**

**Batch- 2016-2019**

Removing some values

The removed value: A

The removed value: M

KARPAGAM

## WORKING WITH FILES

File handling in Visual Basic is based on System.IO namespace with a class library that supports string, character and file manipulation. These classes contain properties, methods and events for creating, copying, moving, and deleting files. Since both strings and numeric data types are supported, they also allow us to incorporate data types in files. The most commonly used classes are FileStream, BinaryReader, BinaryWriter, StreamReader and StreamWriter.

### CLASSIFICATION OF FILES

In Visual Basic .NET, files can be classification into three categories on the basic of their mode of access, as follows:

#### 1. Sequential access:

The sequential access mode is used for accessing a text file. A text file consists of a long chain of ASCII characters.

Generally, the data in the text file is accessed line by line or paragraph by paragraph.

#### 2. Binary access:

The binary access mode is used for accessing a binary file. Generally, a non-text file is called as binary file. For example, an image file is a binary file.

#### 3. Random access:

The random access mode is used for accessing a file that consists of records. In the random files, you can read or write any particular record without disturbing other records. For medium and large-size databases you must use the ADO.NET that comes with Visual Basic.NET.

### GENERIC PROCEDURE OF PROCESSING FILES

In Visual Basic the user can read files, create files, write to the files, modify files, and also delete files. The generic procedure is as follows:

- ❖ Open the file.
- ❖ Now process the file as per your requirement.
- ❖ Close the file.

### HANDLING FILES AND FOLDERS USING FUNCTIONS

The various functions that handle the files and folders are as follows:

- Kill ( ),
- File Data Time ( ),
- File Len ( ),
- Mkdir ( ),
- Rmdir ( ),

Class: III B. Sc. [CT]

Course Name: .NET Programming

Course Code: 16CTU502A

Unit: IV (Objects and Collections)

Batch- 2016-2019

---

- ChDir ( ),
- ChDrive ( ),
- CurDir ( ),
- Dir( ),
- FileCopy ( ),
- Rename ( ).

#### Functions

- **Kill (filename with path)**

This function deletes the file or files from the disk and not moved to the Recycle Bin. The use of wild cards (\* and ?) is permitted in this function. For example, notice the line of code given below:

```
Kill ("c:\files\satara.txt")  
Kill ("c:\files\*.txt")
```

- **FileDateTime (filename with path)**

This function returns the date and time of creation of the file.

```
Debug.WriteLine("Date & Time:"&(FileDateTime(("c:\files\satara.txt"))
```

After the execution, display the following line of the text in the Debug window:

```
Date & Time: 14/3/2014 4:02:44 PM
```

- **FileLen (filename with path)**

This function returns the size of the file in bytes.

```
Debug.WriteLine("Size of file in byte : "& FileLen (("c:\files\satara.txt"))
```

After execution displays the following line of text in the Debug window:

```
Size of file in bytes: 81
```

- **MkDir ( Foldername with path):**

This function creates a new folder or folders

```
MkDir("C:\myfiles")  
MkDir("C:\myfiles\stock")
```

The user can create comfortably create a number of folders in a single line of the code.

```
MkDir("C:\Mina\Lina\Bina")
```

- **Rmdir (foldername with path)**

This function deletes the folder. Only one folder can be deleted in a single function call. You cannot use wild card characters (\* and ?) in argument. The folder to be deleted must be empty

Rmdir("c:\files\stock") removes the folder stock successfully  
Rmdir("c:\files\stock") error occurs as folder stock doesn't exist

- **ChDir(foldername with path)**

This function sets the folder as current.

ChDir("C:\Files")

Once a given folder becomes current, you can handle the file in that folder simply by mentioning their file names (you are not requiring to write the full path of that file). For example, suppose the folder C:\Files contains many files and you want to delete two files in the folder, namely: satara.txt and London.doc. You can delete these files by using the line of codes given below:

ChDir("C:\File") Now folder C:\Files is the current folder  
Kill("satara.txt") File satara.txt in the current folder is deleted  
Kill("London.doc") File London.doc in the current folder is deleted.

- **ChDrive(drivename)**

This function sets the drive as current.

ChDrive("A")

After execution of this line of code, drive A becomes the current drive.

- **CurDir ( )**

This function returns the name of the current directory.

Debug.WriteLine("current directory is : "&CurDir ( ) )

After execution of this line of code, the following line of text is displayed in the Debug window:

Current directory is: A:\

- **Dir (filename or folder name with path)**

This function is used to check the existence of the file or folder. If that file or folder exists, then this function returns the name of that file or folder. If that file or folder doesn't exist, then this function returns the empty string.

strCheck = Dir("C:\Files\satara.txt")  
if strCheck = "satara.txt" Then

```
Debug.WriteLine("The file satara.txt exists.")
```

```
Else
```

```
Debug.WriteLine("The file satara.txt doesn't exists.")
```

```
End if
```

Assuming that file satara.txt really exist in the folder C:\Files, the piece of code, after execution displays the following line of text in the Debug window:

```
File satara.txt exist.
```

- **FileCopy (SourceFile, DestinationFile)**

This functions copies the given files and place it in the desired destination.

```
FileCopy("C:\Files\satara.txt", "C:\My files\city.txt")
```

- **Rename (oldName ,NewName)**

This is functions renames the file or folder.

```
Rename ("C:\My Files", "C:\ourfiles")
```

```
Rename ("C:\Fiels\satara.txt", "C:\Files\comport.txt")
```

```
Rename ( "C:\Files\comport.txt", "C"\Files\nice.txt")
```

Let the folder C:\Files contains the file satara.txt and sangli.txt. now if you try to renames satara.txt as sangli.txt then run-time error occurs. Notice the line of the code give below:

```
Rename("C:\Files\satara.txt", "C:\Files\sangli.txt") error!!!
```

## HANDLING FILES AND FOLDERS USING CLASSES

Visual Basic.NET allows handling the files and folders by using class.

- Directory Class
- File Class

In order to use these classes in a program, type the following statement at the top of the code window:

```
Imports System.IO
```

### Directory Class

Directory class offers a number of methods to handle folders.

```
Exist ( ), CreateDirectory ( ), Delete ( ), SetCurrentDirectory ( ), GetCurrentDirectory ( ),  
GetDirectories ( ), GetDirectoryRoot ( ),GetFiles ( ), GetFileSystemEntries ( ),  
GetLastAccessTime ( ), SetLastAccessTime ( ),and GetLogicalDrives ( ).
```

### Exists(foldername with path)

**Class: III B. Sc. [CT]**

**Course Name: .NET Programming**

**Course Code: 16CTU502A**

**Unit: IV (Objects and Collections)**

**Batch- 2016-2019**

---

This method checks whether the folder exists. If the folder exists, then method returns the value true; otherwise, it returns the value false. For example notice the piece of code given below:

```
If Directory.Exists("C:\MyFiles") Then
    Debug.WriteLine("Folder C:\MyFiles exist.")
Else
    Debug.WriteLine("Folder C:\MyFiles exist.")
End If
```

**CreateDirectory(foldername with path)**

This method creates the folder or nested folders.

```
Directory.CreateDirectory("C:\MyFiles\stock\Reserve")
Directory.CreateDirectory("C:\ExFiles")
```

**Delete (foldername with path,force)**

This method deletes the folder. This method also accepts a second argument (force) which is optional and its data type is Boolean. It means that the possible values of force are True and False. If force is True, then a non-empty folder is also deleted and if force is False, then only an empty try to delete the non-empty folder, then run-time error occurs.

```
Directory.Delete ("C:\BigFolder")           Folder is empty and gets deleted
Directory.Delete ("C:\JamboFolder")         Folder is non-empty, run-time error occurs
Directory.Delete ("C:\ JamboFolder",True)   Folder is non-empty and gets deleted.
```

**SetCurrentDirectory(Foldername with path)**

This method set the folder as current.

```
Directory.SetCurrentDirectory("C:\myfiles")
```

**GetDirectories (Foldername with path)**

This method returns the name of all the folders in the said folder as String data type

```
strArray=Directory.GetDirectories("C:\Myfiles")
For Each strText in strArray
    Debug.WriteLine(strtext)
Next
```

**GetDirectoryRoot( path)**

This method returns the root part of the path.

```
strText=Directory.GetDirectoryRoot ("C:\Myfiles")
```

After execution, the string "C:\:" is assigned to the string variable strText.

**GetFiles( foldername with path)**

This method returns the names of all the files in the said folders

```
strArray=Directory.GetFiles("C:\Myfiles")
For Each strText in strArray
    Debug.WriteLine(strtext)
Next
```

**GetFileSystemEntries( Foldername with path)**

Class: III B. Sc. [CT]

Course Name: .NET Programming

Course Code: 16CTU502A

Unit: IV (Objects and Collections)

Batch- 2016-2019

---

This method returns the names of all the files and folder in the said folder.

```
strArray = Directory. GetFileSystemEntries("C:\MyFiles")
```

```
For Each strText in strArray
```

```
    Debug.WriteLine(strtext)
```

```
Next
```

For each .. next loop that span lines 2,3 and 4, retrieve these strings and display them in the debug window, as shown below:

```
C:\MyFiles\HisFiles
```

```
C:\ MyFiles\Stock
```

```
C:\ MyFiles\Satara.txt
```

```
C:\ MyFiles\Kolhapur.txt
```

#### **GetLastAccessTime (foldername with path)**

This method returns the date of the last access of the said folder.

```
dteDate = Directory. GetLastAccessTime("C:\MyFiles")
```

```
Debug.WriteLine("LastAccessTime of C:\MyFiles is :"&dteDate)
```

After execution, display the following line of text in the debug window:

```
LastAccessTime of C:\MyFiles is :14/3/2002
```

#### **SetLastAccessTime (Foldername with path, date)**

The user can set the last access time of a folder by using this method. Two arguments are passed to this method, the first argument (string type 0 is the name of the folder and the second argument (data type) is the date to be set.

```
Directory. SetLastAccessTime("C:\MyFiles", Now
```

After execution, set the current date as the last access time of the folder C:\MyFiles.

#### **GetLogicalDrives ( )**

This method returns the name of all the drives in your computer. For example, notice the piece of code given below

```
strArray = Directory. GetLogicalDrives ( )
```

```
For Each strText in strArray
```

```
    Debug.WriteLine(strtext)
```

```
Next
```

After execution, displays the following line of text in the debug window.

```
A:\
```

```
C:\
```

```
D:\
```

```
E:\
```



### File Class

File class offers a number of methods for handling files. We will discuss the following methods of file class in this section:

- Copy ( )
- Exist ( )
- Delete ( )
- GetCreationTime ( )
- SetCreationTime ( )
- GetLastAccessTime ( )
- SetLastAccessTime ( )

#### **Copy (source filename, destination filename, overwrite)**

This method creates a copy of the file. This method accepts three arguments, the first and second argument are required while the third argument is optional. The first argument (string type) is the name of the source file with path, the second argument ( string type) is the name of the destination file with path, and third argument (Boolean type ) takes one of the two values : true and false. If the value of third argument is false (or the third argument is omitted), then overwriting is not permitted and if the third argument is true, then overwriting is permitted

```
File.Copy("C:\MyFiles\satara.txt", C:\MyFiles\HisFiles\ satara.txt", True)
```

#### **Exist (filename with path)**

This method checks whether the file exist. This method returns the value true if the file exists, and false if the file doesn't exist.

```
If File.Exist("C:\MyFiles\satara.txt") Then
    Debug.WriteLine("File Satara.txt Exist.")
Else
    Debug.WriteLine("File Satara.txt doesn't Exist.")
End if
```

#### **Delete (filename with path)**

This method deletes the file. The deleted file is not sent to Recycle Bin and cannot be recovered.

```
If File.Exists ("C:\ MyFiles\satara.txt") Then
    File.delete ("C:\ MyFiles\His Files\satara.txt")
Endif
```

#### **GetCreationTime ( )**

This method returns the date and time of creation of the file.

```
dteDate = File.GetCreationTime ("C:\ MyFiles\satara.txt")
Debug.WriteLine("Creation time of satara.txt is : "&dteDate)
```

After the execution, displays the following line of text in the debug window.

Creation time of satara.txt is : 10/3/2012 4.33.22 PM

#### **SetCreationTime (filename with path,date)**

This method allows to set the creation time of file. The first argument (string type) is the name of the file with path and second argument (data type) is the date to be set.

```
File.Setcreation.Time ("C:\Myfiles\satara.txt",Now)
```

#### **GetLastAccessTime ( )**

This method returns the date of last access of file.

```
dteDate = File. GetLastAccessTime("C:\Myfiles\satara.txt")
```

```
Debug.WriteLine("Last access time of satara.txt is :."&dteDate)
```

After execution, displays the following line of text in the Debug window.

Last access time of satara.txt is : 10/12/2013

**SetLastAccessTime (filename with path,date)**

This method allows to set the last access time of the file.

```
File.SetLastAccessTime("C:\Myfiles\satara.txt", Now)
```

**FILE PROCESSING USING FUNCTIONS****Function****Freefile()**

The function returns a number that can be used as file number.

```
IntN1=FreeFile()
```

```
FileOpen (intN1, "C:\Files\Satara.txt",OpenMode.Input)
```

```
'-----generic code
```

```
intN2=FreeFile()
```

```
FileOpen (intN2, C:\Files\Sangli.txt", OpenMode.Input)
```

```
'-----generic code
```

```
FileClose (intN1)
```

```
'-----generic code
```

```
FileClose (intN2)
```

**FileOpen()**

The function is used for opening a file.

```
FileOpen(fileNumber,fileName,OpenMode [,access][,share][recordLen]
```

The first three arguments are required while the three remaining are optional. The various value of OpenMode are Input,Output,and Append are meant for sequence files,the value random is meant for random access files,and the value Binary is meant for binary file.Values of OpenMode Enumeration

Value	Description
Input	Sequential(text) file is opened for reading
Output	sequential(text) file is opened for writing
Append	Sequential(text)file is appending new text to the existing contents of the file
Random	File is opened in random mode
Binary	File is opened binary mode

**Class: III B. Sc. [CT]****Course Name: .NET Programming****Course Code: 16CTU502A****Unit: IV (Objects and Collections)****Batch- 2016-2019**

---

Values of OpenAccess Enumeration

Value	Description
Default	File is Opened for reading and writing .This is the default access
Read	File is Opened for reading only
Write	File is opened for writing only
ReadWrite	File is openedfor reading and writing.same as default

Values of OpenShare Enumeration

Value	Description
Default	other applications can share this file .This is default status.
Shared	other applications can share this file.same as default.
LockRead	other application cannot read this file.
LockWrite	other application cannot write to this file.
LockReadWrite	other application can neither read nor write to this file

FileOpen(1, "c:\File.dat", OpenMode.Random,OpenAccess.Read, OpenShare.LockWritee, 42)

**FileClose (fileNumber)**

This function closes the file when its file number is passed to this function.

FileClose(1) 'Line 1, file with file number 1 is now closed.

**Reset ()**

The function reset () closes all the files. It is equivalent to the FileCose() function without argument.

**EOF(fileNumber)**

Funtion EOF() returns the value true,if the end of the file is reached.Otherwise it returns the value false.

**LOF(fileNumber)**

Function LOF() returns the length of the filein bytes.

FileOpen(1, "C:\Files\satara.txt",OpenMode.Input)

Debug.WriteLine ("Length(size)of file in bytes:" &LOF(1))

After execution,displays the following lines of code in the debug window:

Length (size) of file in bytes:81

**Print(FileNumber,OutputData) and PrintLine (FileNumber,OutputData)**

These functions are used for writing data into sequential

FileOpen(1, "C:\Files\Mina.txt", OpenMode.Input)

Print(1, "Mina", "Learn" , "VisualBasic")

C:\Files>TYPE MINA.TXT

Mina

Learn

Visual Basic

FileOpen(1,"C:\Files\Lina.txt",OpenMode.Output)

'Line1

**Class: III B. Sc. [CT]****Course Name: .NET Programming****Course Code: 16CTU502A****Unit: IV (Objects and Collections)****Batch- 2016-2019**

---

Print(1,"Lina")	'Line2
Print(1,"Learns")	'Line3
Print(1, "VisualBasic")	'Line4
FileOpen(2,"C:\Files\Mina.txt",OpenMode.Output)	'Line5
PrintLine(2,"Mina")	'Line6
PrintLine(2,"learns")	'Line7
PrintLine(2,"Visual Basic")	'Line8

Open the command Prompt window and view the content of the file Lina.txt(written using the Print

C:\Files>TYPE LINA.TXT

Lina learns visual Basic

C:\Files>

The content of file Mina.txt are shown below:

C:\files>TYPE MINA.TXT

Mina

Learns

Visual Basic

C:\Files>

### **Input(FileNumber,Variable)**

This function is used for reading a sequential file. It reads a data from a file(the file number of which is passed to this function as first arguments) and assign the data to the variable that is passed to this function as second argument.

FileOpen(1,"C:\Files\Lina.txt",OpenMode.Input)

'Line1

Input(1,strText)

'Line2

Debug.WriteLine("Contents of strText: &strText)

'Line3

After execution, displays the following line of text in the debug window:

Content of the strText : Lina learns Visual Basic

### **LineInput(FileNumber)**

This function is used for reading a sequential file. It reads the contents of files and returns the text that is generally assigned to string variable.

FileOpen(1, "C:\Files\Lina.txt",OpenMode.Input)

strText=LineInput(1)

Debug.WriteLine("Contents of strText: "&strText)

After execution,displays the following lines of text in the Debug window.

Contents of strText: Lina Learns VisualBasic.

### **Functions filePut() and FileGet()**

These functions are used in random access file which are used to creating mini database. If you want to create a medium or large database, then use a ADO.NET and random- mode file are the most suitable for the task of creating mini database.

### **FilePut(FileNumber,Value [,RecordNumber])**

This function is used for writing a record to a random access file. Three arguments are passed to this function –the first arguments(required)is a file number, second arguments(required)is a value to be written in a record, and the third argument(optional)is a record number. if the third argument is omitted ,then the value is written to the current record.

Class: III B. Sc. [CT]

Course Name: .NET Programming

Course Code: 16CTU502A

Unit: IV (Objects and Collections)

Batch- 2016-2019

---

FilePut(1, "member")

**File Get (File Number, variable [, Record Number])**

This function is used for reading a record from a random-access file. Three arguments are passed to this function-the first argument (required) is a file number, the second argument(required) is a variable in which the record that is read from the file is stored, and the third argument (optional) is a record number. If the third argument is omitted, then the value is read from the current record.

FileGet(1, "member",1)

**FILE PROCESSING USING STREAMS**

There are four stream-based classes are available

1. StreamReader. Using the StreamReader object, you can read a text file.
2. StreamWriter. Using the StreamWriter object, you can write into a text file.
3. Binary Reader. Using the BinaryReader object, you can read a binary file.
4. BinaryWriter. Using the BinaryWriter object, you can write into a binary file.

**StreamReader and StreamWriter Class**

The StreamReader and StreamWriter classes enables us to read or write a sequential stream of characters to or from a file.

**BinaryReader and BinaryWriter Class**

The BinaryReader and BinaryWriter classes enable us to read and write binary data, raw 0's and 1's, the form in which data is stored on the computer.

**Using the StreamReader Class**

In order to understand the generic syntax of using the StreamReader

```
Dim myStream As StreamReader
MyStream=New StreamReader("C:/Files/Kolhapur.txt")
txtShirish.Text=myStream.ReadToEnd()
txtShirish. Select (0,0)
myStream.Close ()
```

**Using the StreamWriter Class**

In order to understand the generic syntax of using StreamWriter

```
Dim myStream As StreamWriter
StreamWriter ("C:/Files/Delhi.txt", False)
myStream.Write(txtShirish.Text)
myStream.Close ()
```

myStream=New

**Code to create a File**

**Class: III B. Sc. [CT]****Course Name: .NET Programming****Course Code: 16CTU502A****Unit: IV (Objects and Collections)****Batch- 2016-2019**

---

Imports System.IO

'Namespace required to be imported to work with files

Public Class Form1 Inherits System.Windows.Forms.Form

Private Sub Form1\_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load

Dim fs as New FileStream("file.doc", FileMode.Create, FileAccess.Write)

'declaring a FileStream and creating a word document file named file with

'access mode of writing

Dim s as new StreamWriter(fs)

'creating a new StreamWriter and passing the filestream object fs as argument

s.BaseStream.Seek(0, SeekOrigin.End)

'the seek method is used to move the cursor to next position to avoid text to be

'overwritten

s.WriteLine("This is an example of using file handling concepts in VB .NET.")

s.WriteLine("This concept is interesting.")

'writing text to the newly created file

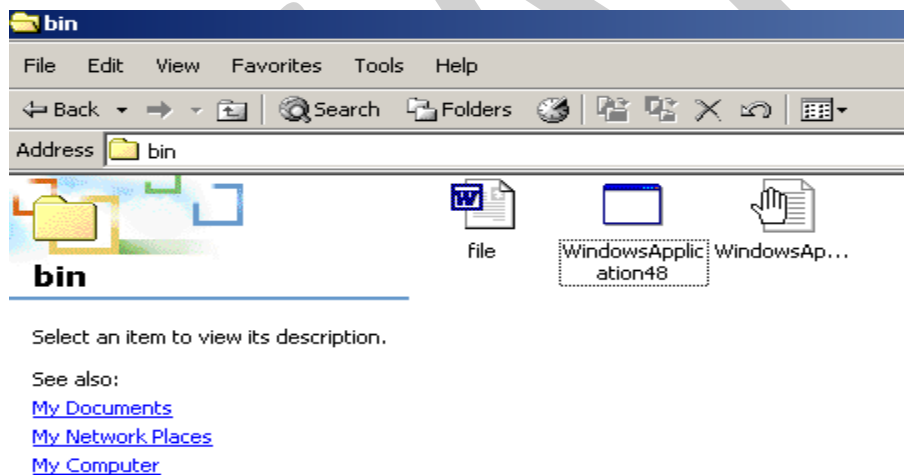
s.Close()

'closing the file

End Sub

End Class

The default location where the files we create are saved is the bin directory of the Windows Application with which we are working. The image below displays that.



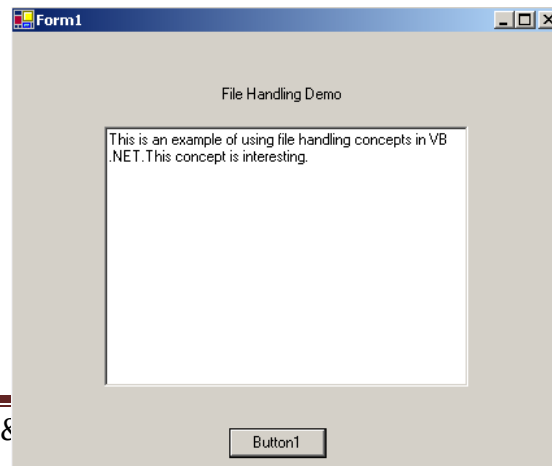
**Code to create a file and read from it**

Drag a Button and a RichTextBox control onto the form. Paste the following code which is shown below.

```
Imports System.IO
'Namespace required to be imported to work with files
Public Class Form1 Inherits System.Windows.Forms.Form
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Dim fs as New FileStream("file.doc", FileMode.Create, FileAccess.Write)
'declaring a FileStream and creating a document file named file with
'access mode of writing
Dim s as new StreamWriter(fs)
'creating a new StreamWriter and passing the filestream object fs as argument
s.WriteLine("This is an example of using file handling concepts in VB .NET.")
s.WriteLine("This concept is interesting.")
'writing text to the newly created file
s.Close()
'closing the file

fs=New FileStream("file.doc",FileMode.Open,FileAccess.Read)
'declaring a FileStream to open the file named file.doc with access mode of reading
Dim d as new StreamReader(fs)
'creating a new StreamReader and passing the filestream object fs as argument
d.BaseStream.Seek(0,SeekOrigin.Begin)
'Seek method is used to move the cursor to different positions in a file, in this code, to
'the beginning
while d.Peek() > -1
'peek method of StreamReader object tells how much more data is left in the file
RichTextbox1.Text &= d.ReadLine()
'displaying text from doc file in the RichTextBox
End while
d.close()
End Sub
```

The image below displays output of the above



code.



**KARPAGAM ACADEMY OF HIGHER EDUCATION**  
**COIMBATORE - 21**

**DEPARTMENT OF COMPUTER SCIENCE, CA & IT**

**CLASS : III B.Sc COMPUTER TECHNOLOGY**

**BATCH : 2016-2019**

**Part -A Online Examinations**

**I One Marks**

**SUBJECT: . NET Programming**

**SUBJECT CODE: 16CTU502A**

**UNIT-IV**

S.No.	Question	Option 1	Option 2	Option 3	Option 4	Answer
1	Last In First Out is called as -----	Stack	Queue	Hash	ArrayList	Stack
2	First In First Out is called as -----	Stack	ArrayList	Queue	ArrayList	Queue
3	Removing an item from queue is -----	Enqueue	Insert()	deleting	Dequeue	Dequeue
4	A _____ is a combination of an array and a hash table.	Stack	Queue	Sorted List	ArrayList	Sorted List
5	_____ Property Gets or sets the number of elements that the ArrayList can contain	Capacity	Count	Item	Set	Capacity
6	_____ Property Gets or sets the element at the specified index.	Capacity	Count	Item	Set	Item
7	The method used to Determines whether an element is in the ArrayList.	Contains	Item	Available	Capacity	Contains
8	_____Returns an ArrayList, which represents a subset of the elements in the source ArrayList.	Set Range	Get Range	Index of	Index Range	Get Range
9	Which methods an element with the specified key and value into the Hashtable.	Add()	Insert()	Insertat()	addat()	Add()
10	_____ Determines whether the Hashtable contains a specific key.	Contains Key	Contains value	ContainsI ndex	ContainsI tem	Contains Key
11	Which Method Inserts an object at the top of the Stack.	Push()	Insert()	Pop()	Add()	Push()

12	_____ Adds an object to the end of the Queue.	Enqueue	Dequeue	Insert	Insertat	Enqueue
13	which Method Sets the capacity to the actual number of elements in the Queue.	LTrimTo Size	RTrimTo Size	TrimToSize	STrimTo Size	TrimToSize
14	_____ Method Removes all elements from the Queue.	Clear	Remove	Cls	Removeat	Clear
15	Which method is used to print the total elements in the list	capacity	count	total	sum	count
16	Adding items to the queue is -----	Enqueue	Insert()	deleting	Dequeue	Enqueue
17	----- can hold more than one value for a corresponding Key	HashTabl e	ArrayList	NameVal ueCollect	Queue	NameVal ueCollect
18	How do you Create Constructors in VB.NET?	Create a method	Create a method	Create a method	Create a method	Create a method
19	How does VB.NET achieve polymorphism?	Encapsul ation	Main function	Abstract Class/Fu	Using Impleme	Abstract Class/Fu
20	How do you create method which can be accessed without the use of objects reference ?	Using FRIEND	Using STATIC	Using SHARE	Using PRIVAT	Using SHARE
21	VB.NET supports -----	encapsula tion	abstractio n	inheritan ce	all	all
22	A ----- is a conceptual representation of all entities that share common attributes and behaviors	specifier	member	object	class	class
23	In VB.Net one can create an abstract class using the ----- keyword	MustOve rride	MustInhe rit	Overrides	public	MustInhe rit
24	----- contains only the declaration of members	Encapsul ation	Inheritan ce	Interface	Polymorp hism	Inheritan ce
25	To create objects, first create its template called -----	Procedur e	Module	Class	Collectio n	Class
26	An OOP's concept that defines wrapping of code and data into a single unit is called -----	Abstracti on	Inheritan ce	Polymorp hism	Encapsul ation	Encapsul ation

27	To create a property that can be read but not changed by client code include ----- construct	only Get construct	only Set construct	Either Get or	Both Get and Set	only Get construct
28	If an Object variable is declared As Object, Which binding is used?	Early Binding	Static Binding	Data Binding	Late Binding	Late Binding
29	What is the best way to destroy an object reference?	Start Garbage	Set object varibale	Call the destructo	use dispose(o	Set object varibale
30	What object is used to manipulate files?	System.I O.Files	System.I O.Stream	Streams.I O.Files	System.I O.Directo	Streams.I O.Files
31	What object is used to manipulate directories?	System.I O.Files	System.I O.Stream	Streams.I O.Files	System.I O.Directo	System.I O.Directo
32	----- is used to rename a file	System.I O.Files.C	System.I O.Files.	System.I O.Files.R	System.I O.Files.C	System.I O.Files.
33	----- is a programming structure that encapsulates data and functionality as a single unit.	Class	Object	Collectio n	methods	Object
34	This is the way we refer to properties of an object in code	{Object name}. {P	{Class}. {Property}	{Class}. {Method}	{Object Name}. {	{Object name}. {P
35	A property that returns an object is called -----	Collectio n	subroutin e	Object Oriented	Object Property	Object Property
36	The process of creating an object is called -----	integratio n	instantiati on	interfacin g	inheritan ce	instantiati on
37	A _____ is a place to store the code we write	class	module	method	subroutin e	module
38	Codes written in _____ modules are always available and need not instantiate an object for it	class module	standard module	library modules	None of the above	standard module
39	We cannot create _____ based on standard modules	classes	methods	objects	propertie s	objects
40	Interfaces are similar to ----- classes	abstract	collection	inheritan ce	polymorp hism	abstract
41	How many types of constructors are there?	3	4	5	2	2

42	----- are the special methods that are used to release the instance of a class from memory	constructors	destructors	inheritance	abstract	destructors
43	The ----- method is called to release a resource such as a database connection	compose()	release()	destroy()	dispose()	dispose()
44	File handling in Visual Basic is based on ----- namespace	System.Input	System.Output	System.IO	System.Files	System.IO
45	How many access methods are there in file?	2	4	5	3	3
46	The ----- class provides access to files and file-related information	Stream	FileStream	StreamFile	FileMode	StreamFile
47	The FileStream class opens a file either in ----- mode	synchronous	asynchronous	synchronous or	sequential	synchronous or
48	The ----- method is used to open a file in synchronous mode	BeginRead()	Read()	BeginWrite()	Write()	Read(), Write()
49	The ----- method is used to open a file in asynchronous mode	Read()	BeginRead()	Write()	BeginWrite()	BeginRead(),
50	By default FileStream class opens file in ----- mode	synchronous	asynchronous	sequential	random	synchronous
51	The ----- class is used to read from binary file	StreamReader	StreamWriter	BinaryReader	BinaryWriter	BinaryReader
52	The ----- class is used to write to binary file	StreamReader	BinaryReader	BinaryWriter	StreamWriter	BinaryWriter
53	The ----- method is used to set the file pointer to the beginning of the file	Offset	Peek()	Seek()	SeekOrigin()	Seek()
54	The ----- method is used to read characters from the file	Read()	ReadChars()	ReadCharacters()	BinaryRead()	ReadChars()
55	The System.IO model also enables to work with drives and folders by using the ----- class	File	Directory	Reader	Stream	Directory
56	How many methods are most frequently used in Directory class	5	3	7	6	6

57	The ----- method is used to delete a directory and all its contents	remove	destroy	delete	dispose	delete
58	VB.Net run-time functions allow ----- types of file access	4	2	5	3	3
59	The functions that allow the types of file access are defined in the ----- namespace	System.I O	System.A ssemblies	System.I O.Files.R	System.I O.File	System.I O.File
60	The function that is used to retrieve the date and time when a file was created or modified is -----	Dir	FileCopy	FileDate Time	FreeFile	FileDate Time
61	The ----- function is used to retrieve a value specifying the current read/write position within an open file	GetAttr	FreeFile	FileDate Time	Loc	Loc
62	The ----- function is used to write data from a variable to a disk file	FreeFile	GetAttr	FilePut	Print	FilePut
63	The file I/O operations can be done in ----- ways	3	4	2	5	2
64	The ----- function is used to retrieve the next file number available for use by FileOpen() function	FreeFile	FileCopy	FileDate Time	FilePut	FreeFile
65	The ----- function is used to retrieve String value containing characters from a file opened in Input or Binary mode	PrintLine	InputStri ng	FilePut	Loc	InputStri ng
66	The function that allows to open a file in any access methods is -----	Open()	Read()	Create()	FileOpen ( )	FileOpen ( )

**UNIT V**

Database programming with ADO .Net: overview of ADO, from ADO to ADO .Net, accessing data using server explorer. Creating connection, command, data adapter and data set with OLEDB and SQLDB. Display data on data bound controls, display data on a data grid. Generate reports using Crystal Report Viewer.

**ADO .NET**

Most applications need data access at one point of time making it a crucial component when working with applications. Data access is making the application interact with a database, where all the data is stored. Different applications have different requirements for database access. VB .NET uses ADO .NET (Active X Data Object) as it's data access and manipulation protocol which also enables us to work with data on the Internet. Let's take a look why ADO .NET came into picture replacing ADO.

**Evolution of ADO.NET**

The first data access model, DAO (data access object) was created for local databases with the built-in Jet engine which had performance and functionality issues. Next came RDO (Remote Data Object) and ADO (ActiveX Data Object) which were designed for Client Server architectures but soon ADO took over RDO. ADO was a good architecture but as the language changes so is the technology. With ADO, all the data is contained in a recordset object which had problems when implemented on the network and penetrating firewalls.

ADO was a connected data access, which means that when a connection to the database is established the connection remains open until the application is closed. Leaving the connection open for the lifetime of the application raises concerns about database security and network traffic. Also, as databases are becoming increasingly important and as they are serving more people, a connected data access model makes us think about its productivity.

Example: an application with connected data access may do well when connected to two clients, the same may do poorly when connected to 10 and might be unusable when connected to 100 or more. Also, open database connections use system resources to a maximum extent making the system performance less effective.

**Why ADO.NET?**

To cope up with some of the problems mentioned above, ADO .NET came into existence. ADO .NET addresses the above mentioned problems by maintaining a disconnected database access model which means, when an application interacts with the database, the connection is opened to serve the request of the application and is closed as soon as the request is completed. Likewise, if a database is Updated, the connection is opened long enough to complete the Update operation and is closed. By keeping connections open for only a minimum period of time, ADO .NET conserves system resources and provides maximum security for databases and also has less impact on system performance. Also, ADO .NET when interacting with the database uses XML and converts all the data into XML format for database related operations making them more efficient.

**The ADO.NET Data Architecture**

Data Access in ADO.NET relies on two components: DataSet and Data Provider.

**DataSet**

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.

**Data Provider**

The Data Provider is responsible for providing and maintaining the connection to the database. A DataProvider is a set of related components that work together to provide data in an efficient and performance driven manner. The .NET Framework currently comes with two DataProviders: the SQL Data Provider which is designed only to work with Microsoft's SQL Server 7.0 or later and the OleDb DataProvider which allows us to connect to other types of databases like Access and Oracle. Each DataProvider consists of the following component classes:

1. The Connection object which provides a connection to the database
2. The Command object which is used to execute a command
3. The DataReader object which provides a forward-only, read only, connected recordset
4. The DataAdapter object which populates a disconnected DataSet with data and performs update

Data access with ADO.NET can be summarized as follows:

1. A connection object establishes the connection for the application with the database.
2. The command object provides direct execution of the command to the database. If the command returns more than a single value, the command object returns a DataReader to provide the data. Alternatively, the DataAdapter can be used to fill the Dataset object. The database can be updated using the command object or the DataAdapter.

**Component classes that make up the Data Providers****The Connection Object**

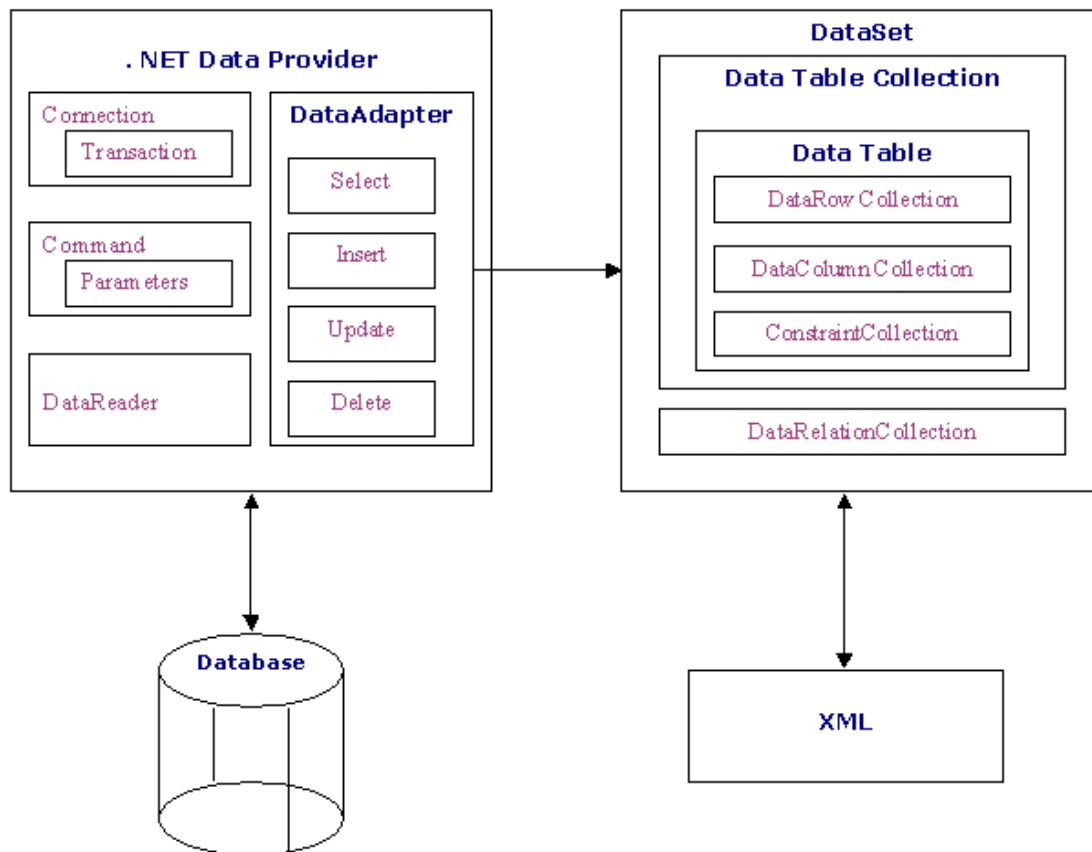
The Connection object creates the connection to the database. Microsoft Visual Studio .NET provides two types of Connection classes: the SqlConnection object, which is designed specifically to connect to Microsoft SQL Server 7.0 or later, and the OleDbConnection object, which can provide connections to a wide range of database types like Microsoft Access and Oracle. The Connection object contains all of the information required to open a connection to the database.

**The Command Object**

The Command object is represented by two corresponding classes: SqlCommand and OleDbCommand. Command objects are used to execute commands to a database across a data connection. The Command objects can be used to execute stored procedures on the database, SQL



commands, or return complete tables directly. Command objects provide three methods that are used to execute commands on the database:



**ADO .NET Data Architecture**

1. **ExecuteNonQuery**: Executes commands that have no return values such as INSERT, UPDATE or DELETE
2. **ExecuteScalar**: Returns a single value from a database query
3. **ExecuteReader**: Returns a result set by way of a DataReader object

### **The DataReader Object**

The DataReader object provides a forward-only, read-only, connected stream recordset from a database. Unlike other components of the Data Provider, DataReader objects cannot be directly instantiated. Rather, the DataReader is returned as the result of the Command object's ExecuteReader method. The SqlCommand.ExecuteReader method returns a SqlDataReader object, and the OleDbCommand.ExecuteReader method returns an OleDbDataReader object. The DataReader can provide rows of data directly to application logic when you do not need to keep the data cached in memory. Because only one row is in memory at a time, the DataReader provides the

lowest overhead in terms of system performance but requires the exclusive use of an open Connection object for the lifetime of the DataReader.

### **The DataAdapter Object**

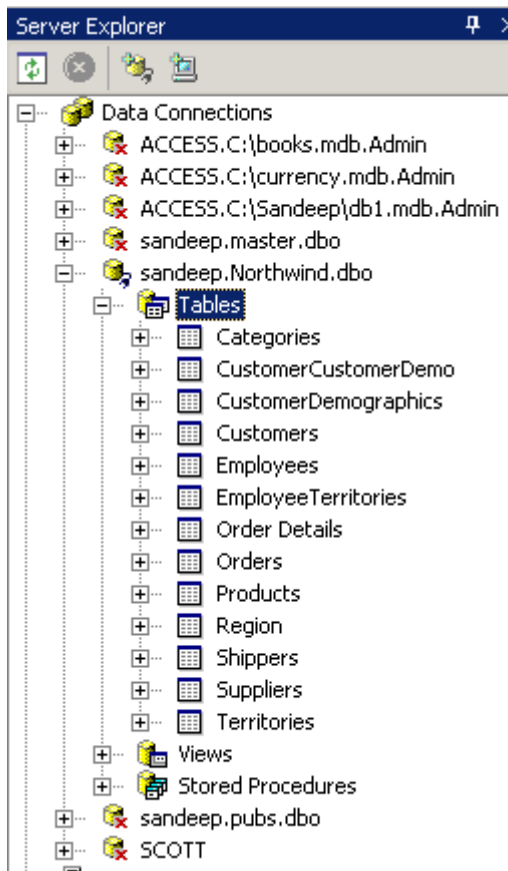
The DataAdapter is the class at the core of ADO .NET's disconnected data access. It is essentially the middleman facilitating all communication between the database and a DataSet. The DataAdapter is used either to fill a DataTable or DataSet with data from the database with its Fill method. After the memory-resident data has been manipulated, the DataAdapter can commit the changes to the database by calling the Update method. The DataAdapter provides four properties that represent database commands:

1. SelectCommand
2. InsertCommand
3. DeleteCommand
4. UpdateCommand

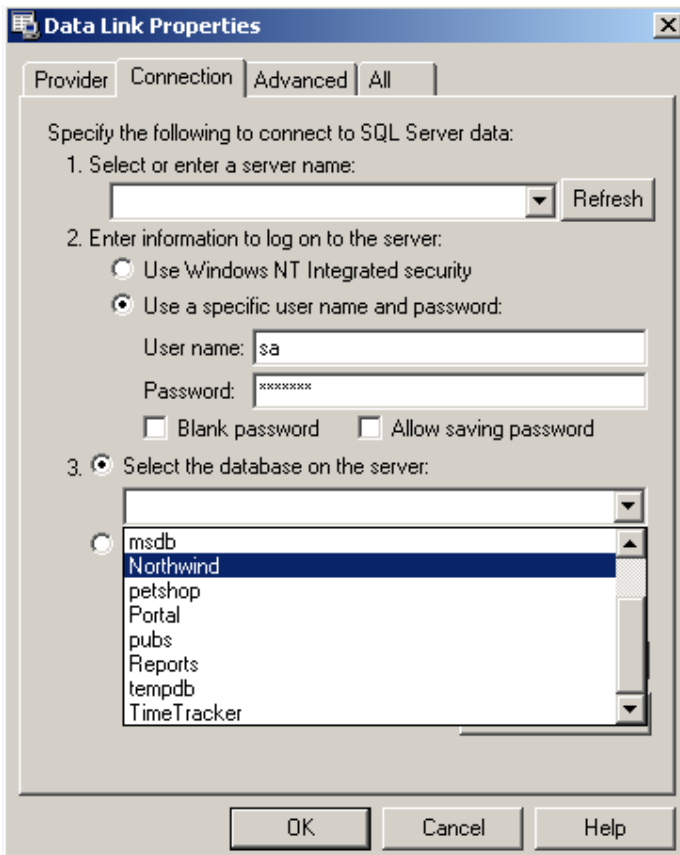
When the Update method is called, changes in the DataSet are copied back to the database and the appropriate InsertCommand, DeleteCommand, or UpdateCommand is executed.

### **Data Access with Server Explorer**

Visual Basic allows us to work with databases in two ways, visually and code. In Visual Basic, Server Explorer allows us to work with connections across different data sources visually. Lets see how we can do that with Server Explorer. Server Explorer can be viewed by selecting View->Server Explorer from the main menu or by pressing Ctrl+Alt+S on the keyboard. The window that is displayed is the Server Explorer which lets us create and examine data connections. The Image below displays the Server Explorer.

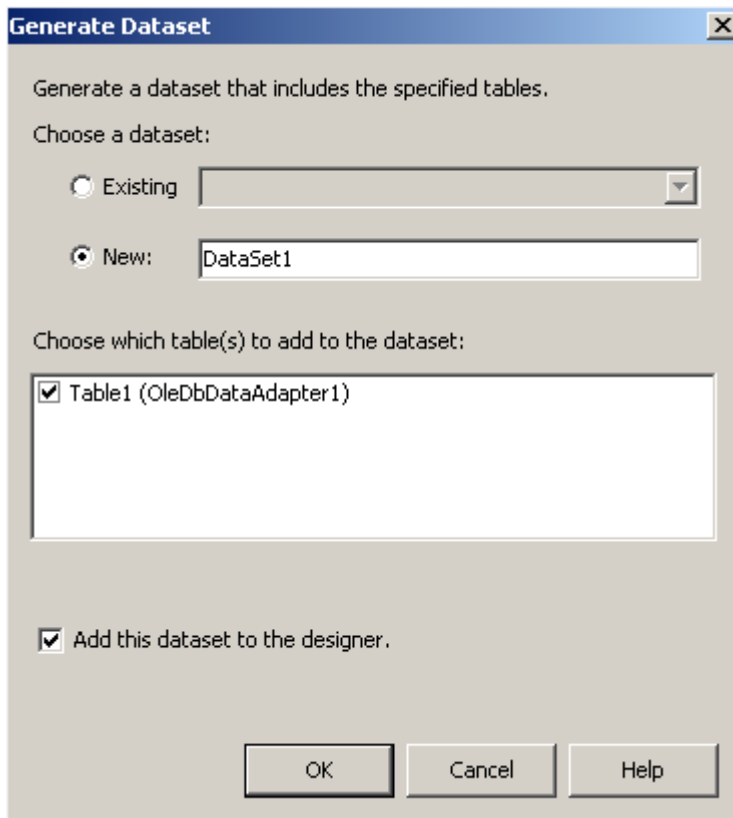


Let's start working with the Server Explorer. We will work with SQL Server, the default provider for .NET. We'll be displaying data from Customers table in sample North wind database in SQL Server. First, we need to establish a connection to this database. To do that, right-click on the Data Connections icon in Server Explorer and select Add Connection item. Doing that opens the Data Link Properties dialog which allows you to enter the name of the server you want to work along with login name and password. The Data Link properties window can be viewed in the Image below.



Since we are working with a database already on the server, select the option "select the database on the server". Selecting that lists the available databases on the server, select Northwind database from the list. Once you finish selecting the database, click on the Test Connection tab to test the connection. If the connection is successful, the message "Test Connection Succeeded" is displayed. When connection to the database is set, click OK and close the Data Link Properties. Closing the data link properties adds a new Northwind database connection to the Server Explorer and this connection which we created just now is part of the whole Visual Basic environment which can be accessed even when working with other applications. When you expand the connection node ("+" sign), it displays the Tables, Views and Stored Procedures in that Northwind sample database. Expanding the Tables node will display all the tables available in the database. In this example we will work with Customers table to display its data.

Now drag Customers table onto the form from the Server Explorer. Doing that creates SqlConnection1 and SqlDataAdapter1 objects which are the data connection and data adapter objects used to work with data. They are displayed on the component tray. Now we need to generate the dataset that holds data from the data adapter. To do that select Data->Generate DataSet from the main menu or right-click SqlDataAdapter1 object and select generate DataSet menu. Doing that displays the generate Dataset dialog box like the image below.



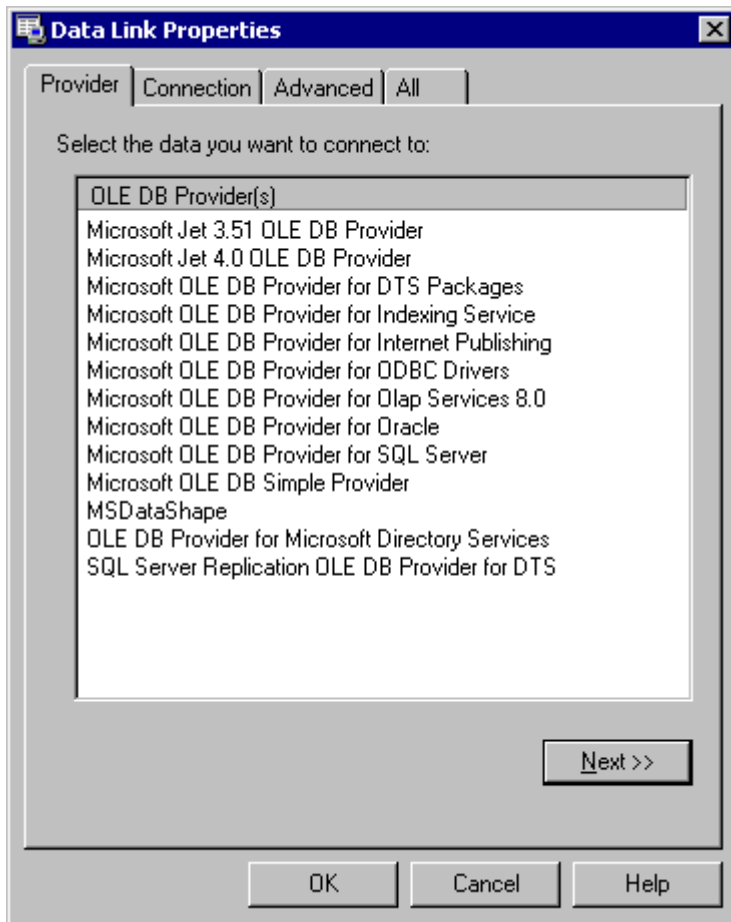
Once the dialogbox is displayed, select the radio button with New option to create a new dataset. Make sure Customers table is checked and click OK. Clicking OK adds a dataset, DataSet11 to the component tray and that's the dataset with which we will work. Now, drag a DataGridView from toolbox. We will display Customers table in this data grid. Set the data grid's DataSource property to DataSet11 and it's DataMember property to Customers. Next, we need to fill the dataset with data from the data adapter. The following code does that:

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) _  
    Handles MyBase.Load  
    DataSet11.Clear()  
    SqlDataAdapter1.Fill(DataSet11)  
    'filling the dataset with the data adapter's fill method  
End Sub
```

Once the application is executed, Customers table is displayed in the data grid. That's one of the simplest ways of displaying data using the Server Explorer window.

### **Microsoft Access and Oracle Database**

The process is same when working with Oracle or MS Access but with some minor changes. When working with Oracle you need to select Microsoft OLE DB Provider for Oracle from the Provider tab in the DataLink dialog. You need to enter the appropriate Username and password. The Data Link Properties window can be viewed in the Image below.



When working with MS Access you need to select Microsoft Jet 4.0 OLE DB provider from the Provider tab in DataLink properties.

Using OleDb Provider

The Objects of the OleDb provider with which we work are:

1. The OleDbConnection Class : The OleDbConnection class represents a connection to OleDb data source. OleDb connections are used to connect to most databases.
2. The OleDbCommand Class: The OleDbCommand class represents a SQL statement or stored procedure that is executed in a database by an OLEDB provider.
3. The OleDbDataAdapter Class : The OleDbDataAdapter class acts as a middleman between the datasets and OleDb data source. We use the Select, Insert, Delete and Update commands for loading and updating the data.
4. The OleDbDataReader Class :The OleDbDataReader class creates a data reader for use with an OleDb data provider. It is used to read a row of data from the database. The data is read as forward-only stream which means that data is read sequentially, one row after another not allowing you to choose a row you want or going backwards.

### **Coding**

Public Class Form1

**DECLARATION**

```
Dim con As ADODB.Connection
Dim cmd As ADODB.Command
Dim str, cnstr, sql As String
Dim cn As OleDb.OleDbConnection
```

**FORM LOAD**

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    con = New ADODB.Connection
    con.Open("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=c:\new
folder\employee.mdb")
End Sub
```

**ADDING A RECORD**

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    str = "insert into table1 values(" + TextBox1.Text + "," +
    TextBox2.Text + "," + TextBox3.Text + "," + TextBox4.Text +
    "," + TextBox5.Text + "," + TextBox6.Text + "," +
    TextBox7.Text + ")"
    cmd = New ADODB.Command
    cmd.ActiveConnection = con
    cmd.CommandText = str
    cmd.Execute(MsgBox("Add"))
    cmd.Cancel()
End Sub
```

**DELETING A RECORD**

```
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
    str = "delete * from table1 where empno=" + ComboBox1.Text + ""
    cmd = New ADODB.Command
    cmd.ActiveConnection = con
    cmd.CommandText = str
    cmd.Execute()
    MsgBox("Delete")
    cmd.Cancel()
End Sub
```

**ADDING ITEMS IN COMBO BOX**

```
Private Sub ComboBox1_GotFocus(ByVal sender As Object, ByVal e As
System.EventArgs) Handles ComboBox1.GotFocus
    ComboBox1.Items.Clear()
    cnstr = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=c:\new
folder\employee.mdb"
```



```
cn = New OleDb.OleDbConnection(cnstr)
cn.Open()
sql = "select empno from table1"
Dim ocmd As New OleDb.OleDbCommand(sql, cn)
Dim odatareader As OleDb.OleDbDataReader = ocmd.ExecuteReader
While odatareader.Read
    ComboBox1.Items.Add(odatareader.GetValue(0).ToString())
End While
odatareader.Close()
cn.Close()
End Sub
```

### **SELECTING ITEMS IN COMBO BOX & DISPLAYING IN TEXT BOX**

Private Sub ComboBox1\_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ComboBox1.SelectedIndexChanged

cnstr = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=c:\new folder\employee.mdb"

```
cn = New OleDb.OleDbConnection(cnstr)
cn.Open()
sql = "select * from table1 where empno=" & CInt(ComboBox1.SelectedItem) & ""
Dim ocmd As New OleDb.OleDbCommand(sql, cn)
Dim odatareader As OleDb.OleDbDataReader = ocmd.ExecuteReader
While odatareader.Read
    TextBox1.Text = odatareader.GetValue(0).ToString
    TextBox2.Text = odatareader.GetValue(1).ToString
    TextBox3.Text = odatareader.GetValue(2).ToString
    TextBox4.Text = odatareader.GetValue(3).ToString
    TextBox5.Text = odatareader.GetValue(4).ToString
    TextBox6.Text = odatareader.GetValue(5).ToString
    TextBox7.Text = odatareader.GetValue(6).ToString
End While
odatareader.Close()
cn.Close()
End Sub
```

### **UPDATING A RECORD**

Private Sub Button3\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button3.Click

str = "delete \* from table1 where empno=" + ComboBox1.Text + ""

cmd = New ADODB.Command

cmd.ActiveConnection = con

cmd.CommandText = str

cmd.Execute()

cmd.Cancel()

str = "insert into table1 values(" + TextBox1.Text + "," + TextBox2.Text + "," + TextBox3.Text + "," + TextBox4.Text + "," + TextBox5.Text + "," + TextBox6.Text + "," + TextBox7.Text + ")"

cmd = New ADODB.Command

```
cmd.ActiveConnection = con
cmd.CommandText = str
cmd.Execute()
MsgBox("Update")
cmd.Cancel()
End Sub
End Class
```

KAHE

**KARPAGAM ACADEMY OF HIGHER EDUCATION**  
**COIMBATORE - 21**

**DEPARTMENT OF COMPUTER SCIENCE,CA & IT**

**CLASS : III B.Sc COMPUTER TECHNOLOGY**

**BATCH : 2016-2019**

**Part -A Online Examinations**

**I One Marks**

**SUBJECT: . NET Programming**

**SUBJECT CODE: 16CTU502A**

**UNIT-IV**

S.No.	QUESTION	Option1	Option 2	Option 3	Option 4	Answer
1	ADO Refers to _____	ActiveX Data	Active Data	Applicati on	None	ActiveX Data
2	Which of this is not a server component	Counter Compone	Permissio n	Distribut ed	Content Linking	Distribut ed
3	The Provider to access MS Access database is	OleDb Data	SQL Data	ADO Data	DOA Data	OleDb Data
4	Which object is used to perform retrieve Operation	Connecti on Object	Comman d Object	Data object	Request Object	Comman d Object
5	_____ provides a language for describing Web Services	UDDI	WSDL	DDT	XML	WSDL
6	Drive,Folder,File Objects allbelong to _____	TextStrea m Object	FileSyste m Object	Dictionar y Object	Network System	FileSyste m Object
7	Which of these properties belong to TextStream Object	Drive	Line	Path	Size	Line
8	CTS Refers to _____	Common type	Common type	Central type	None	Common type
9	What data type is the output of a Web Service?	Any data type we	Only numeric	Text Strings	None of the	None of the
10	SQL Data Provider is used For	MS Access	SQL Server	Oracle	All the above	SQL Server
11	Which of the following operations can you NOT perform on an ADO.NET DataSet?	Develop ment	DataSet can be	DataSet can be	infer the schema	DataSet can be

12	Which of this not a OLE DB Provider	ODBC drivers	Packages OLAP		MSDataShape	Packages OLAP
13	_____ Object is specifically designed to run commands against a data store	Connection	Command	Dataset Object	DataReader Object	Command
14	_____ Object allows to connect to the data stores	Connection	Command	Dataset Object	DataReader Object	Connection
15	_____ ADO Object is to handle data not formatted in structured rows and columns	Connection	Command	Dataset Object	Record	Record
16	Which of these is a Access Data Type	Text	String	Char	Long	Text
17	RDO stands for _____	Remote data	remote developm	Remote data	Real Data Object	Remote data
18	Data set is a _____ architecture	connected	disconnected	self construct	locally connecte	disconnected
19	_____ is responsible for providing and maintaining connection to database	Data reader	Data adapter	data set	Data provider	Data provider
20	DAO stands for _____	Data access	Data adapter	Data available	Data provider	Data access
21	Local copy of database is called as _____	Data base copy	Dataset	Data provider	Data tables	Dataset
22	ADO NET comes with _____ providers	2	3	4	6	2
23	OleDb Data Provider is used For	MS Access	SQL Server	Oracle	SQL Server	a
24	_____ was a connected data access	RDO	ADO.NET	ASP.NET	DAO	ADO
25	In ADO.NET, The data are converted in to _____ Format	XML	XAML	HTML	CSS	XML
26	_____ is a disconnected, in-memory representation of data.	Dataset	Data Reader	Data Adapter	Data Provider	Dataset

27	The _____object which provides a forward-only, read only, connected recordset	Dataset	Data Reader	Data Adapter	Data Provider	Data Reader
28	The _____ object which populates a disconnected DataSet with data and performs update	Dataset	Data Reader	Data Adapter	Data Provider	Data Adapter
29	A _____object establishes the connection for the application with the database.	Connecti on	Comman d	Dataset Object	Record	Connecti on
30	_____ is the Extension for Access Database	.MDB	.RTF	.XML	.GCC	.MDB
31	_____Returns a single value from a database query	ExecuteN onQuery	ExecuteS calar	ExecuteR eader	ExecuteR eader	ExecuteS calar
32	_____Returns a result set by way of a DataReader object	ExecuteN onQuery	ExecuteS calar	ExecuteR eader	ExecuteR eader	ExecuteR eader
33	_____ is essentially the middleman facilitating all communication between the database and a DataSet.	Dataset	Data Reader	Data Adapter	Data Provider	Data Adapter
34	Which Command is used to insert the New Record to the Database Table	Insert	Add	Update	New	Insert
35	_____ is a collection of Record	Database	Table	File	Documen t	Table
36	Which of the Component is not a Component of ADO.NET	DataRead er	DataProv ider	DataAda pter	DataCha nger	DataCha nger
37	Which of the Following does not support Client Server Technology	DAO	ADO	RDO	ADO.NE T	DAO
38	Which object is used to perform Update Operation	Connecti on Object	Comman d Object	Data Adapter	Request Object	Data Adapter
39	In Access, the Image data type are stored using _____ data type	Text	BLOB	Memo	Number	BLOB
40	In SQL, the Image data type are stored in _____ format	Text	Unicode	Binary	Special	Binary
41	Which of these is not a Access Data Type	Text	String	memo	date	String

Reg.No -----

[16CTU502]

**KARPAGAM ACADEMY OF HIGHER EDUCATION**

(Deemed University Established Under Section 3 of UGC Act 1956)

COIMBATORE – 641 021

(For the candidates admitted in 2016 onwards)

**FIRST INTERNAL EXAMINATION, JULY 2018**

**Fifth Semester**

**COMPUTER TECHNOLOGY**

**. Net Programming**

Class : III B.Sc. [CT]

Date & Session: 07.2018 &

Maximum marks: 50

Time: 2 Hrs.

**PART-A**

**[20\*1= 20 Marks]**

**Answer all the questions**

1. .Net is a technology developed by \_\_\_\_\_ company
  - a. **Microsoft**
  - b. Sun Microsystems
  - c. IBM
  - d. Apple computers
2. The Design window appears \_\_\_\_\_ by default.
  - a. Auto-Hidden
  - b. **Docked**
  - c. Floating
  - d. Closed
3. Code that targets the Common Language Runtime is known as \_\_\_\_\_.
  - a. Distributed Code
  - b. **Managed Code**
  - c. Legacy code
  - d. Native Code
4. Which of the following task is done by the Garbage collector?
  - a. **Freeing memory referenced by unreferenced objects**
  - b. Closing unclosed Files and Databases
  - c. Freeing memory on the stack
  - d. All the above
5. VB.Net is a \_\_\_\_\_ programming paradigm.
  - a) Procedural
  - b) Structured
  - c) **Object Oriented**
  - d) Monolithic
6. Data members of a class are by default \_\_\_\_\_.
  - a. public
  - b. **private**
  - c. static
  - d. volatile
7. Member functions of a class are by default \_\_\_\_\_.
  - a) **public**
  - b) private
  - c) static
  - d) volatile
8. IDE stands for
  - a. Internet Design Environment
  - b. **Integrated Development Environment**
  - c. Internet distribution Environment
  - d. Interface Design Environment
9. The final compiled version of a Project is \_\_\_\_\_.
  - a) Form
  - b) Software
  - c) **Components**
  - d) Files

10. \_\_\_\_\_ is a collection of files that can be compiled to create a distributed component  
 a) Form c) Components  
 b) Software **d) Project**
11. \_\_\_\_\_ is a collection of projects and files that composed an application or component  
**a. Solution** b. Software c. Forms d. Project
12. Every object has a distinct set of attributes known as  
 a) members b) data c) properties d) methods
13. The property that must be set first for any new object is the \_\_\_\_\_  
**a) Name** b) Colour c) Size d) Binding
14. Objects that can be placed on a form are called \_\_\_\_\_  
 a) Picture b) Tools c) Buttons **d) Controls**
15. What is the statement used to declare variable?  
 a) loc **b) dim** c) global d) redim
16. The data type of the variable is defined by using the ----- clause  
 a) in b) where **c) as** d) is
17. A composite data type is of ----- types  
 a) 3 b) 4 c) 5 **d) 2**
18. Constants are declared using the keyword  
 a) constant c) consta  
**b) const** d) fixed
19. An abstract class contains  
**a) abstract methods** c) friend method  
 b) non-abstract methods d) overloading method
20. The storage size for Byte data types is -----  
 a) 2 b) 4 **c) 1** d) 8

## PART – B

(3\*1=20 Marks)

### Answer all the questions

21. Define Common Type System
22. Common Type System. In Microsoft's .NET Framework, the Common Type System (CTS) is a standard that specifies how type definitions and specific values of types are represented in computer memory. It is intended to allow programs written in different programming languages to easily share information.
23. Describe Output Window  
 The Output window can display status messages for various features in the integrated development environment (IDE). To open the Output window, on the menu bar, choose View/Output (or click CTRL + ALT + O).
24. What is meant by Menu bar? Give example.  
 Take the following steps: Drag and drop or double click on a MenuStrip control, to add it to the form. Click the Type Here text to open a text box and enter the names of the menu items or sub-menu items you want. Complete



the menu structure shown in the diagram above. Add a sub menu Exit under the File menu.

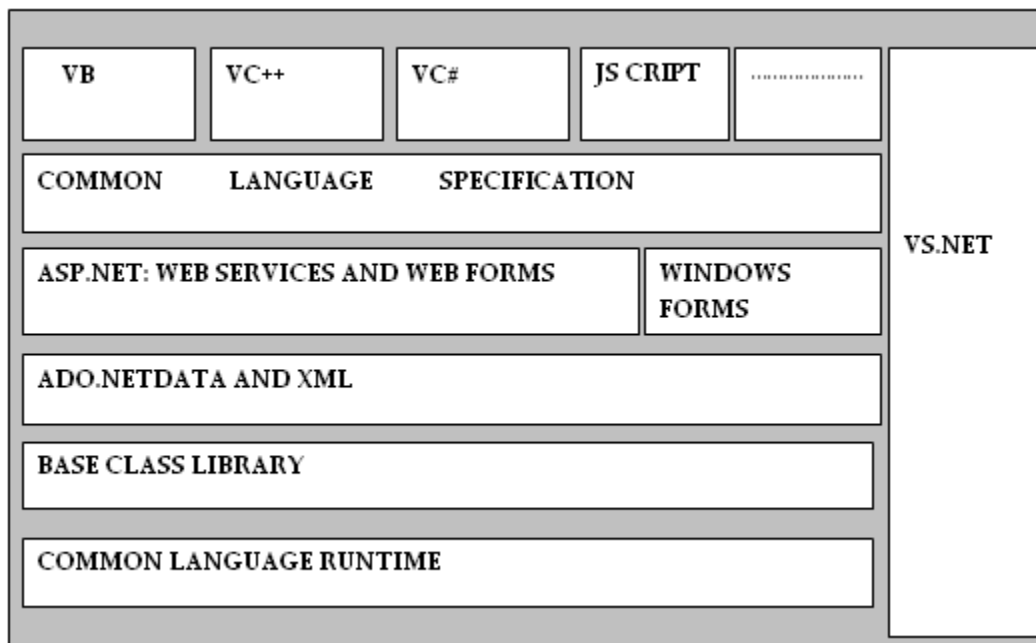
**PART – C**

**(3\*8=24 Marks)**

**Answer all the questions**

25. a. Draw the .NET framework architecture and explain the features.

NET is a "Software Platform". It is a language-neutral environment for developing rich .NET experiences and [building applications](#) that can easily and securely operate within it. When developed applications are deployed, those applications will target .NET and will execute wherever .NET is implemented instead of targeting a particular Hardware/OS combination. The components that make up the .NET platform are collectively called the .NET Framework. The .NET Framework is a managed, type-safe environment for developing and executing applications. The .NET Framework manages all aspects of program execution, like, allocation of memory for the storage of data and instructions, granting and denying permissions to the application, managing execution of the application and reallocation of memory for resources that are not needed.



**Fig Net Framework architecture**

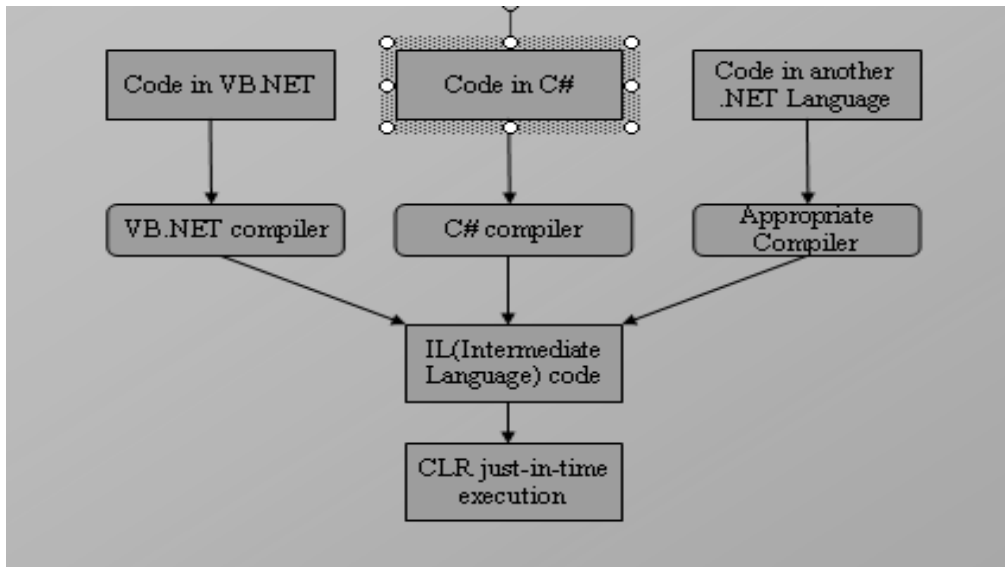
The .NET Framework is designed for cross-language compatibility. Cross-language compatibility means, an application written in Visual Basic .NET may reference a DLL file written in C# (C-Sharp). A [Visual Basic](#) .NET class might be

derived from a C# class or vice versa. The .NET Framework consists of two main components:

- Common Language Runtime (CLR)
- Class Libraries

#### **i) Common Language Runtime (CLR)**

The CLR is also known as the "execution engine" of .NET. It provides the environment within which the programs run. The main responsibility of the CLR is that manages the execution of programs and provides core services, such as code compilation, memory allocation, thread management, and garbage collection. Through the Common Type [System](#) (CTS), it enforces strict type safety, and it ensures that the code is executed in a safe environment by enforcing code access [security](#). The software version of .NET is actually the CLR version. When the .NET program is compiled, the output of the compiler is not an executable file but a file that contains a special type of code called the [Microsoft](#) Intermediate Language (MSIL), which is a low-level set of instructions understood by the common language run time. This MSIL defines a set of portable instructions that are independent of any specific [CPU](#). It's the job of the CLR to translate this Intermediate code into a executable code when the program is executed making the program to run in any environment for which the CLR is implemented. And that's how the .NET Framework achieves Portability. This MSIL is turned into executable code using a JIT (Just in Time) compiler. The process goes like this, when .NET programs are executed, the CLR activates the JIT compiler. The JIT compiler converts MSIL into native code on a demand basis as each part of the program is needed. Thus the program executes as a native code even though it is compiled into MSIL making the program to run as fast as it would if it is compiled to native code but achieves the portability benefits of MSIL.



**Fig Common Language Runtime**

## ii) .Net Framework Class Libraries

Class library is another major entity of the .NET Framework which is designed to integrate with the common language runtime. This library gives the program access to runtime environment. The class library consists of lots of prewritten code that all the applications created in VB .NET and Visual Studio .NET will use. The code for all the elements like forms, controls and the rest in VB .NET applications actually comes from the class library.

## Name Space

.Net provides several ways to organize the Visual basic code. One among them is Namespace. A namespace is a collection of different classes. All VB applications are developed using classes from the .NET System namespace. The namespace with all the built-in VB functionality is the System namespace. All other namespaces are based on this System namespace. Some of the namespace in vb.net includes:

1. System: Includes essential classes and base classes for commonly used data types, events, exceptions and so on
2. System. Collections: Includes classes and interfaces that define various collection of objects such as list, queues, hash tables, arrays, etc

3. System. Data: Includes classes which lets us handle data from data sources
4. System.Data.OleDb: Includes classes that support the OLEDB .NET provider
5. System.Data.SqlClient: Includes classes that support the SQL Server .NET provider
6. System. Diagnostics: Includes classes that allow to debug our application and to step through our code
7. System.Drawing: Provides access to drawing methods
8. System. Reflection: Includes classes and interfaces that return information about types, methods and fields
9. System. Security: Includes classes to support the structure of common language runtime security system
10. System. Threading: Includes classes and interfaces to support multithreaded applications
11. System. Web: Includes classes and interfaces that support browser-server communication
12. System.Web.Services: Includes classes that let us build and use Web Services
13. System.Windows.Forms: Includes classes for creating Windows based forms
14. System.XML: Includes classes for XML support
15. System. Globalization: Includes classes that specify culture-related information
16. Systemic: Includes classes for data access with Files
17. System.Net: Provides interface to protocols used on the internet

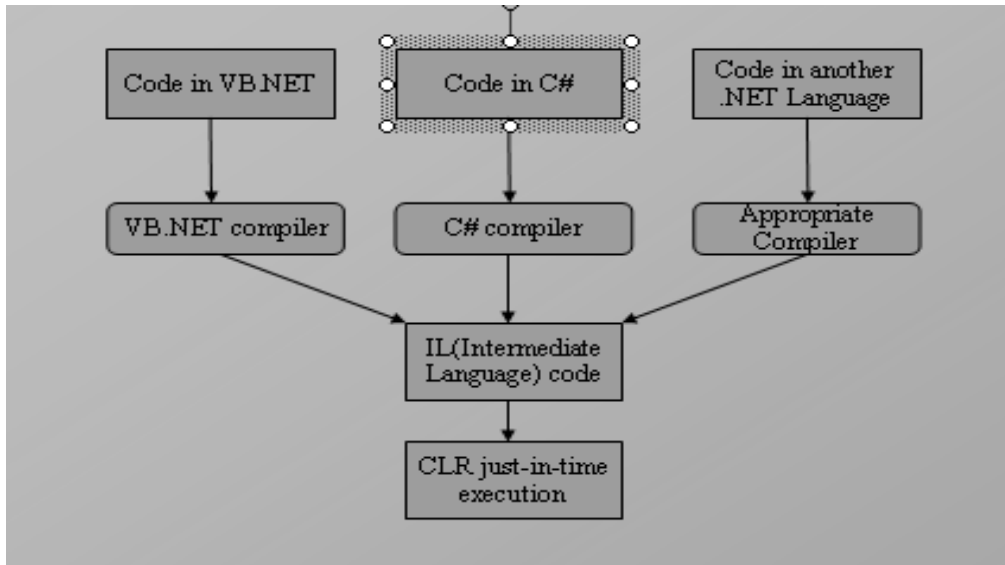
(OR)

b. Enlighten Common Language Runtime with diagram.

**iii) Common Language Runtime (CLR)**

The CLR is also known as the "execution engine" of .NET. It provides the environment within which the programs run. The main responsibility of the CLR is that manages the execution of programs and provides core services, such as code compilation, memory allocation, thread management, and garbage collection. Through the Common Type [System](#) (CTS), it enforces strict type safety, and it ensures that the code is executed in a safe environment by enforcing code access [security](#). The software version of .NET is actually the CLR version. When the .NET program is compiled, the output of the compiler is not an executable file but a file that contains a special type of code called the

[Microsoft](#) Intermediate Language (MSIL), which is a low-level set of instructions understood by the common language run time. This MSIL defines a set of portable instructions that are independent of any specific [CPU](#). It's the job of the CLR to translate this Intermediate code into a executable code when the program is executed making the program to run in any environment for which the CLR is implemented. And that's how the .NET Framework achieves Portability. This MSIL is turned into executable code using a JIT (Just in Time) compiler. The process goes like this, when .NET programs are executed, the CLR activates the JIT compiler. The JIT compiler converts MSIL into native code on a demand basis as each part of the program is needed. Thus the program executes as a native code even though it is compiled into MSIL making the program to run as fast as it would if it is compiled to native code but achieves the portability benefits of MSIL.



**Fig Common Language Runtime**

26. a. Explain the Methods and events with example.

### **Methods and Events**

In Visual Basic you will work with objects, which have properties, methods, and events. Each object is based on a class.

- **Objects:** Think of an **object** as a thing, or a noun. Examples of objects are forms and controls. **Forms** are the windows and dialog boxes you place on the screen; **controls** are the components you place inside a form, such as text boxes, buttons, and list boxes.

- **Properties:** **Properties** tell something about an object, such as its name, color, size, location, or how it will behave. You can think of properties as adjectives that describe objects. When you refer to a property, you first name the object, add a period, and then name the property. For example, refer to the Text property of a form called Form1 as Form1.Text (pronounced “form1 dot text”).
- **Methods:** Actions associated with objects are called *methods*. **Methods** are the verbs of object-oriented programming. Some typical methods are **Move**, **Hide**, **Show**, and **Clear**. You refer to methods as Object.Method (“object dot method”). For example, a **Show** method can apply to different objects. **Form1.Show** shows the form object called Form1; **btnExit.Show** shows the button object called btnExit.
- **Events:** You can write procedures that execute when a particular event occurs. An **event** occurs when the user takes an action, such as clicking a button, pressing a key, scrolling, or closing a window. Events can also be triggered by actions of other objects, such as repainting a form or a timer reaching a preset point.
- **Classes:** A **class** is a template or blueprint used to create a new object. Classes contain the definition of all available properties, methods, and events.

(OR)

b. Draw VB.Net Integrated Development Environment and explain

## Integrated Development Environment (IDE)

The Integrated Development Environment (IDE) is the development environment for all .Net based applications. To say in other words, IDE is shared by all programming languages in Visual Studio. You can view the toolbars towards the left side of the image along with the Solution Explorer window towards the right. This section Details the components of IDE with the screen shots and their usage in developing the applications.

### 1 Startup Page

Many of the "global" resources in VS.NET are on the VS.NET Start Page. Many links are provided from the start up page. They are organized into three categories such as projects, online resources and profiles. Under the project tab, it is possible to open the existing projects/application or to create new project/applications. Using the online resource option, it is possible to link to Microsoft help and reference page. With the profile option it is possible to change

keyboard schemes, windows layout, help filter and show help settings. Following figure shows the screenshot of startup page.

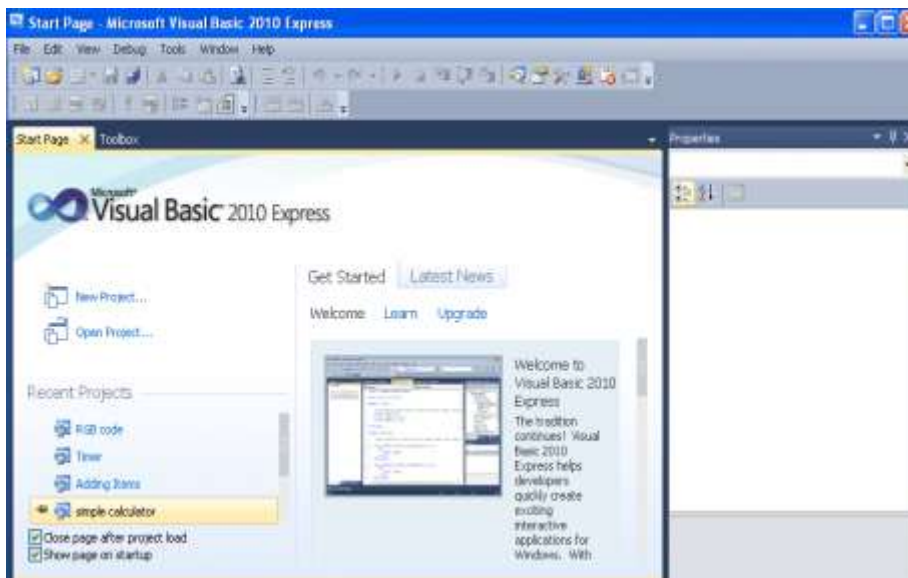
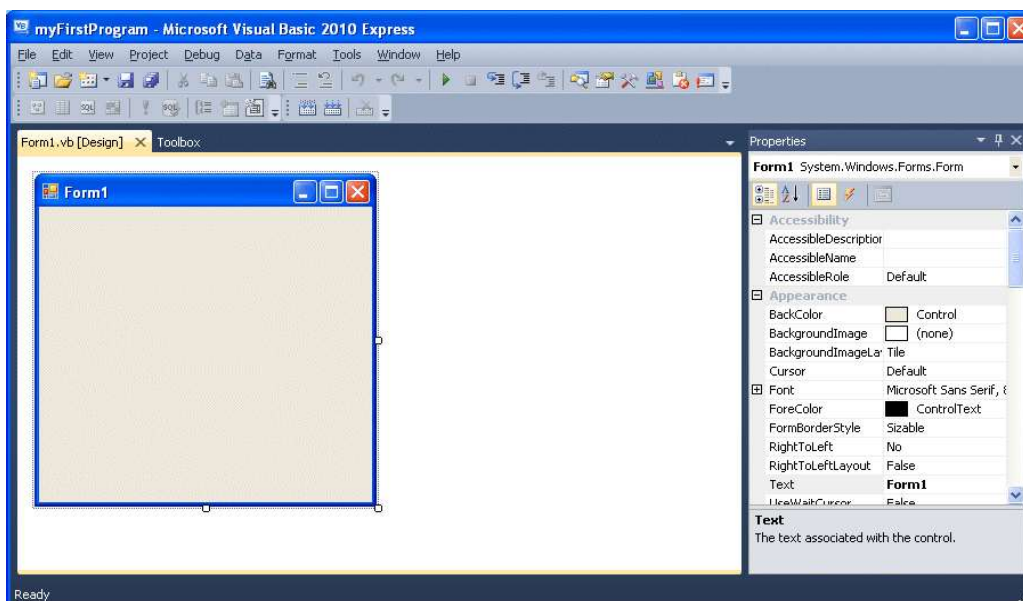


Fig Startup page

## 2 Windows Form Designer

System.Windows.Forms is the foundation class for all forms to be created. All the forms that are created in VB .NET are also inheriting from this base class. This class provides for all the facilities needed for the form. Additional functionality can be added by separate codes.



## Fig Windows Form Designer

### 3 Solution Explorer Window

The Solution Explorer window gives an overview of the solution we are working with and lists all the files in the project. An image of the Solution Explorer window is shown below. It shows the project named as WindowsApplication34, an assembly file which contains assembly information of the application that is discussed in the previous section and the Form which has the default name Form1.vb. If the programmer adds a new resource for instance, another Form then the form will be visible in solution explorer window.

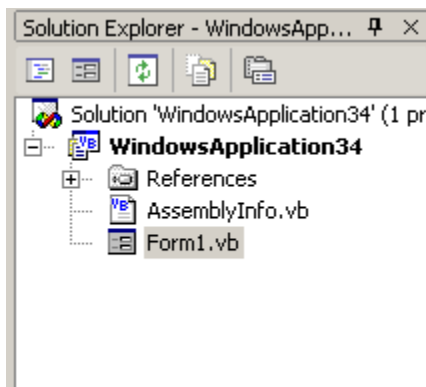
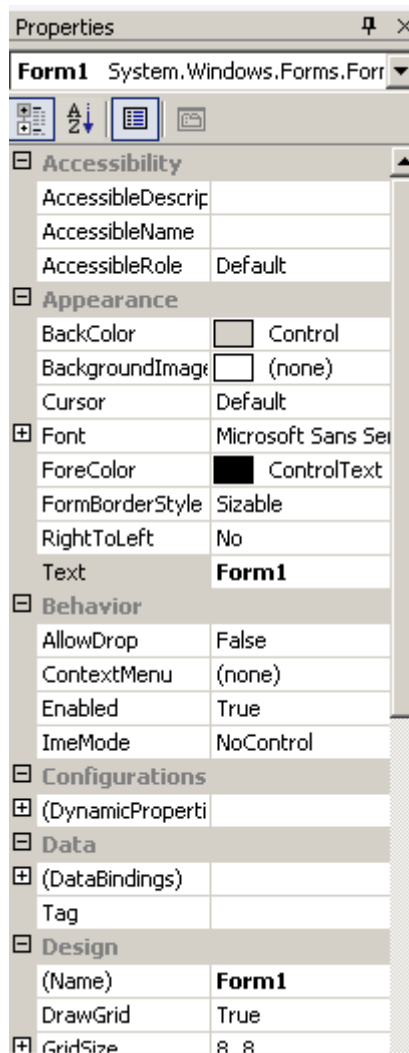


Fig Solution Explorer Windows

### 4 Properties Window

The properties window allows us to set properties for various objects at design time. For example, if you want to change the font, font size, back color, name, text that appears on a button, textbox etc, you can do that in this window. Below is the image of properties window. You can view the properties window by selecting View->Properties Window from the main menu or by pressing F4 on the keyboard.

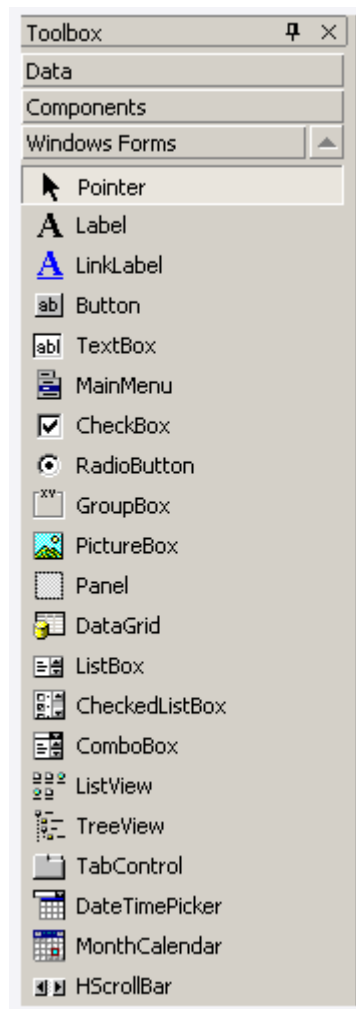




**Fig Property Windows**

## 5 Toolbox Window

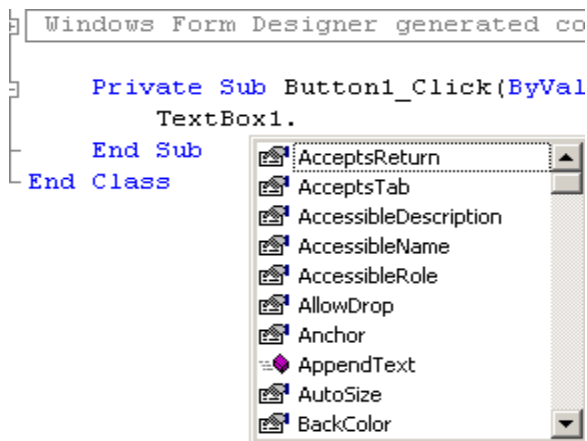
The toolbox window is the window that gives us access to all controls, components, etc. As you can see from the image below, the toolbox uses tabs to divide its contents into categories (Data, Components, Windows Forms and General). The Data tab displays tools for creating datasets and making data connections, the Windows Forms tab displays tools for adding controls to forms, the General tab is left empty by default, the Clipboard Ring tab displays recent items stored in the clipboard and allows us to select from them.



**Fig Toolbox**

## **6 Intellisense**

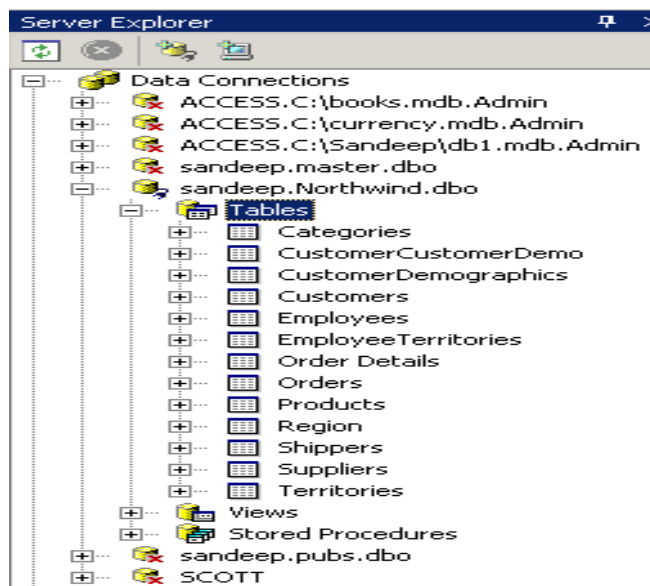
Intellisense is what that is responsible for the boxes that open as we type the code. IntelliSense provides a list of options that make language references easily accessible and helps us to find the information we need. They also complete the typing for us. The image below displays that.



**Fig Intellisense**

## 7 Server Explorer Window

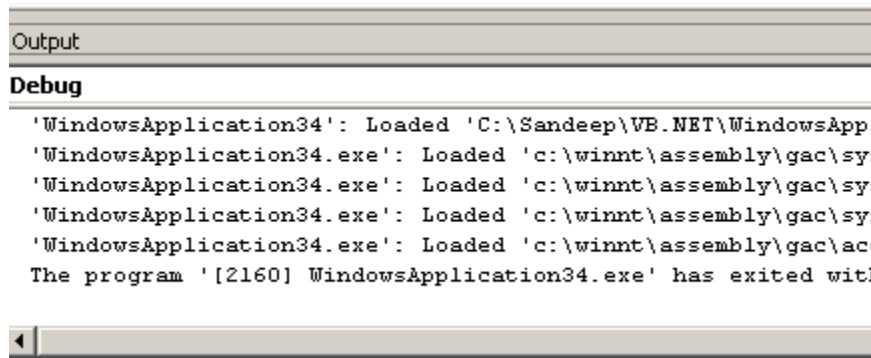
The Server Explorer window is a tool that provides "drag and drop" feature and helps us work with databases in an easy graphical environment. For example, if we drag and drop a database table onto a form, VB .NET automatically creates connection and command objects that are needed to access that table. The image below displays Server Explorer window.



**Fig Server Explorer Window**

## 8 Output Window

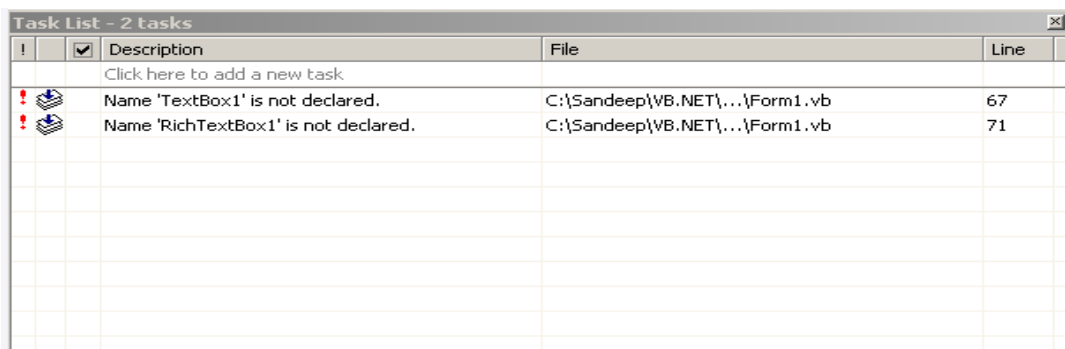
The output window as you can see in the image below displays the results of building and running applications.



**Fig Windows Form Designer**

## 9 Task List Window

The task list window displays all the tasks that VB .NET assumes we still have to finish. You can view the task list window by selecting View->Show tasks->All or View->Other Windows->Task List from the main menu. The image below shows that. As you can see from the image, the task list displayed "TextBox1 not declared", "RichTextBox1 not declared". The reason for that message is, there were no controls on the form and attempts were made to write code for a textbox and a rich textbox. Task list also displays syntax errors and other errors you normally encounter during coding



**Fig Task List Window**

27. a. Why do we need Variables? And explain with example.

## Variable and Rules

An identifier is used to identify and store some value. If the value of the identifier is changed during the execution of the program, then the identifier is known as variable. When the variable holds numeric data, it is called as numeric variable and when it holds character(s) it is called as Character/String variable. To name a variable in Visual Basic 2010, you have to follow a set of rules.

1. It must be less than 255 characters
2. No spacing is allowed
3. It must not begin with a number
4. Period is not permitted

Examples of valid and invalid variable names are displayed in Table

Valid Variable Name	Invalid Variable Name
Name	Name of student
Student_name	1name
Student123	Student-name
Number1, number2	Number 1

## 5 Declaring Variables

Before using a variable in the program, it must be declared. The declaration of variable tells the compiler about the type of data to be held by the variable and the memory to be reserved for it and to store the initial value in it. Variables are normally declared in the general section of the codes' windows using the Dim statement. Dim is a keyword. Dim stands for Dimension. The format is as follows:

### Dim Variable Name As Data Type

Some of the valid variables are as follows

```
Dim i as integer      // Integer variable declaration
Dim name as string    // String variable declaration
Dim x, y as integer   // Two variables are used in the same declaration
Dim x as double        // Real variable declaration
Dim doDate As Date    // Date is a special data type for Date function
```

It is possible to combine two or more variables in a single statement. The variables names can be separated by using a comma as shown in the above statement. There are two possible formats are available for declaring string such as one for the variable-length string and another for the fixed-length string. For the variable-length string, just use the same format as shown above. However, for the fixed-length string the total number of characters should be specified at the declaration time. The format for fixed length string is

### **Dim VariableName as String \* n**

Where n defines the total number of string that a string can hold.

```
Private Sub Button1_Click (ByVal sender As System.Object, ByVal e_
As System.EventArgs) Handles Button1.Click
```

```
Dim i, j, k As Integer
```

```
i = 20
```

```
j = 20
```

```
k = i + j
```

```
TextBox1.Text = k
```

```
Dim a, b, c As String
```

```
a = " Welcome"
```

```
b = " To "
```

```
c = "Visual Basic"
```

```
TextBox2.Text = a + b + c
```

```
End Sub
```

In the above example, The variable C produces the output string “Welcome to Visual Basic”. Here + symbol denotes Concatenation operator.

### **6 Variable initialization**

After declaring various variables using the Dim statements, it is possible to assign values to those variables. The general format of an assignment is

**Variable=Expression**

The variable can be a declared variable or a control property value. The expression could be a mathematical expression, a number, a string, a Boolean value (true or false) and etc. The following are some examples:

```
Number1 = 100
```

```
Number2 = Number1-45
```

```
Username = “AnandKumar”
```

```
Button1.Visible = True
```

```
Label4.Caption = textbox1.Text
```

```
Number3 = Val (Textbox1..Text)
```

### **7 New and Nothing Keyword**

The New is a Keyword which is used to create the instance of the class. Unlike value types, such as Integer and Double, objects are reference types, and it should be created explicitly before using them in the program. It is also possible to create an instance of the form as well as all the controls. For example Button is control and the class for button is Button. So the instance can be created in such a way as shown below

```
Dim Button1 As System.Windows.Forms.Button  
Dim Button2 As New System.Windows.Forms.Button()  
Dim frm As New System.Windows.Forms.Form1()
```

In the above statements, the first statement declares an object variable that can contain a reference to a button object. However, the variable Button1 contains the value nothing until you assign an object of type Button to it. The second statement also defines a variable that can contain a button object, but the New keyword creates a button object and assigns it to the variable Button2. Both forms and controls are actually classes; the New keyword can be used to create new instances of these items as needed.

The Nothing keyword represents the default value of any data type. Assigning Nothing to a variable sets it to the default value for its declared type. If that type contains variable members, they are all set to their default values. The following example uses the Nothing keyword

```
Private Sub Button1_Click (ByVal sender As System.Object, ByVal e_ As  
System.EventArgs) Handles Button1.Click
```

```
    Dim i as integer  
    Dim s as Boolean  
    i=Nothing           //sets i to 0  
    s= Nothing         //sets s to False  
End Sub
```

## **8 Implicit and Explicit declarations**

There are two ways to declare the variables for the program. They are implicit declaration and Explicit declaration. In the implicit declaration, Visual basic automatically create the variable for the user application. Implicit declaration means that Visual Basic automatically creates a variant for each identifier it recognizes as a variable in an application

The second approach to declaring variable is to explicitly declare them with one of the following keywords Dim, Static, Private, and Public. The choice of keyword has a profound effect on the variable's scope within the application and determines where the variable can be used in the program

```
Dim VariableName as DataType  
Static VariableName as DataType  
Private VariableName as DataType  
Public VariableName as DataType
```

Visual Basic.Net reserves the amount of memory required to hold the variable as soon as the declaration statement is executed. After a variable is declared, it is not possible to change its data type, although it is quite easy to convert the value of a variable and assign the converted value to another variable

## 9. Scope of Variable

The scope of a variable referred to as accessibility of a variable. It includes the information about the variable such as where the variable can be read from and/or written to, and the variable's lifetime, or how long it stays in memory.. Variables can be declared in four different locations in the programs as shown in the following table.

S.No	Location	Usage
1	Block	If the variable is declared within a control statement such as an If statement, then that variable's scope is only until the end of the block. The lifetime is until the procedure ends.
2	Procedure	If the variable is declared within a procedure, but outside of any If statement, then the scope is until the End Sub or End Function. The lifetime of the variable is until the procedures ends
3	Module/Class	Variables can be declared outside of any procedure, but it must be within a Class...End Class or Module...End Module statement. The scope is any procedure within this module. The lifetime for a variable defined within a class is until the object is cleaned up by the garbage collector. The lifetime for a variable defined within a module is until the program ends.
4	Project	It is possible to declare a Public variable within a Module...End Module statement, and that variable's scope will be any procedure or method within the project. The lifetime of the variable will be until the program ends.

**Table Scope of the variables**

There are many different ways you can declare variables. Variables can be declared with any one of the scope such as public, private, protected, friend and protected friend. If the variable is declared as public then the variable will be available anywhere in or outside of the project. If it is declared as private then it is visible only within the block where it is declared. When the variable is declared as protected then it can be used in the class where defined and within any inherited class. When used with friend it can only be accessed by code in the same project/assembly. If the variable is declared as protected friend then it has the combination of protected and friend.

(OR)

b. Write VB.Net program to implement the calculator.





### **Public Class Form1**

```

    Dim operand1 As Double
    Dim operand2 As Double
    Dim [operator] As String
    Dim d As Double
    Dim b As Single
    Dim result As Double

```

**Private Sub Button1\_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles Button1.Click, Button2.Click, Button3.Click, Button4.Click, Button5.Click, Button6.Click, Button7.Click, Button8.Click, Button9.Click, Button10.Click**

```

    TextBox1.Text = TextBox1.Text & sender.text

```

**End Sub**

**Private Sub Button20\_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles Button20.Click**

```

    TextBox1.Text = ""

```

**End Sub**

**Private Sub Button11\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button11.Click**

```

    operand1 = Val(TextBox1.Text)
    TextBox1.Text = ""
    TextBox1.Focus()
    [operator] = "+"

```

**End Sub**

**Private Sub Button12\_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles Button12.Click**

operand1 = Val(TextBox1.Text)

TextBox1.Text = ""

TextBox1.Focus()

[operator] = "-"

**End Sub**

**Private Sub Button13\_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles Button13.Click**

operand1 = Val(TextBox1.Text)

TextBox1.Text = ""

TextBox1.Focus()

[operator] = "\*"

**End Sub**

**Private Sub Button14\_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles Button14.Click**

operand1 = Val(TextBox1.Text)

TextBox1.Text = ""

TextBox1.Focus()

[operator] = "/"

**End Sub**

**Private Sub Button16\_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles Button16.Click**

b = -1 \* Val(TextBox1.Text)

TextBox1.Text = b

**End Sub**

**Private Sub Button18\_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles Button18.Click**

If TextBox1.Text <> 0 Then

d = 1 / Val(TextBox1.Text)

TextBox1.Text = d

End If

**End Sub**

**Private Sub Button17\_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles Button17.Click**

TextBox1.Text = TextBox1.Text & "."

**End Sub**

**Private Sub Button15\_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles Button15.Click**

operand2 = Val(TextBox1.Text)

```

Select Case [operator]
Case "+"
    result = operand1 + operand2
    TextBox1.Text = result
Case "-"
    result = operand1 - operand2
    TextBox1.Text = result
Case "*"
    result = operand1 * operand2
    TextBox1.Text = result
Case "/"
    result = operand1 / operand2
    TextBox1.Text = result
End Select
End Sub
End Class

```

## OUTPUT



Reg. No -----

[16CTU502A]

**KARPAGAM ACADEMY OF HIGHER EDUCATION**

(Deemed to be University)

(Established Under Section 3 of UGC Act 1956)

COIMBATORE – 641 021

(For the candidates admitted in 2016 onwards)

**SECOND INTERNAL EXAMINATION, AUGUST 2018**

**Fifth Semester**

**COMPUTER TECHNOLOGY**

**. Net Programming**

Class : III B.Sc. [CT]

Date & Session: 12.08.2018 (AN)

Maximum: 50

Time: 2 Hrs.

**PART-A**

**[20\*1= 20 Marks]**

**Answer all the questions**

1. This data type can be used for currency values  
a) Currency    b) Dollar    c) Object    **d) Decimal**
2. Size of integer data type is ----- bits  
a) 8    b) 16    c) 24    **d) 32**
3. Which of the given data types used to represent integer numbers  
a) long    b) short    c) byte    **d) All the above**
4. ----- is the operator used for string concatenation  
a) Cat    b) Str    c) ^    **d) &**
5. The order in which the operators in an expression are evaluated is known as -----  
**a) operator precedence**    b) operator overloading  
b) associativity of operators    d) operator evaluation
6. And , Or , Not, Xor are called \_\_\_\_\_ operators  
**a) Boolean**    b) Relational    c) comparison    d) String
7. \_\_\_\_\_ function is used to retrieve only the month part of the date  
a) DateDiff()    **b) DatePart()**    c) DateInterval()    d) Date.Month()
8. Which function returns the system's current date and time  
**a) DateTime.Now**    b) DateTime.Today  
c) DateTime.System    d) DateTime.Current
9. XAML documents are arranged as a heavily \_\_\_\_\_ of elements.  
**a) Nested Tree**    b) Binary Tree    c) TPR Tree    d) BBX Tree
10. Data types standardized across the CLR are called \_\_\_\_\_  
a) Standard Data Types    **b) CTS**    c) CLR    d) System Data types
11. The CTS equivalent of Integer data type is  
**a) System.Int32**    b) System.Int16    c) System.Int64    d) System.Int24
12. The Size of 'System.Double' data type is  
a) 8 bits    b) 2 bytes    c) 4 bytes    **d) 8 bytes**
13. \_\_\_\_\_ panel stretches controls against one of its outside edges  
a) WrapPanel    b) StackPanel    **c) DockPanel**    d) DoublePanel

14. The \_\_\_\_\_ is the most powerful layout container in WPF.  
a) Layout      b) **Grid**      c) Flexgrid      d) Datagrid
15. \_\_\_\_\_ an ideal tool for carving your window into smaller regions that user can manage with other panels  
a) WrapPanel      b) StackPanel      c) DockPanel      d) **Grid**
16. \_\_\_\_\_ allows user to resize rows or columns in WPF Applications  
a) **Grid Splitter**      b) Grid Extractor      c) Grid Merger      d) Grid Arrabger
17. Which of the following is not a Text Control  
a) TextBox      b) RichTextBox      c) PasswordBox      d) **ListBox**
18. ScrollBar, ProgressBar, and Slider are derive from the \_\_\_\_\_ class  
a) **RangeBase** b) ControlBase      c) ListBase      d) QueryBase
19. The \_\_\_\_\_ control provides a convenient way to enable scrolling of content in Windows Presentation Foundation (WPF) applications.  
a) **ScrollViewer**      b) ImageViewer      c) ListView      d) GridView
20. \_\_\_\_\_ Control provide an out-of-the-box spell checking functionality.  
a) **TextBox**      b) ListBox      c) ComboBox      d) LabelBox

### PART – B

(3\*2 = 6 Marks)

#### Answer all the questions

21. Write the syntax for MsgBox()  
syntax is  
MsgBox(prompt [, buttons] [, title])
22. Define WPF  
The Windows Presentation Foundation (WPF) changes all this by introducing a new model with entirely different technology. Although WPF includes the standard controls you're familiar with, it draws every text, border, and background fill itself. As a result, WPF can provide much more powerful features that let you alter the way any piece of screen content is rendered. Using these features, you can restyle common controls such as buttons, often without writing any code. The Windows Presentation Foundation (WPF) is an entirely new graphical display system for Windows. WPF is designed for .NET, influenced by modern display technologies such as HTML and Flash, and hardware-accelerated.
23. Name the properties and events of XAML  
VerticalAlignment,  
HorizontalAlignment,  
FontFamily,  
FontSize, and  
Foreground

**PART – C**  
**Answer all the questions**

**(3\*8 =24 Marks)**

**24. a) Explain the about conditional statements with example**

The conditional control statements are used to check some condition and then transfer the control based on the condition result. There are few control statements available and they are

1. If....Then statement
2. If....Then... Else
3. If....Then....Else If
4. Select Case

**If....Then Statement**

This is the simplest control structure which asks the computer to perform a certain action specified by the VB expression if the condition is true. However, when the condition is false, no action will be performed. The general format for the if...then.. Statement is

If condition Then

Statements

End If

**Program to check whether the given number is greater than 50 or not**

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

```
Dim number1 As Integer
```

```
number1 = TextBox1.Text
```

```
If number1 > 50 Then
```

```
Msgbox” The number is greater than 50”
```

```
End If
```

```
End Sub
```

```
End Sub
```

This program displays the message box if the condition is satisfied. If the user enters the number smaller than 50 then the user will not receive any output. This indicates that if only the condition is satisfied then the statement will be executed. It is the drawback of simple If statement.

**If....Then...Else Statement**

If....Then statement that is used above is not very useful in programming and it does not provide choices for the users. In order to provide a choice, If....Then...Else Statement can be used. This control structure will ask the computer to perform a certain action specified by the VB expression if the condition is true. And when the condition is false an alternative action will be executed. The general format for the if...then.. Else statement is

```

If condition Then
    Statement (1)
Else
    Statement (2)
End If

```

The above program can be rewritten with the use of If Then Else Statement

```

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

```

```

Dim number1 As Integer

```

```

number1 = TextBox1.Text

```

```

If number1 > 50 Then

```

```

Msgbox " The number is greater than 50"

```

```

Else

```

```

Msgbox " The number is smaller than 50"

```

```

End If

```

```

End Sub

```

```

End Sub

```

In the previous program, when user inputs the value smaller than 50 then no output will be displayed. But in this program, even if the input is smaller than 50 the message box will be displayed as "The number is smaller than 50". This is possible only with the use of If Then else statement.

### **If....Then...Else If Statement**

If there are more than two alternative choices, using just If....Then....Else statement will not be enough. In order to provide more choices, we can use the If....Then...ElseIf Statement. The general format for the if...then.. Else statement is

```

If condition Then
    Statement (1)
ElseIf condition Then
    Statement (2)
ElseIf condition Then
    Statement (3).

```

```

.
.
.

```

```

Else

```

```

    Statement (n)

```

```

End If

```

**This is a program to print the Grade of a student based on the average of marks.**

```

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

```

```

Dim Mark As Integer

```

```

Dim Grade as String

```

```

Mark = TextBox1.Text
If Mark >=80 Then
Grade="A"
ElseIf Mark>=60 and Mark<80 then
Grade="B"
ElseIf Mark>=40 and Mark<60 then
Grade="C"
Else
Grade="D"
End If
End Sub

```

### **Select Case Statement**

Another way to control the program flow is to use the Select Case control structure. The Select Case control structure is slightly different from the If....Else If control structure. The main difference is that the Select Case control structure only make decision on one expression or dimension, while the If ...Else If statement control structure may evaluate only one expression, each If....Else If statement may also compute entirely different dimensions. Select Case is preferred when there exist many different conditions because using If...Then..ElseIf statements might become too messy. The syntax for select case structure is shown below.

```

Select Case test expression
  Case expression list 1
    Block of one or more statements
  Case expression list 2
    Block of one or more Statements
  Case expression list 3
    Block of one or more statements
  Case expression list 4
    .
  Case Else
    Block of one or more VB Statements
End Select

```

In the select case statement it is possible to use the IS Keyword along with the comparison operators For Example

```

Select Case mark
Case Is >= 85
Textbox1.Text= "Excellence"

```

It is possible to use Select Case statement with numbers, as well, and the To word comes in very handy here. If you were checking a variable to see if the number that was in the variable fell within a certain range, you could use something like this:



**Select Case mark**

**Case 0 to 40**

**Textbox1.Text = "Fail"**

Here, a variable called mark is being tested. It's being checked to see if it falls between certain values. If it does, then a message box is displayed.

(Or)

**b) Illustrate the Looping statements with example**

The simple statements that had discussed in the previous section are used to execute the statements only once. If suppose a programmer needs to execute the same statements multiple times. Then some other statement should be used. The solution for this is a looping statement. . For example, to print the string "Visual basic" five times. The same statement should be used five times in the coding. But, imagine if the no. of time increases to print 1000 times or N times. By using the looping statements a statement or set of statements can be executed repeatedly.

Visual Basic 2010 allows a procedure to be repeated as many times as long as the processor and memory could support. This is generally called looping. In Visual Basic 2010, different looping statements are available such as Do Loop, For Loops etc.

**Do Looping Statements**

Looping is a very useful feature of Visual Basic because it makes repetitive works easier. Do Loop statements provide a way to continue iterating while one or more conditions are true? Many types of Do Loops exists in Visual Basic.Net such as

1. Do...While Statement
2. Do...Loop While Statement
3. Do...Until Statement
4. Do...Loop Until

**Do... While Statement**

The Do loop executes a block of statements either until a condition becomes true or while a condition remains true. The condition can be tested at the beginning or at the end of each iteration. If the test is performed at the end of each iteration, the block of statements is guaranteed to execute at least once. The Do loop can also be written without any conditions, in which case it executes repeatedly until and unless an Exit Do statement is executed within the body of the loop. The general form of Do...While Statement is

Do while condition

Block of one or more statements

Loop

### **Program to demonstrate the use of Do while Loop**

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

```
Dim counter as Integer
```

```
Counter =1
```

```
Do while counter <=100
```

```
    TextBox1.Text=counter
```

```
    Counter +=1
```

```
Loop
```

```
End Sub
```

This program prints the numbers from 1 to 100 in the textbox. Once it reach 100, the loop terminates.

### **Do...Loop While Statement**

It is another format of Do statement. Here the statements are executed while the condition is satisfied. The format is given below.

```
Do
```

```
Block of one or more statements
```

```
Loop While condition
```

**Program to demonstrate the use of Do Loop While statement. The code that is used in the previous example is rewritten here**

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

```
Counter=1
```

```
Do
```

```
    TextBox1.Text=counter
```

```
    counter+=1
```

```
Loop While counter>100
```

```
End Sub
```

### **Do Until Statement**

Another format of Do Loops is Do ... Until statement. There is no much difference between the two looping statements. The format is given below

```
Do Until condition
```

```
Block of one or more statements
```

```
Loop
```

### **Program to demonstrate the use of Do Until statement.**

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

```
Do Until number > 5
```

```
MsgBox number
```

**number = number + 1**

**Loop**

End Sub

The above program displays the numbers from 1 to 5 in the message box when user clicks the button that is placed in the form.

### **Do Loop Until Statement**

The format for this statement is

Do

Block of one or more statements

Loop Until condition

### **Program to demonstrate the use of Do Loop Until statement**

Private Sub Button1\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click

**Do**

**MsgBox number**

**number = number + 1**

**Loop Until number >= 5**

End Sub

### **For Loop Statement**

If the programmer wants to execute a block of statements to certain number of times in VB.NET program, then For Loop can be used. The For loop requires the user to provide a top index, a bottom index, and also allows an optional step. The For loop is a powerful and expressive way to iterate through numbers.

### **For...Next Statement**

The For loop executes a block of statements a specified number of times. The number of iterations is controlled by a loop variable, which is initialized to a certain value by the For statement, then is incremented for each iteration of the loop. The statements in the body of the loop are repeatedly executed until the loop variable exceeds a given upper bound.

For variable = expression To expression [ Step expression ]

Statements

Next [ variable\_list ]

The loop variable can be of any numeric type. The variable is set equal to the value of the first expression before entering the first iteration of the loop body. Prior to executing each iteration of the loop, the loop variable is compared with the value of the

second expression. If the value of the loop variable is greater than the expression (or less than the expression if the step expression is negative), the loop exits and execution continues with the first statement following the Next statement

The step expression is a numeric value that is added to the loop variable between loop iterations. If the Step clause is omitted, the step expression is taken to be 1. The Next statement marks the end of the loop body. The Next keyword can either appear by itself in the statement or be followed by the name of the loop variable. If For statements are nested, a single Next statement can terminate the bodies of multiple loops. For example:

Program to print the numbers from 1 to 10 using For Loop

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

```
    Dim i as Integer
```

```
    For i=1 to 10
```

```
        MsgBox i
```

```
    Next
```

```
End Sub
```

This program will print the numbers from 1 to 10 in the message box.

Program to print the numbers from 1 to 100 using For Loop and Step

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

```
Dim i , sum As Integer
```

```
For i=1 to 100 step 10
```

```
    sum+=i
```

```
    MsgBox sum
```

```
Next
```

```
End Sub
```

The program will calculate the sum of the numbers as follows:  
sum=0+10+20+30+40+.....

### **For...Each...Next Statement**

The For Each statement is similar to the For statement, except that the loop variable need not be numeric, and successive iterations do not increment the loop variable. Instead, the loop variable takes successive values from a collection of values. Here is the syntax

```
For Each variable In expression
```

```
    Statements
```

```
Next [ variable ]
```

The loop variable can be of any type. The expression must be a reference to an object that exposes the IEnumerable interface

```

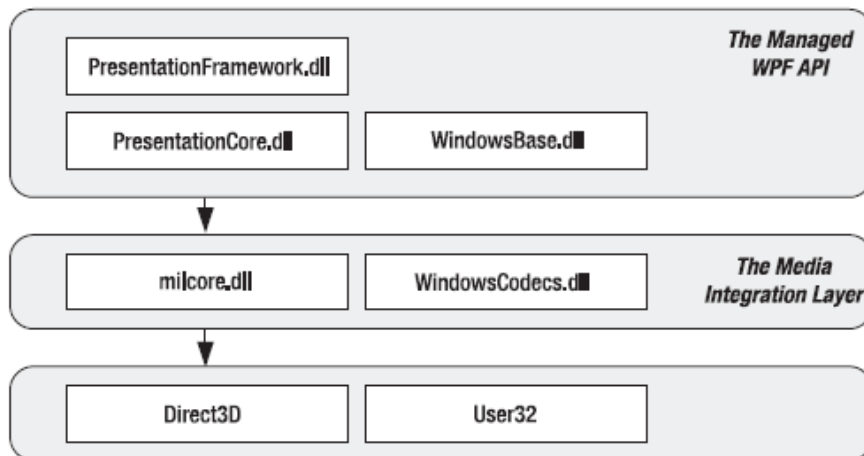
Program to print the numbers from 1 to 100 using For Loop and Step
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
Dim d As Integer
For d = 1 To 100 step 10
Textbox1.Text= d
Next d
End Sub

```

## 25. a) Enlighten the Architecture of WPF with neat diagram.

### THE ARCHITECTURE OF WPF

WPF uses a multilayered architecture. At the top, your application interacts with a high-level set of services that are completely written in managed C# code. The actual work of translating .NET objects into Direct3D textures and triangles happens behind the scenes, using a lowerlevel unmanaged component called milcore.dll.



Architecture of WPF

- **PresentationFramework.dll** holds the top-level WPF types, including those that represent windows, panels, and other types of controls. It also implements higher-level programming abstractions such as styles. Most of the classes used directly come from this assembly.
- **PresentationCore.dll** holds base types, such as UIElement and Visual, from which all shapes and controls derive
- **WindowsBase.dll** holds even more basic ingredients that have the potential to be reused outside of WPF, such as DispatcherObject and DependencyObject, which introduces the plumbing for dependency properties
- **milcore.dll** is the core of the WPF rendering system and the foundation of the Media Integration Layer (MIL). Its composition engine translates visual elements into the triangle and textures that Direct3D expects. Although milcore.dll is considered a part of WPF, it's also an essential system component for Windows

Vista. In fact, the Desktop Window Manager (DWM) in Windows Vista uses `milcore.dll` to render the desktop.

- **WindowsCodecs.dll** is a low-level API that provides imaging support (for example, processing, displaying, and scaling bitmaps and JPEGs).
- **Direct3D** is the low-level API through which all the graphics in a WPF are rendered.
- **User32** is used to determine what program gets what real estate. As a result, it's still involved in WPF, but it plays no part in rendering common controls.

(Or)

### c) Discuss XAML

XAML (short for Extensible Application Markup Language, and pronounced “zammel”) is a markup language used to instantiate .NET objects. Although XAML is a technology that can be applied to many different problem domains, its primary role in life is to construct WPF user interfaces. In other words, XAML documents define the arrangement of panels, buttons, and controls that make up the windows in a WPF application.

### UNDERSTANDING XAML

Developers realized long ago that the most efficient way to tackle complex, graphically rich applications is to separate the graphical portion from the underlying code. That way, artists can own the graphics and developers can own the code. Both pieces can be designed and refined separately, without any versioning headaches.

### XAML BASICS

The XAML standard is quite straightforward once you understand a few ground rules:

- Every element in a XAML document maps to an instance of a .NET class. The name of the element matches the name of the class exactly. For example, the element `<Button>` instructs WPF to create a `Button` object.
- As with any XML document, you can nest one element inside another. As you'll see, XAML gives every class the flexibility to decide how it handles this situation. However, nesting is usually a way to express containment—in other words, if you find a `Button` element inside a `Grid` element, your user interface probably includes a `Grid` that contains a `Button` inside.
- You can set the properties of each class through attributes. However, in some situations an attribute isn't powerful enough to handle the job. In these cases, you'll use nested tags with a special syntax

Before continuing, take a look at this bare-bones XAML document, which represents a new blank window (as created by Visual Studio). The lines have been numbered for easy reference:

```
<Window x:Class="Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Window1" Height="300" Width="300">
```

```
<Grid>
</Grid>
</Window>
```

This document includes only two elements—the top-level Window element, which represents the entire window, and the Grid, in which you can place all your controls. Although you could use any top-level element, WPF applications rely on just a few:

- Window
- Page (which is similar to Window, but used for navigable applications)
- Application (which defines application resources and startup settings)

As in all XML documents, there can only be one top-level element. In the previous example, that means that as soon as you close the Window element with the `</Window>` tag, you end the document. No more content can follow.

Looking at the start tag for the Window element you'll find several interesting attributes, including a class name and two XML namespaces (described in the following sections). You'll also find the three properties shown here:

```
Title="Window1" Height="300" Width="300">
```

Each attribute corresponds to a separate property of the Window class. All in all, this tells WPF to create a window with the caption Window1 and to make it 300 by 300 units large.

## XAML Namespaces

Clearly, it's not enough to supply just a class name. The XAML parser also needs to know the .NET namespace where this class is located. For example, the Window class could exist in several places—it might refer to the `system.Windows.Window` class, or it could refer to a Window class in a third-party component, or one you've defined in your application. To figure out which class you really want, the XAML parser examines the XML namespace that's applied to the element. Here's how it works. In the sample document shown earlier, two namespaces are defined:

```
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

The `xmlns` attribute is a specialized attribute in the world of XML that's reserved for declaring namespaces. This snippet of markup declares two namespaces that you'll find in every WPF XAML document you create:

- `http://schemas.microsoft.com/winfx/2006/xaml/presentation` is the core WPF namespace. It encompasses all the WPF classes, including the controls you use to build user interfaces. In this example, this namespace is declared without a namespace prefix, so it becomes the default namespace for the entire document.

In other words, every element is automatically placed in this namespace unless you specify otherwise.

- <http://schemas.microsoft.com/winfx/2006/xaml> is the XAML namespace. It includes various XAML utility features that allow you to influence how your document is interpreted. This namespace is mapped to the prefix `x`. That means you can apply it by placing the namespace prefix before the element name (as in `<x:ElementName>`).

As you can see, the XML namespace name doesn't match any particular .NET namespace. There are a couple of reasons the creators of XAML chose this design. By convention, XML namespaces are often URIs (as they are here). These URIs look like they point to a location on the Web, but they don't. The URI format is used because it makes it unlikely that different organizations will inadvertently create different XML-based languages with the same namespace. Because the domain `schemas.microsoft.com` is owned by Microsoft, only Microsoft will use it in an XML namespace name.

The other reason that there isn't a one-to-one mapping between the XML namespaces used in XAML and .NET namespaces is because it would significantly complicate your XAML documents. The problem here is that WPF encompasses well over a dozen namespaces (all of which start with `System.Windows`). If each .NET namespace had a different XML namespace, you'd need to specify the right namespace for each and every control you use, which quickly gets messy. Instead, the creators of WPF chose to combine all of these .NET namespaces into a single XML namespace. This works because within the different .NET namespaces that are a part of WPF, there aren't any classes that have the same name. The namespace information allows the XAML parser to find the right class. For example, when it looks at the `Window` and `Grid` elements, it sees that they are placed in the default WPF namespace. It then searches the corresponding .NET namespaces, until it finds `System.Windows`. `Window` and `system.Windows.Controls.Grid`.

## The Code-Behind Class

XAML allows you to construct a user interface, but in order to make a functioning application you need a way to connect the event handlers that contain your application code. XAML makes this easy using the `Class` attribute that's shown here:

```
<Window x:Class="Window1"
```

The `x` namespace prefix places the `Class` attribute in the XAML namespace, which means this is a more general part of the XAML language. In fact, the `Class` attribute tells the XAML parser to generate a new class with the specified name. That class derives from the class that's named by the XML element. In other words, this example creates a new class named `Window1`, which derives from the base `Window` class. The `Window1` class is generated automatically at compile time. But here's where things get interesting. You can supply a piece of the `Window1` class that will be merged into the automatically generated portion. The piece you specify is the perfect container for your event handling code.



When you compile your application, the XAML that defines your user interface (such as Window1.xaml) is translated into CLR type declaration that is merged with the logic in your code-behind class file (such as Window1.xaml.vb) to form one single unit. Initially, the code behind class that Visual Studio creates is empty:

```
Class Window1
End Class
```

### The InitializeComponent() Method

Currently, the Window1 class code doesn't include any code. However, there's one detail that's completely hidden—the default constructor. If you could see the constructor that Visual Studio generates at compile time, it would look like this:

```
Public Sub New()
InitializeComponent()
End Sub
```

The InitializeComponent() method is another piece of automatically generated code that doesn't appear in your class. It loads the BAML (the compiled XAML) from your assembly and uses it to build your user interface. As it parses the BAML, it creates each control object, sets its properties, and attaches any event handlers. At this point, you're probably wondering why it's important to understand a detail that Visual Studio hides from you completely. The reason becomes apparent if you create a custom constructor for your window. In this case, you need to explicitly call the InitializeComponent() method. If you don't, the window won't be initialized, and none of your controls will appear.

## 26. a) Explain the Content controls and text controls with example

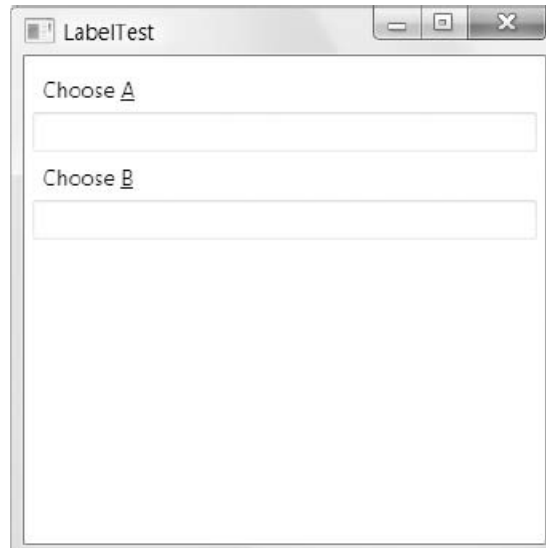
### CONTENT CONTROLS

These include the well-worn Label, Button, CheckBox, and RadioButton.

#### Labels

The simplest of all content controls is the Label control. Like any other content control, it accepts any single piece of content you want to place inside. But what distinguishes the Label control is its support for *mnemonics*—essentially, shortcut keys that set the focus to a linked control. To support this functionality, the Label control adds a single property, named Target. To set the Target property, you need to use a binding expression that points to another control. Here's the syntax you must use:

```
<Label Target="{Binding ElementName=txtA}">Choose _A</Label>
<TextBox Name="txtA"></TextBox>
<Label Target="{Binding ElementName=txtB}">Choose _B</Label>
<TextBox Name="txtB"></TextBox>
```



## Buttons

WPF recognizes three types of button controls: the familiar `Button`, the `CheckBox`, and the `RadioButton`. All of these controls are content controls that derive from `ButtonBase`. The `ButtonBase` class includes only a few members. It defines the `Click` event and adds support for commands, which allow you to wire buttons to higher-level application tasks.

The `ButtonBase` class adds a `ClickMode` property, which determines when a button fires its `Click` event in response to mouse actions. The default value is `ClickMode.Release`, which means the `Click` event fires when the mouse is clicked and released. However, you can also choose to fire the `Click` event mouse when the mouse button is first pressed (`ClickMode.Press`) or, oddly enough, whenever the mouse moves over the button and pauses there (`ClickMode.Hover`).

### *The Button*

The `Button` class represents the ever-present Windows push button. It adds just two writeable properties, `IsCancel` and `IsDefault`:

- **When `IsCancel` is `True`**, this button is designated as the cancel button for a window. If you press the Escape key while positioned anywhere on the current window, this button is triggered.
- **When `IsDefault` is `True`**, this button is designated as the default button (also known as the accept button). If Enter key is pressed, this button is triggered.

### *The ToggleButton and RepeatButton*

Alongside `Button`, three more classes derive from `ButtonBase`. These include the following:

- `GridViewColumnHeader`, which represents the clickable header of a column when you use a grid-based `ListView`.
- `RepeatButton`, which fires `Click` events continuously, as long as the button is held down. Ordinary buttons fire one `Click` event per user click.
- `ToggleButton`, which represents a button that has two states (pushed or unpushed). When you click a `ToggleButton`, it stays in its pushed state until you click it again to release it. This is sometimes described as “sticky click” behavior.

### ***The CheckBox***

Both the `CheckBox` and the `RadioButton` are buttons of a different sort. They derive from `ToggleButton`, which means they can be switched on or off by the user, hence their “toggle” behavior. In the case of the `CheckBox`, switching the control “on” means placing a check mark in it.

To assign a null value in WPF markup, you need to use the null markup extension, as shown here:

```
<CheckBox IsChecked="{x:Null}">A check box in indeterminate state</CheckBox>
```

The `ToggleButton` class also defines three events that fire when the check box enters specific states: `Checked`, `Unchecked`, and `Indeterminate`.

### ***The RadioButton***

The `RadioButton` also derives from `ToggleButton` and uses the same `IsChecked` property and the same `Checked`, `Unchecked`, and `Indeterminate` events. Along with these, the `RadioButton` adds a single property named `GroupName`, which allows you to control how radio buttons are placed into groups. Ordinarily, radio buttons are grouped by their container. That means if you place three `RadioButton` controls in a single `StackPanel`, they form a group from which you can select just one of the three. On the other hand, if you place a combination of radio buttons in two separate `StackPanel` controls, you have two independent groups.

```
<StackPanel>
  <GroupBox Margin="5">
    <StackPanel>
      <RadioButton>Group 1</RadioButton>
      <RadioButton>Group 1</RadioButton>
      <RadioButton>Group 1</RadioButton>
      <RadioButton Margin="0,10,0,0" GroupName="Group2">Group 2</RadioButton>
    </StackPanel>
  </GroupBox>
  <GroupBox Margin="5">
    <StackPanel>
      <RadioButton>Group 3</RadioButton>
      <RadioButton>Group 3</RadioButton>
      <RadioButton>Group 3</RadioButton>
      <RadioButton Margin="0,10,0,0" GroupName="Group2">Group 2</RadioButton>
    </StackPanel>
  </GroupBox>
</StackPanel>
```

Here, there are two containers holding radio buttons, but three groups. The final radio button at the bottom of each group box is part of a third group.

### **Tooltips**

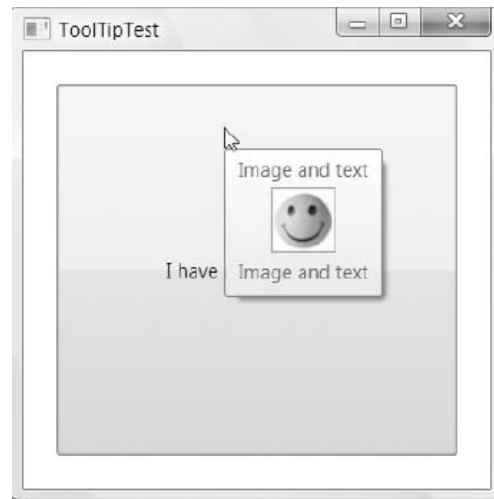
WPF has a flexible model for *tooltips*. The easiest way to show a tooltip doesn’t involve using the `ToolTip` class directly. Instead, set the `ToolTip` property of the element. The `ToolTip` property is defined in the `FrameworkElement` class, so it’s available on anything in a WPF window. For example, here’s a button that has a basic tooltip:

```
<Button ToolTip="This is my tooltip">I have a tooltip</Button>
```

```

<Button>
  <Button.ToolTip>
    <StackPanel>
      <TextBlock Margin="3" >Image and text</TextBlock>
      <Image Source="happyface.jpg" Stretch="None" />
      <TextBlock Margin="3" >Image and text</TextBlock>
    </StackPanel>
  </Button.ToolTip>
  <Button.Content>I have a fancy tooltip</Button.Content>
</Button>

```



### ***Setting ToolTip Properties***

<b>Name</b>	<b>Description</b>
HasDropShadow	Determines whether the tooltip has a diffuse black drop shadow that makes it stand out from the window underneath.
Placement	Determines how the tooltip is positioned, using one of the values from the PlacementMode enumeration. The default value is Mouse, which means that the top-left corner of the tooltip is placed relative to the current mouse position. (The actual position of the tooltip may be offset from this starting point based on the HorizontalOffset and VerticalOffset properties.) Other possibilities allow you to place the tooltip using absolute screen coordinates or place it relative to some element (which you indicate using the PlacementTarget property).
HorizontalOffset and VerticalOffset	Allows you to nudge the tooltip into the exact position you want. You can use positive or negative values.
PlacementTarget	Allows you to place a tooltip relative to another element. In order to use this property, the Placement property must be set to Left, Right, Top, Bottom, or Center. (This is the edge of the element to which the tooltip is aligned.)
PlacementRectangle	Allows you to offset the position of the tooltip. This works in much the same way as the HorizontalOffset and VerticalOffset properties. This property doesn't have an effect if Placement property is set to Mouse.

Name	Description
CustomPopupPlacementCallback	Allows you to position a tooltip dynamically using code. If the Placement property is set to Custom, this property identifies the method that will be called by the ToolTip to get the position where the ToolTip should be placed. Your callback method receives three pieces of information—popupSize (the size of the ToolTip), targetSize (the size of the Placement-Target, if it's used), and offset (a point that's created based on HorizontalOffset and VerticalOffset properties). The method returns a CustomPopupPlacement object that tells WPF where to place the tooltip.
StaysOpen	Has no effect in practice. The intended purpose of this property is to allow you to create a tooltip that remains open until the user clicks somewhere else. However, the ToolTipService.ShowDuration property overrides the StaysOpen property. As a result, tooltips always disappear after a configurable amount of time (usually about 5 seconds) or when the user moves the mouse away. If you want to create a tooltip-like window that stays open indefinitely, the easiest approach is to use the Popup control.

Using the ToolTip properties, the following markup creates a tooltip that has no drop shadow but uses a transparent red background that lets the underlying window (and controls) show through:

```
<Button>
  <Button.ToolTip>
    <ToolTip Background="#60AA4030" Foreground="White"
      HasDropShadow="False" >
      <StackPanel>
        <TextBlock Margin="3" >Image and text</TextBlock>
        <Image Source="happyface.jpg" Stretch="None" />
        <TextBlock Margin="3" >Image and text</TextBlock>
      </StackPanel>
    </ToolTip>
  </Button.ToolTip>
  <Button.Content>I have a fancy tooltip</Button.Content>
</Button>
```

Here are some strategies you can use to place a tooltip:

- **Based on the current position of the mouse.** This is the standard behavior, which relies on Placement being set to Mouse. The top-left corner of the tooltip box is lined up with the bottom-left corner of the invisible “bounding box” around the mouse pointer.
- **Based on the position of the moused-over element.** Set the Placement property to Left, Right, Top, Bottom, or Center, depending on the edge of the element. The top-left corner of the tooltip box will be lined up with that edge.

- **Based on the position of another element (or the window).** Set the Placement property in the same way you would if you were lining the tooltip up with the current element. (Use the value Left, Right, Top, Bottom, or Center.) Then choose the element by setting the PlacementTarget property. Remember to use the {Binding ElementName=*Name*} syntax to identify the element you want to use.
- **With an offset.** Use any of the strategies described previously, but set the HorizontalOffset and VerticalOffset properties to add a little extra space.
- **Using absolute coordinates.** Set Placement to Absolute and use the HorizontalOffset and VerticalOffset properties (or the PlacementRectangle) to set some space between the tooltip and the top-left corner of the window.
- **Using a calculation at runtime.** Set Placement to Custom. Set the CustomPopup PlacementCallback property to point to a method that created.

### *Setting ToolTipService Properties*

There are some tooltip properties that can't be configured using the properties of the ToolTip class. In this case, use a different class, which is named ToolTipService. ToolTipService allows to configure the time delays associated with the display of a tooltip. All the properties of the ToolTipService class are attached properties, so set them directly in the control tag, as shown here:

```
<Button ToolTipService.InitialShowDelay="1">
...
</Button>
```

The ToolTipService class defines many of the same properties as ToolTip. This allows to use a simpler syntax when dealing with text-only tooltips. Rather than adding a nested ToolTip element, set everything using attributes:

```
<Button ToolTip="This tooltip is aligned with the bottom edge"
ToolTipService.Placement="Bottom">I have a tooltip</Button>
```

### *The Popup*

The Popup control has a great deal in common with the ToolTip, although neither one derives from the other. Like the ToolTip, the Popup can hold a single piece of content, which can include any WPF element. The differences between the Popup and ToolTip are more important. They include the following:

- The Popup is never shown automatically. Set the IsOpen property for it to appear.
- By default, the Popup.StaysOpen property is set to True, and the Popup does not disappear until explicitly set its IsOpen property to False. If you set StaysOpen to False, the Popup disappears when the user clicks somewhere else.
- The Popup provides a PopupAnimation property that lets you control how it comes into view when you set IsOpen to True. Your options include None (the default), Fade (the opacity of the popup gradually increases), Scroll (the popup slides in from the upperleft corner of the window, space permitting), and Slide (the popup slides down into place, space permitting). In order for any of these animations to work, you must also set the AllowsTransparency property to True.
- The Popup can accept focus. Thus, you can place user-interactive controls in it, such as a Button. This functionality is one of the key reasons to use the Popup instead of the ToolTip.

- The Popup control is defined in the `System.Windows.Controls.Primitives` namespace because it is most commonly used as a building block for more complex controls.

## TEXT CONTROLS

WPF includes three text-entry controls: `TextBox`, `RichTextBox`, and `PasswordBox`. The `PasswordBox` derives directly from `Control`. The `TextBox` and `RichTextBox` controls go through another level and derive from `TextBoxBase`.

## Multiple Lines of Text

Ordinarily, the `TextBox` control stores a single line of text. (You can limit the allowed number of characters by setting the `MaxLength` property.) However, there are many cases to create a multiline text box for dealing with large amounts of content. In this case, set the `TextWrapping` property to `Wrap` or `WrapWithOverflow`. `Wrap` always breaks at the edge of the control, even if it means severing an extremely long word in two. `WrapWithOverflow` allows some lines to stretch beyond the right edge if the line-break algorithm.

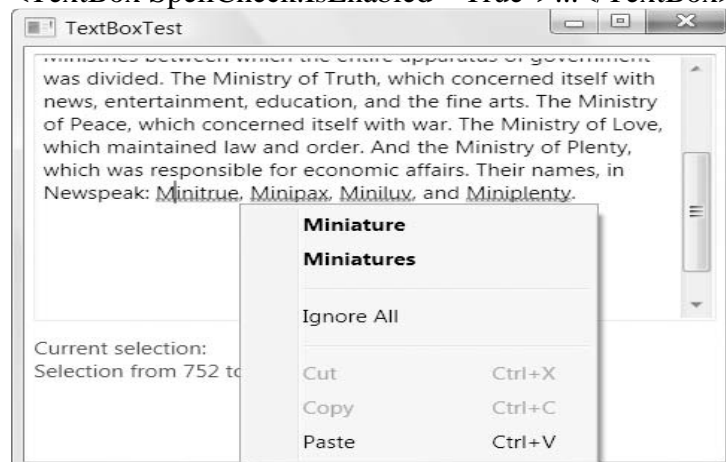
## Text Selection

The `TextBox` class also gives the ability to determine or change the currently selected text programmatically, using the `SelectionStart`, `SelectionLength`, and `SelectedText` properties. `SelectionStart` identifies the zero-based position where the selection begins. For example, if you set this property to 10, the first selected character is the 11th character in the text box. The `SelectionLength` indicates the total number of selected characters. (A value of 0 indicates no selected characters.) Finally, the `SelectedText` property allows you to quickly examine or change the selected text in the text box.

## Miscellaneous TextBox Features

The TextBox includes a few more specialized frills. The most interesting is the spelling-checker feature, which underlines unrecognized words with a red squiggly line. The user can rightclick an unrecognized word and choose from a list of possibilities. To turn on this spelling-checker functionality for the TextBox control, set the SpellCheck.IsEnabled dependency property, as shown here:

```
<TextBox SpellCheck.IsEnabled="True">...</TextBox>
```



## The PasswordBox

The PasswordBox looks like a TextBox, but it displays a string of circle symbols to mask the characters it shows. (choose a different mask character by setting the PasswordChar property.) Additionally, the PasswordBox does not support the clipboard, so you can't copy the text inside.

(Or)

**b) Discuss List controls and range-based controls with example**

**LIST CONTROLS**

WPF includes many controls that wrap a collection of items, ranging from the simple ListBox and ComboBox. The class hierarchy that leads from ItemsControls is a bit tangled. One major branch is the *selectors*, which includes the ListBox, the ComboBox, and the TabControl. These controls derive from Selector and have properties that let you track down the currently selected item (SelectedItem) or its position (SelectedIndex).

**The ListBox**

The ListBox and ComboBox class represent two common staples of Windows design—variable-length lists that allow the user to select an item. To add items to the ListBox, you can nest ListBoxItem elements inside the ListBox element. For example, here's a ListBox that contains a list of colors:

```
<ListBox>
  <ListBoxItem>Green</ListBoxItem>
  <ListBoxItem>Blue</ListBoxItem>
  <ListBoxItem>Yellow</ListBoxItem>
  <ListBoxItem>Red</ListBoxItem>
</ListBox>
```

For example, if you decided you want to create a list with images, you could create markup like this:

```
<ListBox>
  <ListBoxItem>
    <Image Source="happyface.jpg"></Image>
  </ListBoxItem>
  <ListBoxItem>
    <Image Source="happyface.jpg"></Image>
  </ListBoxItem>
</ListBox>
```

Here's a more ambitious example that uses nested StackPanel objects to combine text and image content:



```

<ListBox>
  <StackPanel Orientation="Horizontal">
    <Image Source="happyface.jpg" Width="30" Height="30"></Image>
    <Label VerticalContentAlignment="Center">A happy face</Label>
  </StackPanel>
  <StackPanel Orientation="Horizontal">
    <Image Source="redx.jpg" Width="30" Height="30"></Image>
    <Label VerticalContentAlignment="Center">A warning sign</Label>
  </StackPanel>
  <StackPanel Orientation="Horizontal">
    <Image Source="happyface.jpg" Width="30" Height="30"></Image>
    <Label VerticalContentAlignment="Center">A happy face</Label>
  </StackPanel>
</ListBox>

```



### The ComboBox

The ComboBox is similar to the ListBox control. It holds a collection of ComboBoxItem objects, which are created either implicitly or explicitly. As with the ListBoxItem, the ComboBoxItem is a content control that can contain any nested element. The ComboBox control uses a drop-down list, which means only one item can be selected at a time.

If you want to allow the user to type in text in the combo box to select an item, you must set the IsEditable property to True.

### RANGE-BASED CONTROLS

WPF includes three controls that use the concept of a *range*. These controls take a numeric value that falls in between a specific minimum and maximum value. These

controls— ScrollBar, ProgressBar, and Slider—all derive from the RangeBase class (which itself derives from the Control class).

Name	Description
Value	This is the current value of the control (which must fall between the minimum and maximum). By default, it starts at 0.
Maximum	This is the upper limit (the largest allowed value).
Minimum	This is the lower limit (the smallest allowed value).
SmallChange	This is the amount the Value property is adjusted up or down for a “small change.” The meaning of a small change depends on the control (and may not be used at all). For the ScrollBar and Slider, this is the amount the value changes when you use the arrow keys. For the ScrollBar, you can also use the arrow buttons at either end of the bar.
LargeChange	This is the amount the Value property is adjusted up or down for a “large change.” The meaning of a large change depends on the control (and may not be used at all). For the ScrollBar and Slider, this is the amount the value changes when you use the Page Up and Page Down keys or when you click the bar on either side of the thumb (which indicates the current position).

## The ScrollBar

The ScrollViewer control provides a convenient way to enable scrolling of content in Windows Presentation Foundation (WPF) applications. There are two predefined elements that enable scrolling in WPF applications: ScrollBar and ScrollViewer. The ScrollViewer control encapsulates horizontal and vertical ScrollBar elements and a content container (such as a Panel element) in order to display other visible elements in a scrollable area. You must build a custom object in order to use the ScrollBar element for content scrolling. However, you can use the ScrollViewer element by itself because it is a composite control that encapsulates ScrollBar functionality.

The ScrollViewer control responds to both mouse and keyboard commands, and defines numerous methods with which to scroll content by predetermined increments. User can use the ScrollChanged event to detect a change in a ScrollViewer state. A ScrollViewer can only have one child, typically a Panel element that can host a Children collection of elements. The Content property defines the sole child of the ScrollViewer.

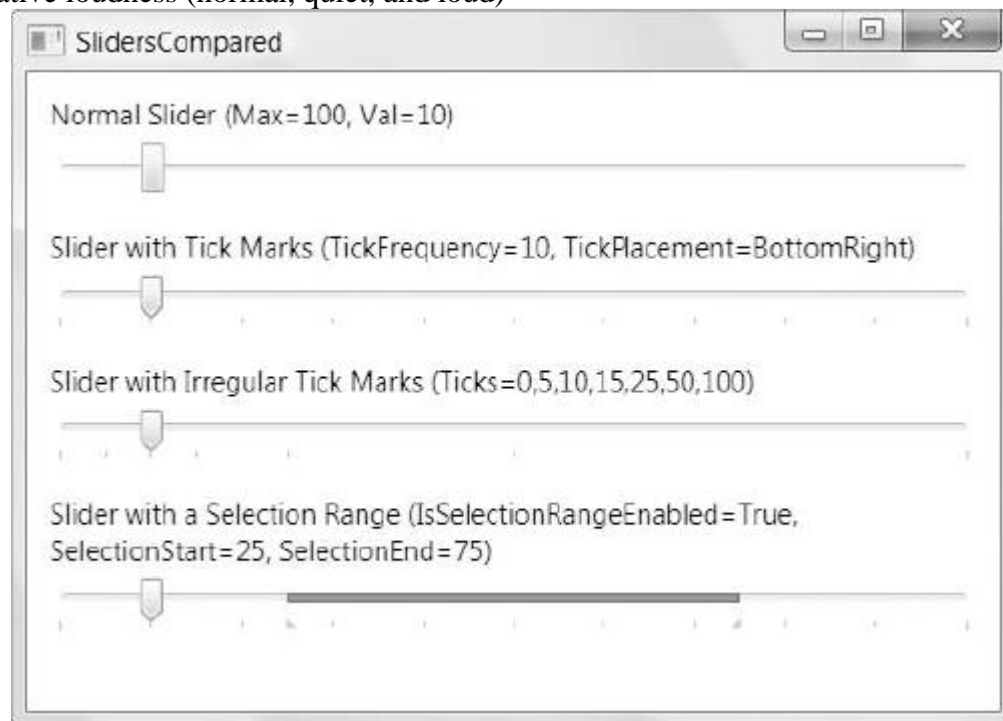
```

<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      WindowTitle="ScrollViewer Sample">
  <ScrollViewer HorizontalScrollBarVisibility="Auto">
    <StackPanel VerticalAlignment="Top" HorizontalAlignment="Left">
      <TextBlock TextWrapping="Wrap" Margin="0,0,0,20">Scrolling is enabled
when it is necessary.
      Resize the window, making it larger and smaller.</TextBlock>
      <Rectangle Fill="Red" Width="500" Height="500"></Rectangle>
    </StackPanel>
  </ScrollViewer>
</Page>

```

## The Slider

The Slider is a specialized control that's occasionally useful—for example, you might use it to set numeric values in situations where the number itself isn't particularly significant. For example, it makes sense to set the volume in a media player by dragging the thumb in a slider bar from side to side. The general position of the thumb indicates the relative loudness (normal, quiet, and loud)



Name	Description
Orientation	Switches between a vertical and a horizontal slider.
Delay and Interval	Controls how fast the thumb moves along the track when you click and hold down either side of the slider. Both are millisecond values. The Delay is the time before the thumb moves one (small change) unit after you click, and the Interval is the time before it moves again if you continue holding the mouse button down.
TickPlacement	Determines where the tick marks appear. (Tick marks are notches that appear near the bar to help you visualize the scale.) By default, the TickPlacement is set to None, and no tick marks appear. If you have a horizontal slider, you can place the tick marks above (TopLeft) or below (BottomRight) the track. With a vertical slider, you can place them on the left (TopLeft) and right (BottomRight). (The TickPlacement names are a bit confusing because two values cover four possibilities, depending on the orientation of the slider.)
TickFrequency	Sets the interval in between ticks, which determines how many ticks appear. For example, you could place them every 5 numeric units, every 10, and so on.
Ticks	If you want to place ticks in specific, irregular positions, you can use the Ticks collection. Simply add one number (as a Double) to this collection for each tick mark. For example, you could place ticks at the positions 1, 1.5, 2, and 10 on the scale by adding these numbers.
IsSnapToTickEnabled	If True, when you move the slider, it automatically snaps into place, jumping to the nearest tick mark. The default is False.
IsSelectionRangeEnabled	If True, you can use a selection range to shade in a portion of the slider bar. You set the position selection range using the SelectionStart and SelectionEnd properties. The selection range has no intrinsic meaning, but you can use it for whatever purpose makes sense. For example, media players sometimes use a shaded background bar to indicate the download progress for a media file.

## The ProgressBar

The ProgressBar indicates the progress of a long-running task. Unlike the slider, the ProgressBar isn't user interactive. Instead, it's up to the code to periodically increment the Value property.

```
<ProgressBar Height="18" Width="200"IsIndeterminate="True"></ProgressBar>
```

When setting IsIndeterminate, no longer use the Minimum, Maximum, and Value properties. Instead, this ProgressBar shows a periodic green pulse that travels from left to right, which the universal Windows convention is indicating that there's work in progress. This sort of indicator makes particular sense in an application's status bar. For example, you could use it to indicate that you're contacting a remote server for information.