Visual Basic 2017 Made Easy

By

Dr.Liew

Disclaimer

Visual Basic 2017 Made Easy is an independent publication and is not affiliated with, nor has it been authorized, sponsored, or otherwise approved by Microsoft Corporation.

Trademarks

Microsoft, Visual Basic, Excel and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks belong to their respective owners.

Liability

The purpose of this book is to provide basic guides for people interested in Visual Basic 2017 programming. Although every effort and care has been taken to make The information as accurate as possible, the author shall not be liable for any error, Harm or damage arising from using the instructions given in this book.

Copyright ® 2017 Liew Voon Kiong

All rights reserved. No Part of this e-book may be reproduced, in any form or by any means, without permission in writing from the author.

Acknowledgement

I would like to express my sincere gratitude to many people who have made their contributions in one way or another to the successful publication of this book.
My special thanks go to my children Xiang, Yi and Xun who have contributed their ideas and help in editing this book. I would also like to appreciate the support provided by my beloved wife Kim Huang and my youngest daughter Yuan. I would also like to thank the millions of readers who have visited my **Visual Basic Tutorial** website at **vbtutor.net** for their support and encouragement.

About the Author

Dr. Liew Voon Kiong holds a bachelor's degree in Mathematics, a master's degree in Management and a doctorate in Business Administration. He has been involved in Visual Basic programming for more than 20 years. He created the popular online Visual Basic Tutorial at **www.vbtutor.net**, which has attracted millions of visitors since 1996. It has consistently been one of the highest ranked Visual Basic websites.

To provide more support for Visual Basic students, teachers, and hobbyists, Dr. Liew has written this book to complement the free Visual Basic 2017 tutorial with much more content. He is also the author of the Visual Basic Made Easy series, which includes **Visual Basic 6 Made Easy, Visual Basic 2008 Made Easy, Visual Basic 2010 Made Easy, Visual Basic 2013 Made Easy** and **Excel VBA Made Easy.** Dr. Liew's books have been used in high school and university computer science courses all over the world.

Contents

Chapter 1 Introduction to Visual Basic 2017

- ❖ A brief description of Visual Basic 2017
- ❖ Getting to know the Visual Basic 2017 Integrated Development Environment

1.1 A Brief Description of Visual Basic 2017

Visual Basic is a third-generation event-driven programming language first released by Microsoft in 1991. The final version of the classic Visual Basic was Visual Basic 6. Visual Basic 6 is a user-friendly programming language designed for beginners. Many developers still favor VB6 over its successor VB.NET.

In 2002, Microsoft released Visual Basic.NET(VB.NET) to replace Visual Basic 6. Thereafter, Microsoft declared VB6 a legacy programming language in 2008. However, Microsoft still provides some form of support for VB6. VB.NET is a fully object-oriented programming language implemented in the .NET Framework. It was created to cater for the development of the web as well as mobile applications.

Subsequently, Microsoft has released many versions of VB.NET. They are Visual Basic 2005, Visual Basic 2008, Visual Basic 2010, Visual Basic 2012, Visual Basic 2013, Visual Basic 2015 and Visual Basic 2017. Although the .NET portion was discarded in 2005, all versions of the Visual Basic programming language released since 2002 are regarded as VB.NET programming language

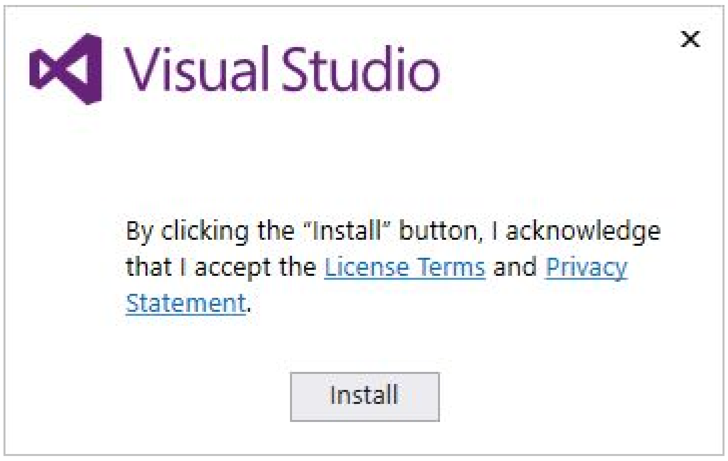
Visual Basic 2017 was released in 2017. It is bundled as a .NET desktop development component Visual Studio 2017. VB2017 can be used to build windows desktop applications using the .NET framework. Besides that, Visual Studio Community 2017 also comes with other Windows development tools that include Universal Windows Platform Development that creates applications for the Universal Windows Platform with C#, VB , JavaScript and C++. On top of that, it also includes Desktop Development with C++.

In addition, to cater for the increasing needs of web and cloud-based applications, VS2017 also provides the Web and Cloud development tools that include ASP.NET, Python, Azure SDK, Node.js, data storage and processing, data science and analytical applications as well as Office/SharePoint development. Furthermore, VS2017 also cater for the development of mobile applications by including the mobile and gaming tools like mobile development with .NET, game development with Unity, mobile development with JavaScript, mobile development with C++ and game development with C++. With the mobile development and gaming tools, you can build IOS and Android mobile apps and mobile games.

You can download Visual Studio 2017 from the following link:

https://visualstudio.microsoft.com/vs/older-downloads/

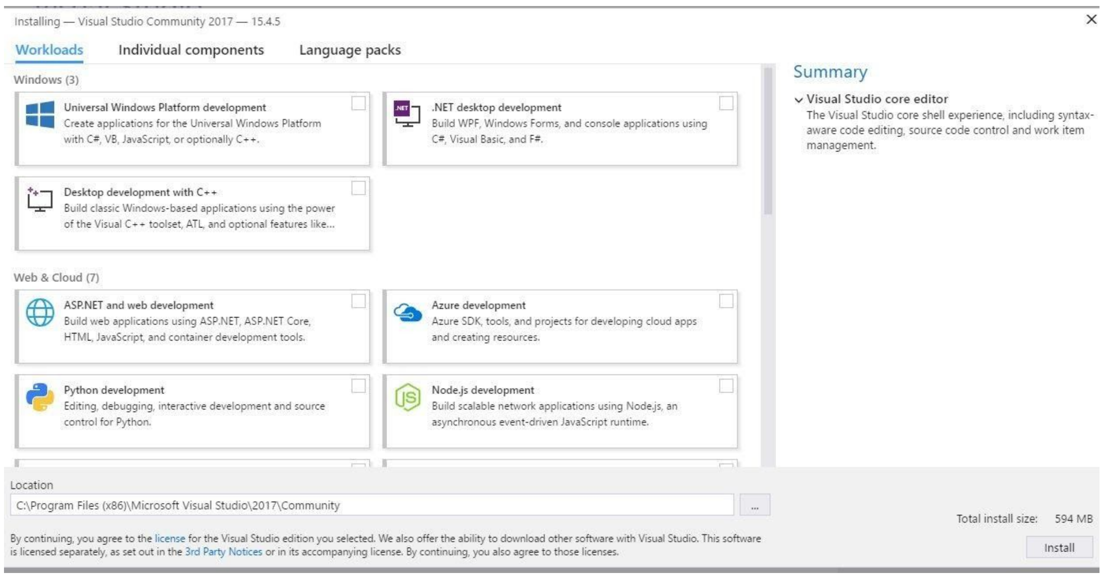
After downloading the file, run the VS2017RC community installer file. If you receive a User Account Control notice, click Yes. Next, it will ask you to acknowledge the Microsoft License Terms and the Microsoft Privacy Statement, as shown in Figure 1.1. Click Install to continue.



By clicking the "Install" button, I acknowledge that I accept the License Terms and Privacy Statement.

Figure 1.1

You will see several status screens that show the progress of the installation. After installation completed, you can select the feature set that you want, as shown in Figure 1.2. Since we are keen on developing Visual Basic 2017 desktop app, we will select the .NET desktop development component. Besides that, you might want to install a language by clicking the Language packs. After making your selections, click install.



Installing — Visual Studio Community 2017 — 15.4.5

Windows (3)

- Universal Windows Platform development — Create applications for the Universal Windows Platform with C#, VB, JavaScript, or optionally C++.
- .NET desktop development — Build WPF, Windows Forms, and console applications using C#, Visual Basic, and F#.
- Desktop development with C++ — Build classic Windows-based applications using the power of the Visual C++ toolset, ATL, and optional features like...

Web & Cloud (7)

- ASP.NET and web development — Build web applications using ASP.NET, ASP.NET Core, HTML, JavaScript, and container development tools.
- Azure development — Azure SDK, tools, and projects for developing cloud apps and creating resources.
- Python development — Editing, debugging, interactive development and source control for Python.
- Node.js development — Build scalable network applications using Node.js, an asynchronous event-driven JavaScript runtime.

Summary

Visual Studio core editor — The Visual Studio core shell experience, including syntax-aware code editing, source code control and work item management.

Location: C:\Program Files (x86)\Microsoft Visual Studio\2017\Community

By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the 3rd Party Notices or in its accompanying license. By continuing, you also agree to those licenses.

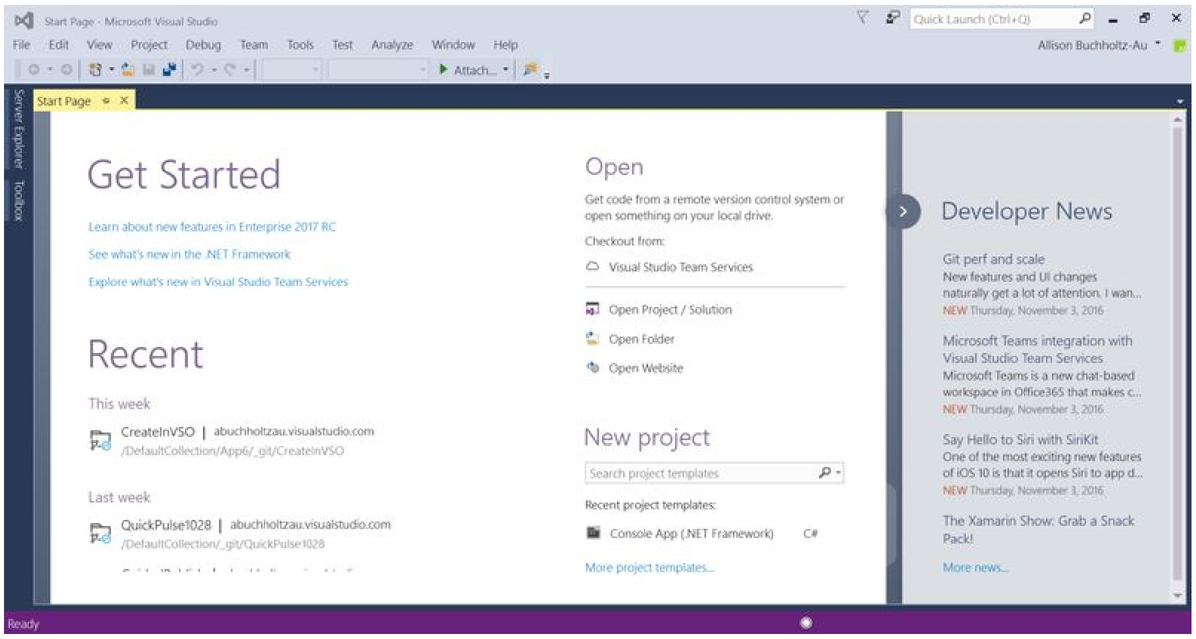
Total install size: 594 MB

Figure 1.2

Upon completion of the installation, you can start programming in Visual Basic 2017.

1.2 The Visual Studio 2017 IDE

When you launch Microsoft Visual Studio 2017, you will be presented with the Start Page of Microsoft VS 2017, as shown in Figure 1.3


Start Page - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
File Edit View Project Debug Team Tools Test Analyze Window Help
Allison Buchholtz-Au
Attach...
Server Explorer Toolbox
Start Page
Ready

Get Started

Learn about new features in Enterprise 2017 RC

See what's new in the .NET Framework

Explore what's new in Visual Studio Team Services

Recent

This week

CreateInVSO | abuchholtzau.visualstudio.com
/DefaultCollection/App6/_git/CreateInVSO

Last week

QuickPulse1028 | abuchholtzau.visualstudio.com
/DefaultCollection/_git/QuickPulse1028

Open

Get code from a remote version control system or open something on your local drive.

Checkout from:

Visual Studio Team Services

Open Project / Solution

Open Folder

Open Website

New project

Search project templates

Recent project templates:

Console App (.NET Framework) C#

More project templates...

Developer News

Git perf and scale
New features and UI changes naturally get a lot of attention. I wan...
NEW Thursday, November 3, 2016

Microsoft Teams integration with Visual Studio Team Services
Microsoft Teams is a new chat-based workspace in Office365 that makes c...
NEW Thursday, November 3, 2016

Say Hello to Siri with SiriKit
One of the most exciting new features of iOS 10 is that it opens Siri to app d...
NEW Thursday, November 3, 2016

The Xamarin Show: Grab a Snack Pack!

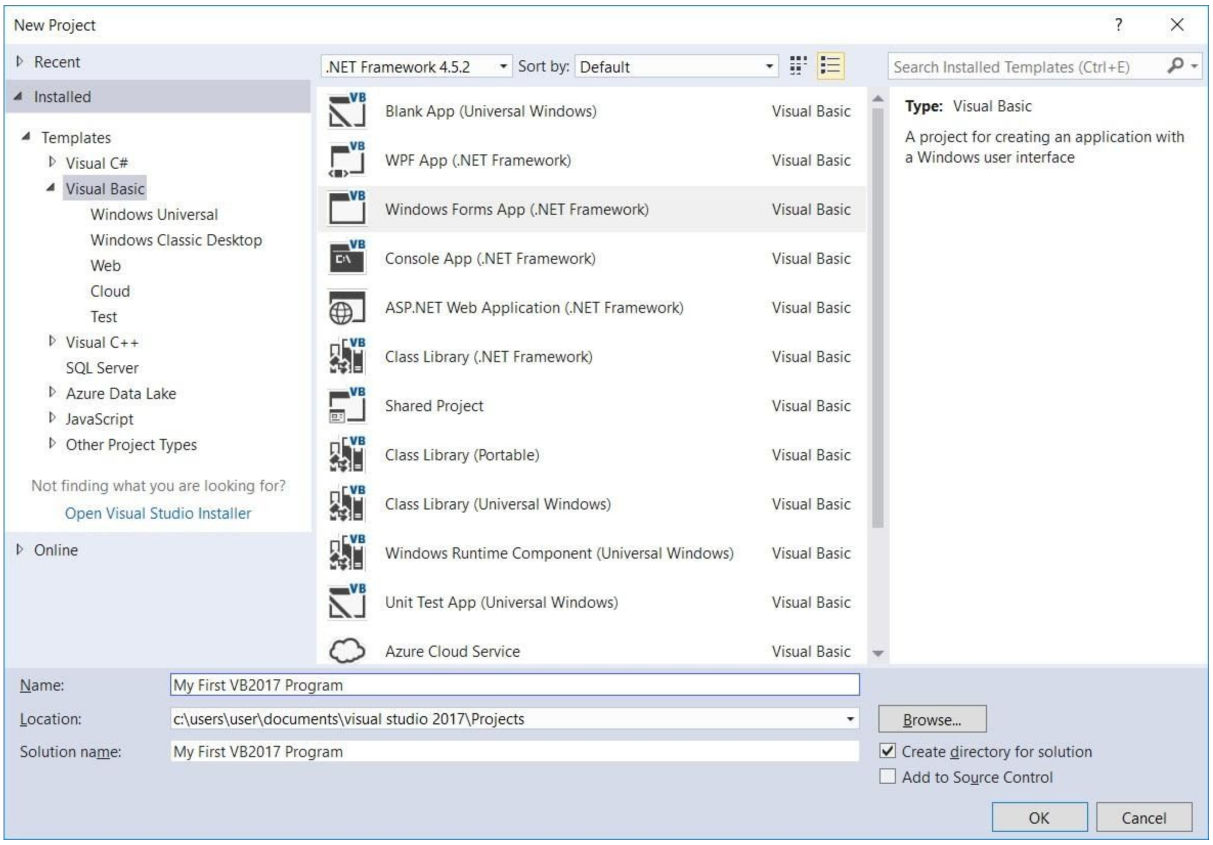
More news...

Figure 1.3 Visual Studio 2017 IDE

The Visual Studio 2017 start page comprises a few sections, the Get Started section, the Recent section, the Open section, the New project section and the Developers News section. In the start page, you can either start a new project, open a project or open a recent project. Besides that, you can also check for the latest news in Visual Studio 2017 for Windows Desktop. The Start Page also consists of a menu bar and a toolbar where you can perform various tasks by clicking the menu items.

1.3 Creating a New Project in Visual Studio 2017

To start a new Visual Studio 2017 project, click on New Project under the Start section to launch the Visual Studio 2017 New Project page as shown in Figure 1.4. You can also choose to open a recent project:

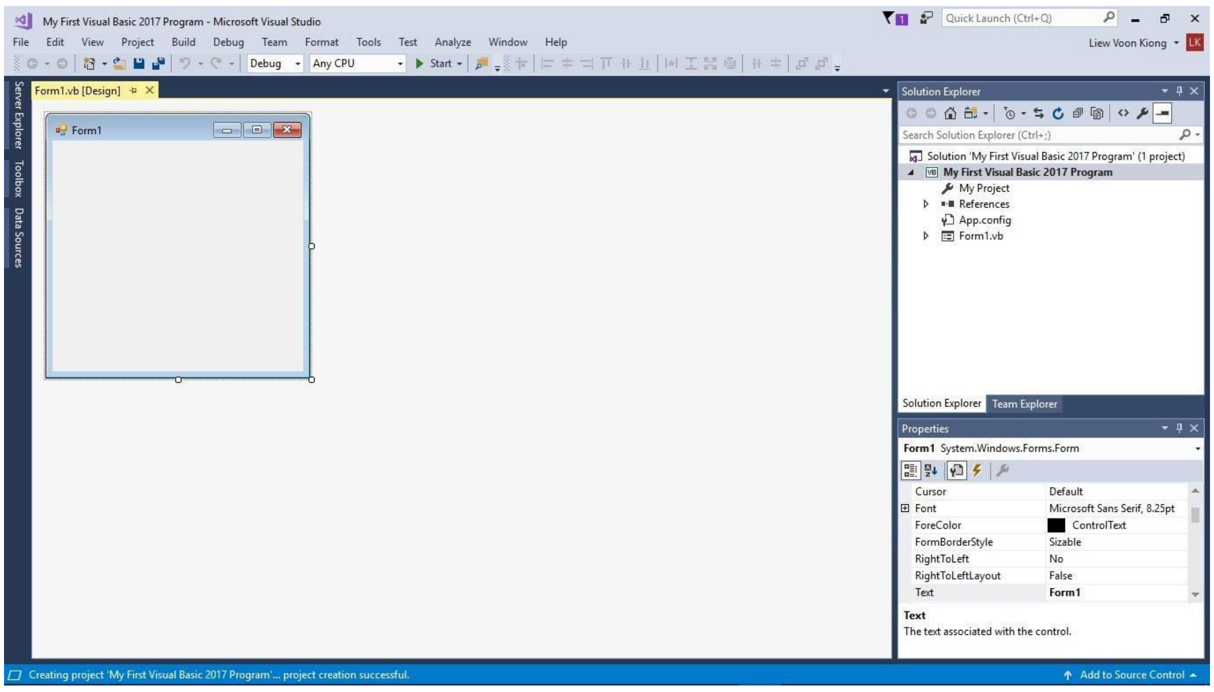

New Project
Recent
Installed
Templates
Visual C#
Visual Basic
Windows Universal
Windows Classic Desktop
Web
Cloud
Test
Visual C++
SQL Server
Azure Data Lake
JavaScript
Other Project Types
Not finding what you are looking for?
Open Visual Studio Installer
Online
.NET Framework 4.5.2
Sort by: Default
Search Installed Templates (Ctrl+E)
Blank App (Universal Windows) Visual Basic
WPF App (.NET Framework) Visual Basic
Windows Forms App (.NET Framework) Visual Basic
Console App (.NET Framework) Visual Basic
ASP.NET Web Application (.NET Framework) Visual Basic
Class Library (.NET Framework) Visual Basic
Shared Project Visual Basic
Class Library (Portable) Visual Basic
Class Library (Universal Windows) Visual Basic
Windows Runtime Component (Universal Windows) Visual Basic
Unit Test App (Universal Windows) Visual Basic
Azure Cloud Service Visual Basic
Type: Visual Basic
A project for creating an application with a Windows user interface
Browse...
OK
Cancel

Name: My First VB2017 Program
Location: c:\users\user\documents\visual studio 2017\Projects
Solution name: My First VB2017 Program
☑ Create directory for solution
☐ Add to Source Control

Figure 1.4 **Visual Studio 2017 New Project Page**

The New Project Page comprises a few templates, among them are Visual Basic, Visual C# and Visual C++. Since we are only learning Visual Basic 2017, we shall select Visual Basic. Visual Basic 2017 offers you several types of projects that you can create; they are *Blank Apps*, *Windows Forms APP(.NET Framework), WPF App(.NET Framework), Console App(.NET Framework) ,Class Library(.NET Framework), Shared Project* and more. Since we are only learning how to create windows desktop applications, we shall select Windows Forms App.

At the bottom of this dialog box, you can change the default project name WindowsApplication1 to some other name you like, for example, My First Visual Basic 2017 Program. After you have renamed the project, click OK to continue. The Visual Basic 2017 IDE Windows will appear, as shown in Figure 1.5. Visual Basic 2017 IDE comprises a few windows, the Form window, the Solution Explorer window and the Properties window. It also consists of a Toolbox which contains many useful controls that allows a programmer to develop his or her Visual Basic 2017 programs.


My First Visual Basic 2017 Program - Microsoft Visual Studio
File Edit View Project Build Debug Team Format Tools Test Analyze Window Help
Quick Launch (Ctrl+Q)
Liew Voon Kiong
Debug
Any CPU
Start
Server Explorer
Toolbox
Data Sources
Form1.vb [Design]
Form1
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'My First Visual Basic 2017 Program' (1 project)
My First Visual Basic 2017 Program
My Project
References
App.config
Form1.vb
Solution Explorer
Team Explorer
Properties
Add to Source Control

Form1 System.Windows.Forms.Form

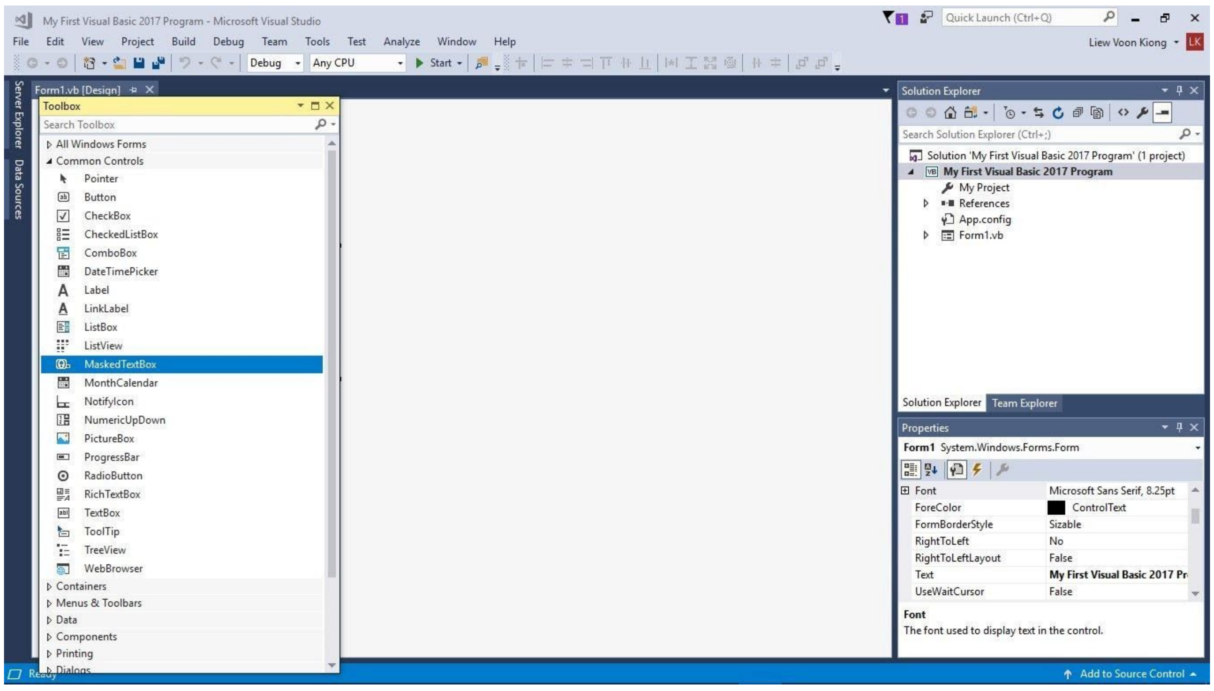
Cursor	Default
Font	Microsoft Sans Serif, 8.25pt
ForeColor	■ ControlText
FormBorderStyle	Sizable
RightToLeft	No
RightToLeftLayout	False
Text	Form1

Text
The text associated with the control.

Creating project 'My First Visual Basic 2017 Program'... project creation successful.

Figure 1.5 The Visual Basic 2017 Express IDE

The Toolbox is not shown until you click on the Toolbox tab. When you click on the Toolbox tab or use the shortcut keys Ctrl+Alt+x, the common controls Toolbox will appear, as shown in Figure 1.6. You can drag and move your toolbox around and dock it to the right, left ,top or bottom of the IDE.



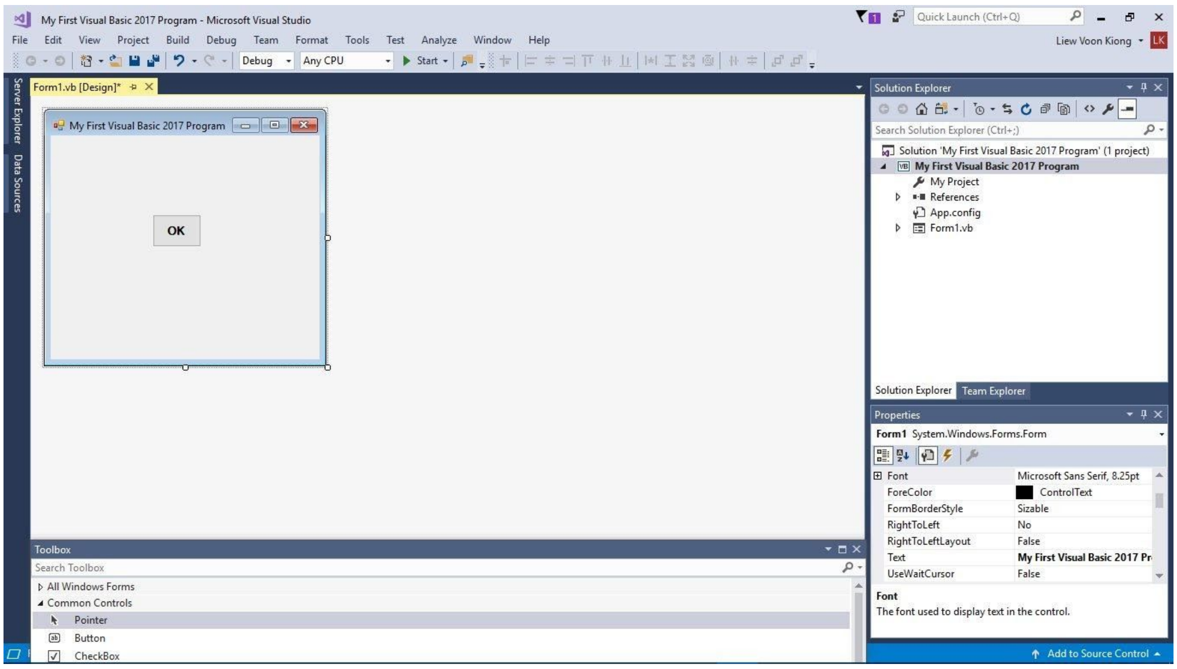
Form1 System.Windows.Forms.Form

Font	Microsoft Sans Serif, 8.25pt
ForeColor	■ ControlText
FormBorderStyle	Sizable
RightToLeft	No
RightToLeftLayout	False
Text	My First Visual Basic 2017 Pr
UseWaitCursor	False

Font
The font used to display text in the control.

Figure 1.6 Visual Basic 2017 Express Toolbox

Next, we shall proceed to show you how to create your first program. First, change the text of the form to 'My First Visual Basic 2017 Program' in the properties window; it will appear as the title of the program. Next, insert a button and change its text to OK. The design interface is shown in Figure 1.7


My First Visual Basic 2017 Program - Microsoft Visual Studio
File Edit View Project Build Debug Team Format Tools Test Analyze Window Help
Quick Launch (Ctrl+Q)
Liew Voon Kiong
Debug
Any CPU
Start
Form1.vb [Design]*
My First Visual Basic 2017 Program
OK
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'My First Visual Basic 2017 Program' (1 project)
My First Visual Basic 2017 Program
My Project
References
App.config
Form1.vb
Team Explorer
Toolbox
Search Toolbox
All Windows Forms
Common Controls
Pointer
Button
CheckBox
Add to Source Control

Properties

Form1 System.Windows.Forms.Form

Font	Microsoft Sans Serif, 8.25pt
ForeColor	ControlText
FormBorderStyle	Sizable
RightToLeft	No
RightToLeftLayout	False
Text	My First Visual Basic 2017 Pr
UseWaitCursor	False

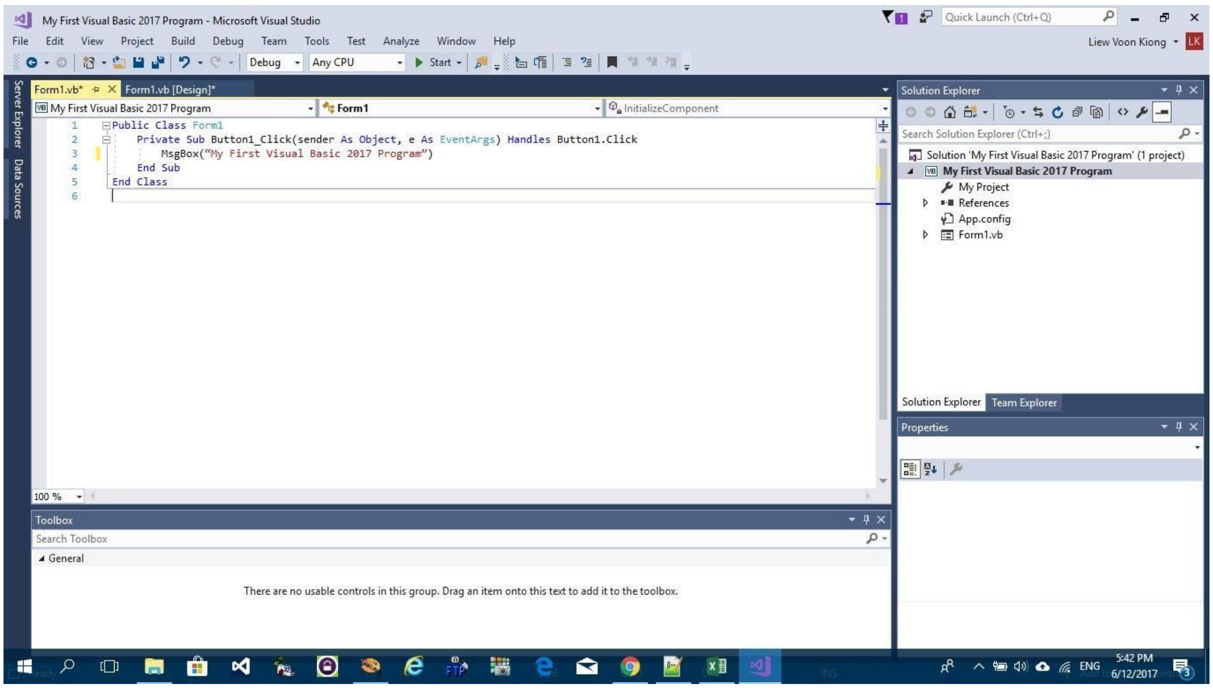
Font
The font used to display text in the control.

Figure 1.7 The Design Interface

Now click on the OK button to bring up the code window and enter the following statement between **`Private Sub`** and **`End Sub`** procedure, as shown in Figure 1.6:

```
MsgBox("My First Visual Basic 2017 Program")
```

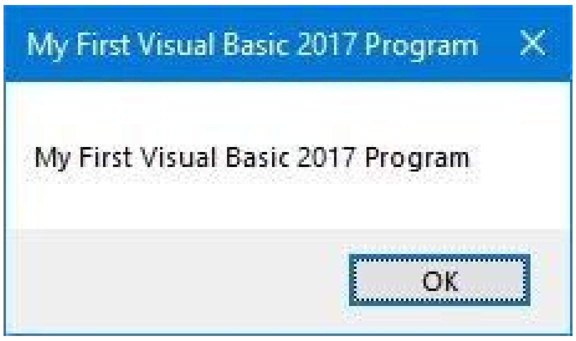
Now click on the Start button on the toolbar or press F5 to run the program then click on the OK button, a dialog box that displays the “My First Visual Basic 2017 Program" message will appear, as shown in Figure 1.8. The function **`MsgBox`** is a built-in function of Visual Basic 2017 which can display the text enclosed within the brackets



```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        MsgBox("My First Visual Basic 2017 Program")
    End Sub
End Class
```

My First Visual Basic 2017 Program - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug
Any CPU
Start
Form1.vb*
Form1.vb [Design]*
Solution Explorer
Solution 'My First Visual Basic 2017 Program' (1 project)
My First Visual Basic 2017 Program
My Project
References
App.config
Form1.vb
Properties
Toolbox
General
There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.

Figure 1.8 Visual Basic 2017 Code Window



My First Visual Basic 2017 Program

My First Visual Basic 2017 Program

OK

Figure 1.9 The Runtime Interface

Summary

- In section 1.1, you have learned about the history of Visual Basic 2017
- In section 1.2, you have learned how to install and launch Visual Basic Studio 2017
- In section 1.3, you have learned how to launch the new project dialog and the Visual Basic 2017 IDE. You have also learned how to write your first program.

Chapter 2 Designing the Interface

- ❖ Customizing
- ❖ Adding controls
- ❖ Setting Control Properties

Since Visual Basic 2017 is a GUI-based programming language, the first step in developing an application is to build a graphical user interface. To build a graphical user interface, you need to customize the default form by changing its properties at design phase and at runtime. After customizing the default form, you may proceed to add controls from the toolbox to the form and then customize their properties.

2.1 Customizing the Form

When you start a new Visual Basic 2017 project, the VS2017 IDE will display the default form along with the Solution Explorer window and the Properties window for the form as shown in Figure 2.1. The name of the default form is Form1. The properties window displays all the properties related to Form1 and their corresponding attributes or values. You can change the name of the form, the title of the form using the text property, the background color, the foreground color, the size and more. Try changing the properties in Table 1.

Table 1

Property	***Value***
Name	MyForm
Text	My First Visual Basic 2017 Program
BackColor	Aqua
ForeColor	DarkBlue
MaximizeBox	False

In fact, you do not have to type in the color manually, you may select a color from the color drop-down list that comprises three tabs, Custom, Web, and System, as shown in Figure 2.1. Clicking on the drop-down arrow will bring out a color palette or a list of color rectangles where you can select a color.

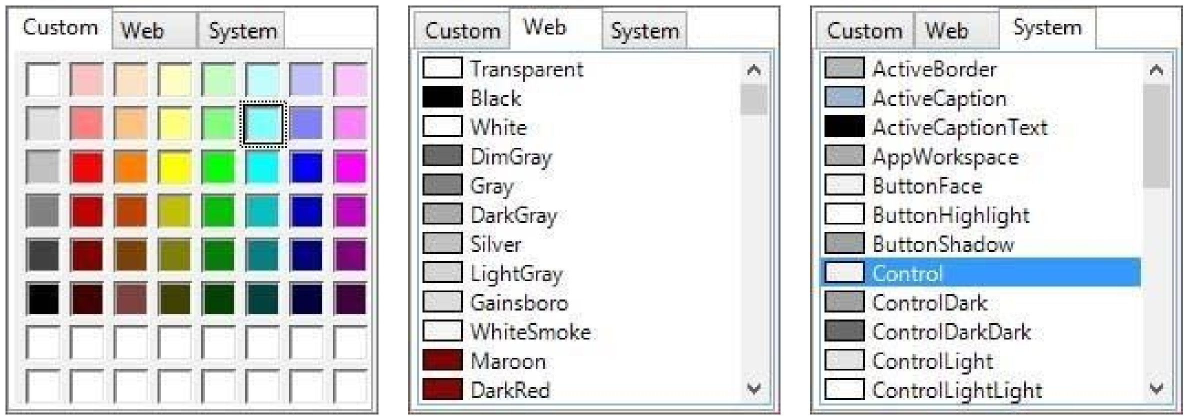
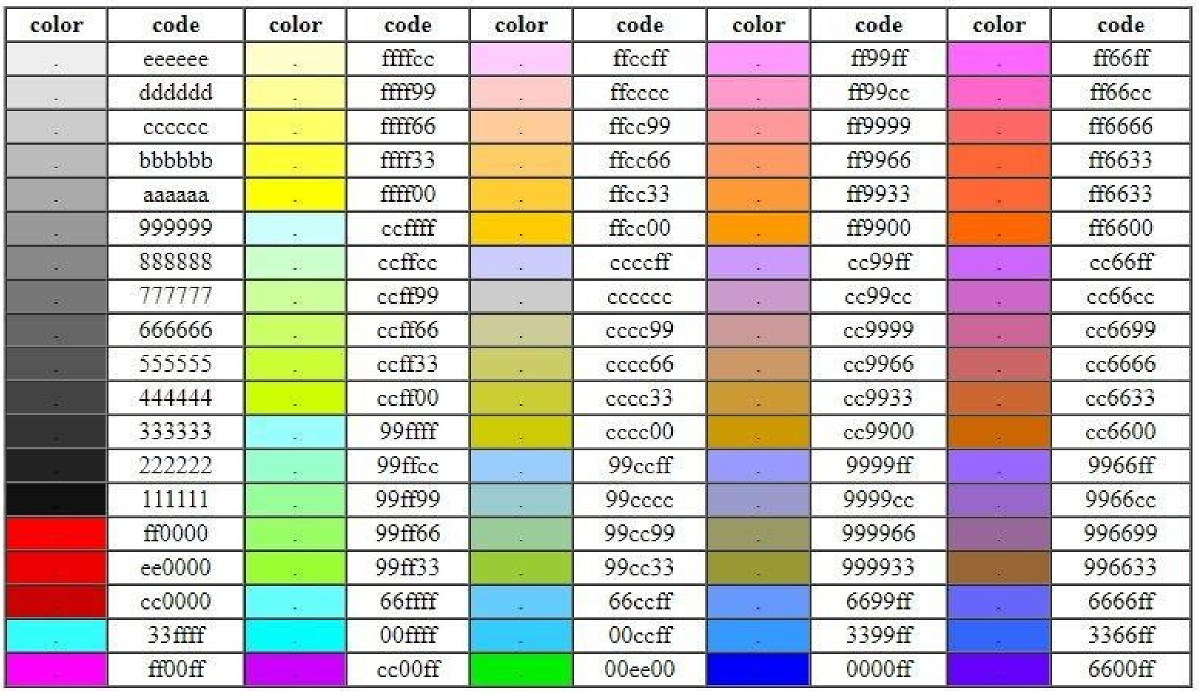


Figure 2.1

Another method of setting the colors is to manually type in the RGB color code or the hex color code. The values of R, G and B range from 0 to 255, by varying the values of the RGB we can obtain different colors. For example, an RGB value of 128, 255, 255 yields the cyan color.

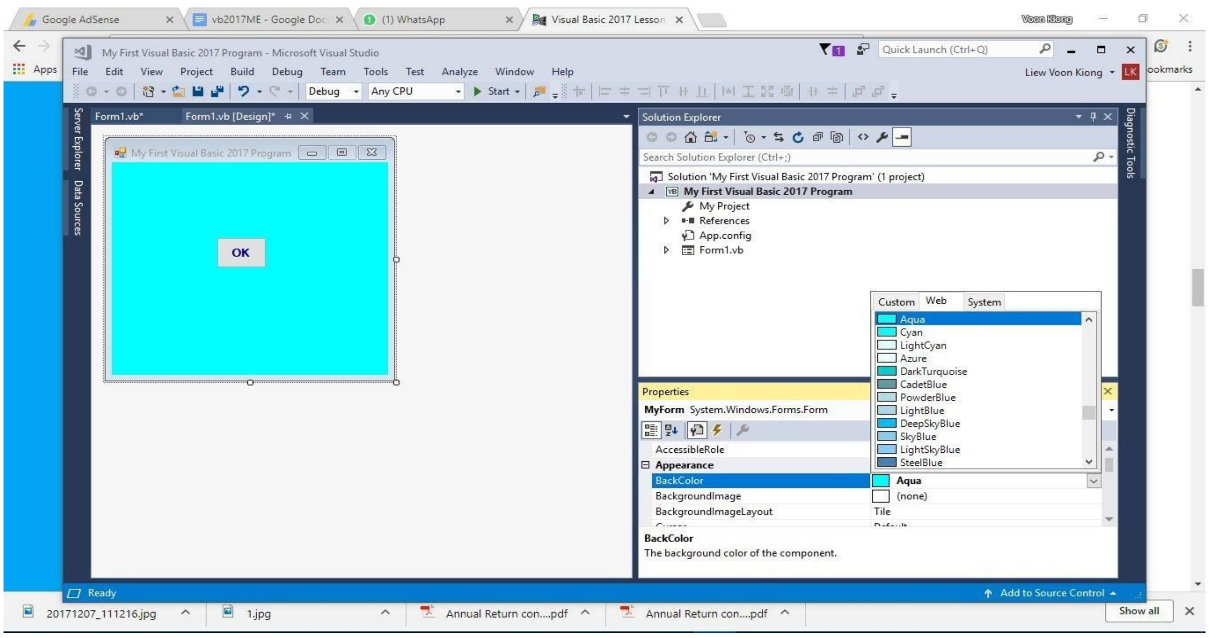
On the other hand, the hex color code system use a six-digit, three-byte hexadecimal number to represent colors. The bytes represent the red, green and blue components of the color. One byte represents a number in the range 00 to ff (in hexadecimal notation), or 0 to 255 in decimal notation. For example, **`#0000ff`** represents the cyan color. However, when you type in the hex color code in the properties window of VS2017, it automatically converts the color to RGB color or the color name. Figure 2.2 shows a list of Hex color codes and the corresponding colors.



color	code	color	code	color	code	color	code	color	code
.	eeeeee	.	ffffcc	.	ffccff	.	ff99ff	.	ff66ff
.	dddddd	.	ffff99	.	ffcccc	.	ff99cc	.	ff66cc
.	cccccc	.	ffff66	.	ffcc99	.	ff9999	.	ff6666
.	bbbbbb	.	ffff33	.	ffcc66	.	ff9966	.	ff6633
.	aaaaaa	.	ffff00	.	ffcc33	.	ff9933	.	ff6633
.	999999	.	ccffff	.	ffcc00	.	ff9900	.	ff6600
.	888888	.	ccffcc	.	ccccff	.	cc99ff	.	cc66ff
.	777777	.	ccff99	.	cccccc	.	cc99cc	.	cc66cc
.	666666	.	ccff66	.	cccc99	.	cc9999	.	cc6699
.	555555	.	ccff33	.	cccc66	.	cc9966	.	cc6666
.	444444	.	ccff00	.	cccc33	.	cc9933	.	cc6633
.	333333	.	99ffff	.	cccc00	.	cc9900	.	cc6600
.	222222	.	99ffcc	.	99ccff	.	9999ff	.	9966ff
.	111111	.	99ff99	.	99cccc	.	9999cc	.	9966cc
.	ff0000	.	99ff66	.	99cc99	.	999966	.	996699
.	ee0000	.	99ff33	.	99cc33	.	999933	.	996633
.	cc0000	.	66ffff	.	66ccff	.	6699ff	.	6666ff
.	33ffff	.	00ffff	.	00ccff	.	3399ff	.	3366ff
.	ff00ff	.	cc00ff	.	00ee00	.	0000ff	.	6600ff

Figure 2.2 Hex Color Codes

The design interface is shown in Figure 2.2 and the runtime interface is shown in Figure 2.4. In the runtime interface, notice that the title has been changed from Form1 to My First Visual Basic 2017 Program, background color changed to aqua , the text OK color has been changed to dark blue and the window cannot be maximized.



My First Visual Basic 2017 Program - Microsoft Visual Studio

Solution 'My First Visual Basic 2017 Program' (1 project)

My First Visual Basic 2017 Program

MyForm System.Windows.Forms.Form

BackColor Aqua

BackgroundImage (none)

BackgroundImageLayout Tile

BackColor
The background color of the component.

Figure 2.3

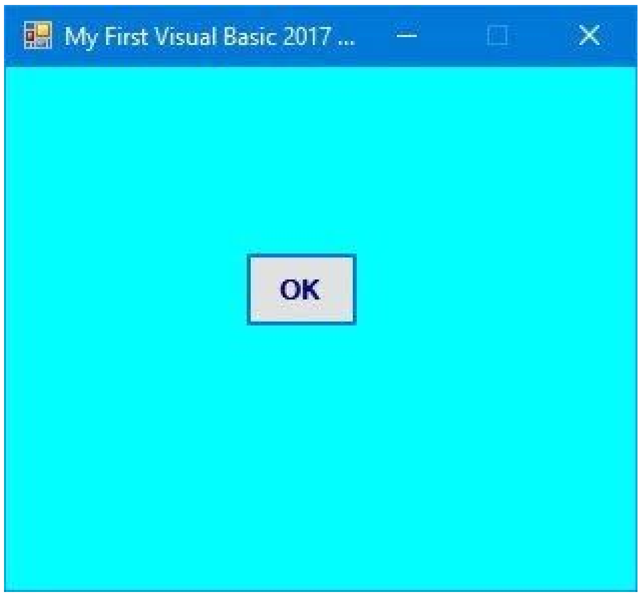


Figure 2.4

You can change the properties of the form at run-time by writing the relevant code. The default form is an object and an instant of the form can be denoted by the name **`Me.`** The property of the object is defined by specifying the object's name followed by a period:

```
ObjectName.property
```

For example, we can set the background of the form to blue using the following code:

```
Me.BackColor=Color.Blue
```

In addition, you can also use the **`FromArgb`** method to specify the color using the RGB codes, as follows:

```
Me.BackColor = Color.FromArgb(0, 255, 0)
```

To achieve the same interface as shown in Figure 2.4, use following code :

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
Me.Text = "My First Visual Basic 2017 Program"
Me.BackColor = Color.Cyan
Me.MaximizeBox = False
Me.MinimizeBox = True
End Sub
```

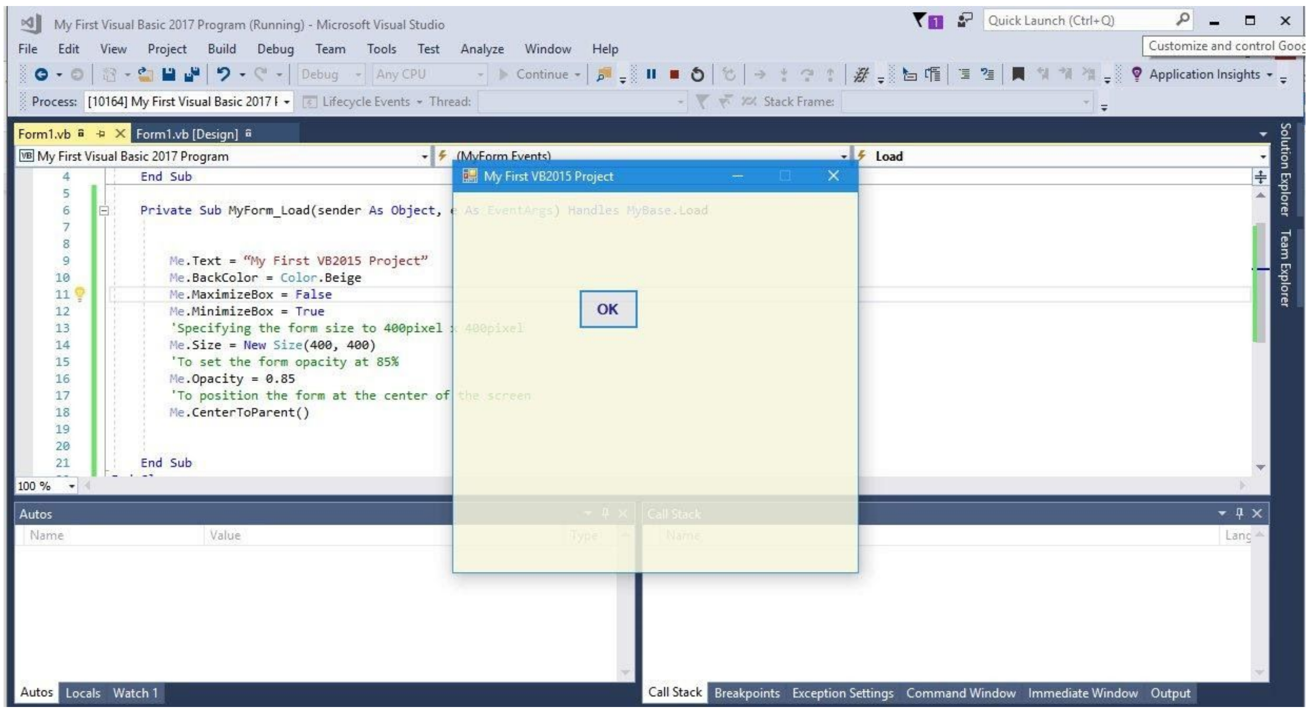
Instead of using the color name cyan, you can use RGB code, as follows:

```
Me.BackColor = Color.FromArgb(0,255,255)
```

Press F5 to run the program and you will get the exact interface as that shown in Figure 2.4. In addition, you can also specify the size, the opacity and the position of the default form using the code, as follows:

```
Private Sub Form1_Load(sender As Object, e As EventArgs Handles
MyBase.Load
Me.Text = "My First VB2017 Project"
Me.BackColor =Color.Beige
Me.MaximizeBox = False
Me.MinimizeBox = True
Me.Size = New Size(400, 400)
Me.Opacity = 0.85
Me.CenterToParent()
End Sub
```

The runtime interface is as shown in Figure 2.5



```
    End Sub

    Private Sub MyForm_Load(sender As Object, e As EventArgs) Handles MyBase.Load


        Me.Text = "My First VB2015 Project"
        Me.BackColor = Color.Beige
        Me.MaximizeBox = False
        Me.MinimizeBox = True
        'Specifying the form size to 400pixel x 400pixel
        Me.Size = New Size(400, 400)
        'To set the form opacity at 85%
        Me.Opacity = 0.85
        'To position the form at the center of the screen
        Me.CenterToParent()


    End Sub
```

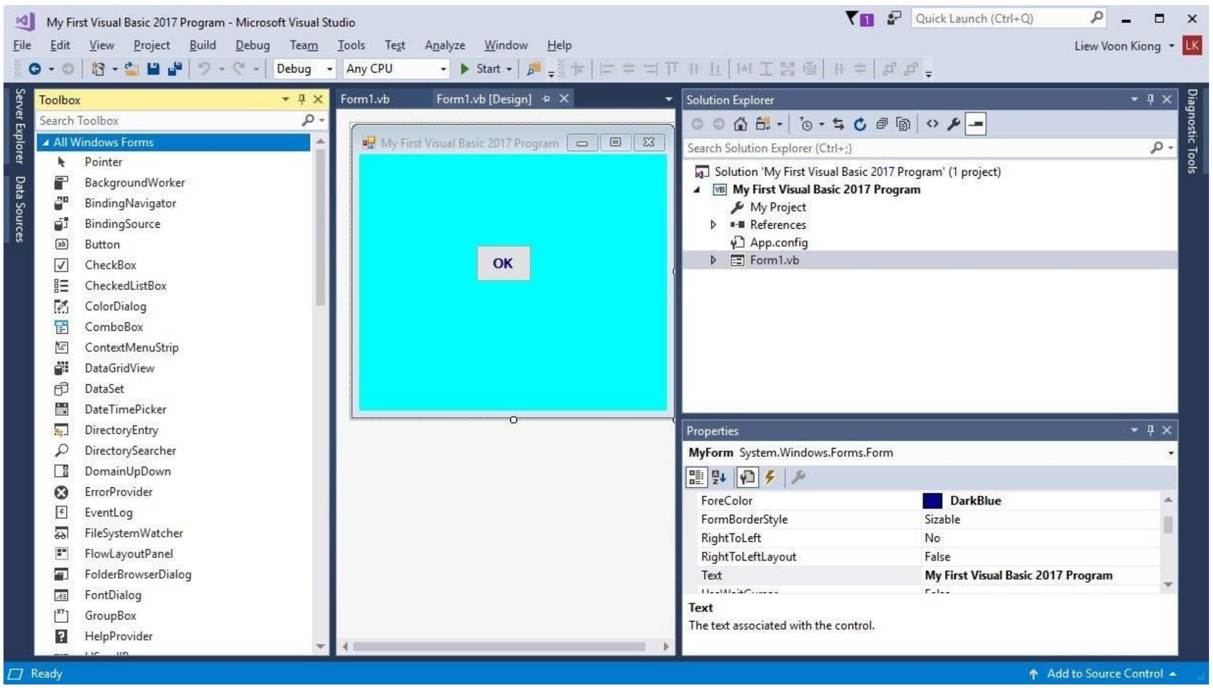
My First Visual Basic 2017 Program (Running) - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Quick Launch (Ctrl+Q)
Debug
Any CPU
Continue
Application Insights
Process: [10164] My First Visual Basic 2017 P
Lifecycle Events
Thread:
Stack Frame:
Form1.vb
Form1.vb [Design]
My First Visual Basic 2017 Program
(MyForm Events)
Load
My First VB2015 Project
OK
100 %
Autos
Name
Value
Type
Call Stack
Name
Lang
Autos Locals Watch 1
Call Stack Breakpoints Exception Settings Command Window Immediate Window Output
Solution Explorer Team Explorer

Figure 2.5

2.2 Adding Controls to the Form

In section 2.1, we have learned how to build an initial interface in Visual Basic 2017 by customizing the default form. Next, we shall continue to build the interface by adding some controls to the form. The controls are objects that consist of three elements, i.e. properties, methods, and events. Controls can be added to the form from the Toolbox. Among the controls, the most common ones are the button, label, textbox, list box, combo box, picture box, checkbox, radio button and more. The controls can be made visible or invisible at runtime. However, some controls will only run in the background and cannot be seen at runtime, one such control is the timer.

The Toolbox is usually hidden when you start Visual Basic 2017 IDE, you need to click View on the menu bar and then select Toolbox to reveal the Toolbox, as shown in Figure 2.6. You can also use shortcut keys Ctrl+w+x to bring out the toolbox.



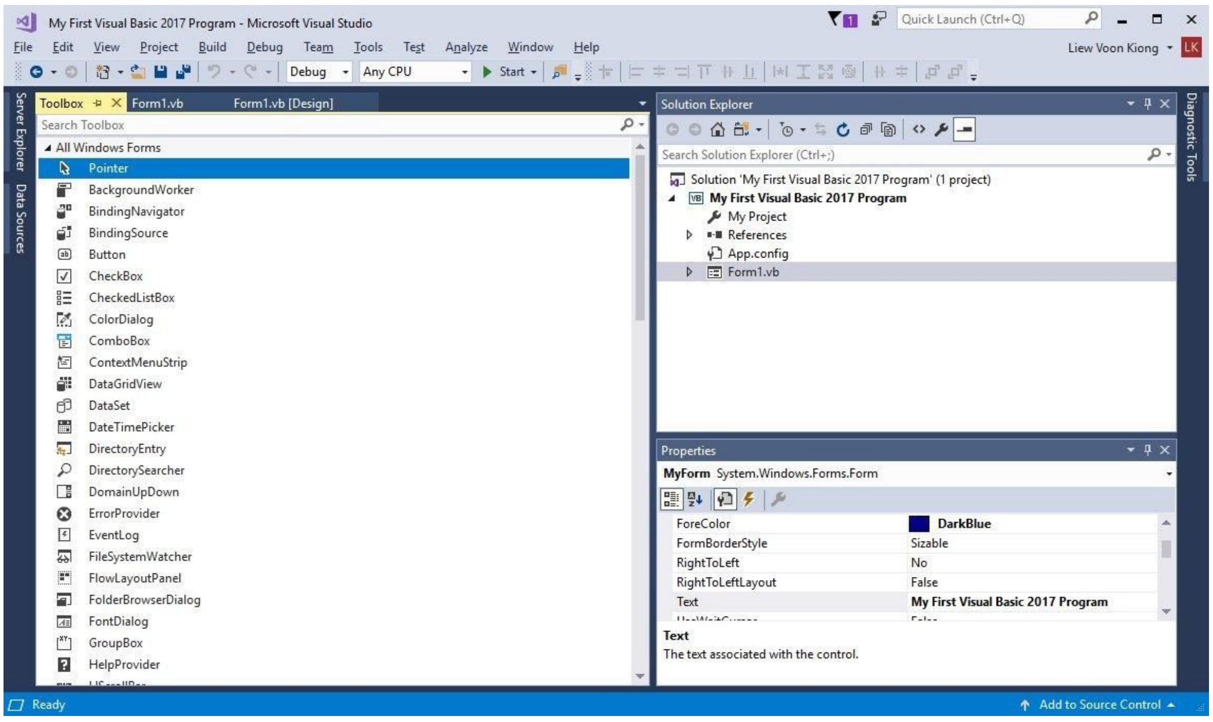
MyForm System.Windows.Forms.Form

ForeColor	DarkBlue
FormBorderStyle	Sizable
RightToLeft	No
RightToLeftLayout	False
Text	My First Visual Basic 2017 Program

Text
The text associated with the control.

Figure 2.6: Toolbox

You can drag the Toolbox to any position you like. You can also dock the Toolbox by right-clicking on the Toolbox and choose dock from the pop-up menu. The docked Toolbox appears side by side with the Solution Explorer, and as one of the tabbed windows together with the Form Design window and the code window, as shown in Figure 2.7.



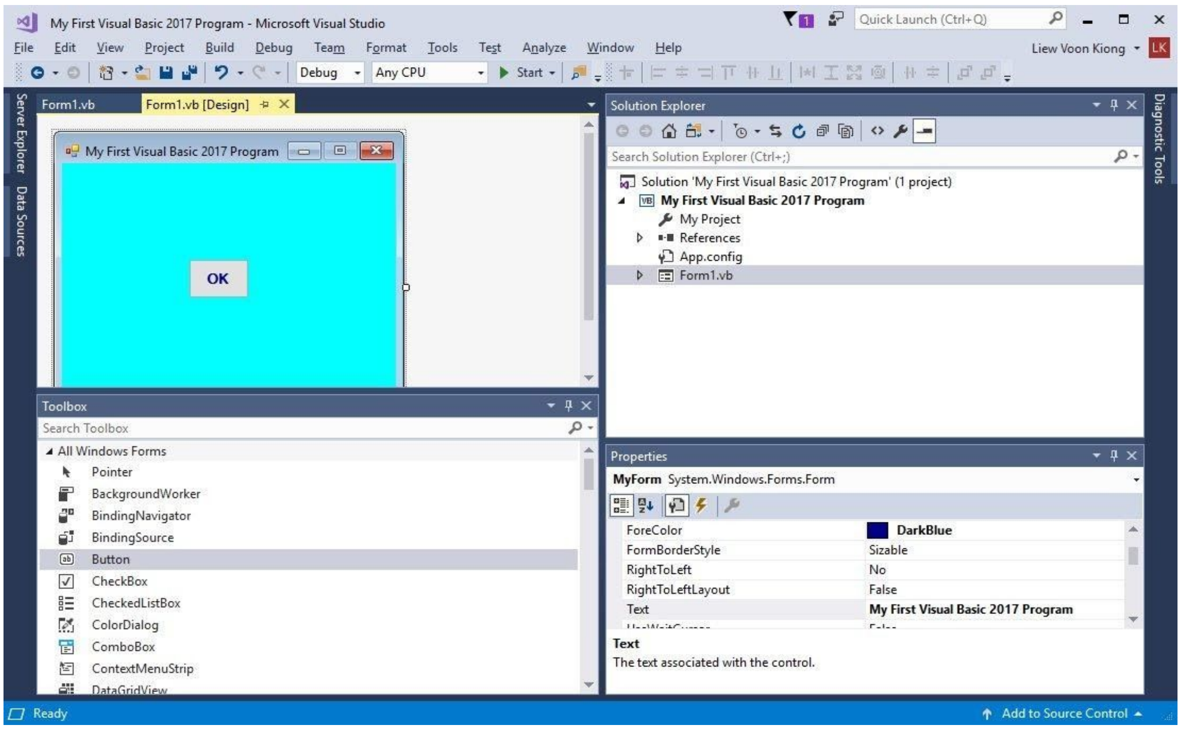
MyForm System.Windows.Forms.Form

ForeColor	DarkBlue
FormBorderStyle	Sizable
RightToLeft	No
RightToLeftLayout	False
Text	My First Visual Basic 2017 Program

Text
The text associated with the control.

Figure 2.7

You can also dock the Toolbox at the bottom, below the default form, as shown in Figure 2.8. Further, you may also pin the Toolbox to the side bar or the bottom bar. We suggest that you place the toolbox alongside or at the bottom of the default form so that it is easy for you to add controls from the toolbox into the form.



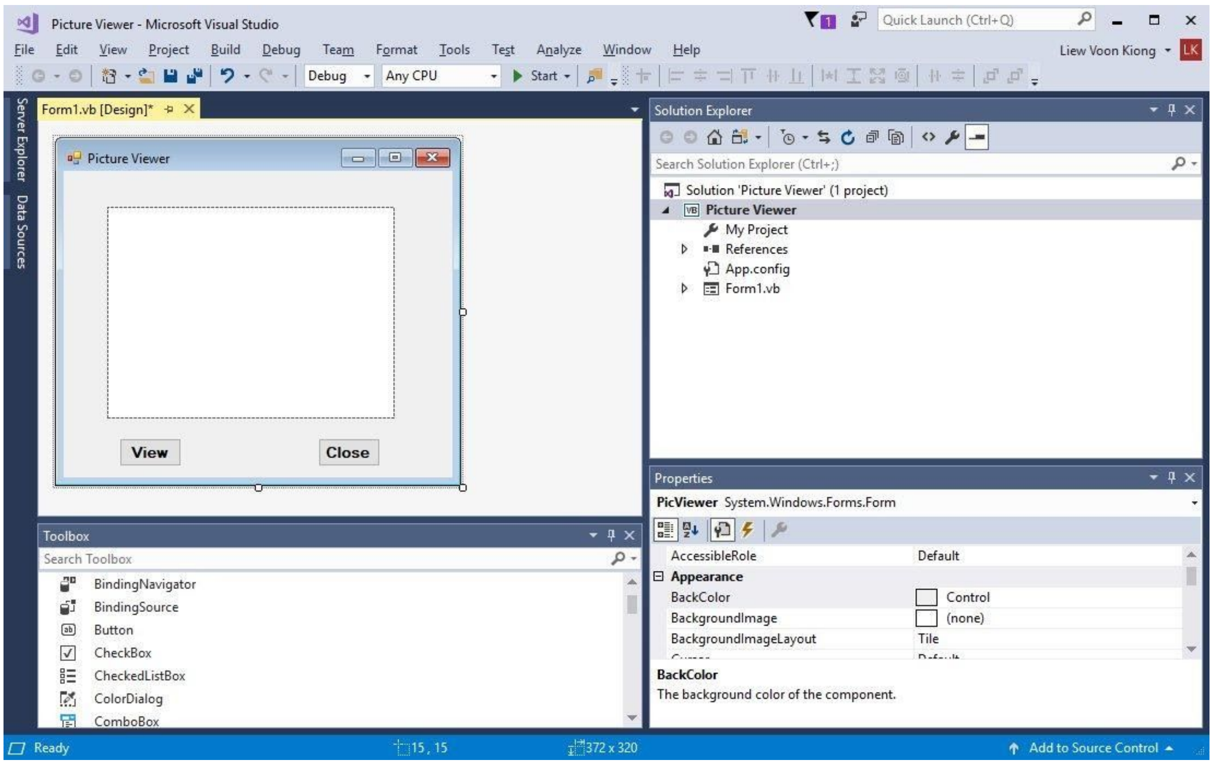
ForeColor	DarkBlue
FormBorderStyle	Sizable
RightToLeft	No
RightToLeftLayout	False
Text	My First Visual Basic 2017 Program

Text
The text associated with the control.

Figure 2.8

To add a control to the form, click the control and drag it onto the form. You can drag the control around in the form and you can also resize it.

To demonstrate how to add the controls and then change their properties, we shall design a picture viewer. First, change the title of the default form to Picture Viewer in its properties window. Next, insert a picture box on the form and change its background color to white. To do this, right click the picture box and select properties in the popup menu, then look for the **`BackColor`** Property as shown in the properties window in Figure 2.9. Finally, add two buttons to the form and change the text to View and Close in their respective properties' windows. The picture viewer is not yet functional until we write code to it. We shall deal with the programming part in the coming chapters.



PicViewer System.Windows.Forms.Form

AccessibleRole	Default
Appearance	
BackColor	Control
BackgroundImage	(none)
BackgroundImageLayout	Tile

BackColor
The background color of the component.

Figure 2.9

Summary

- In section 2.1, you have learned how to customize the form by changing the values of its properties.
- In section 2.2, you have learned how to add controls to the form and change their properties at design phase and at runtime.

Chapter 3 Writing the Code

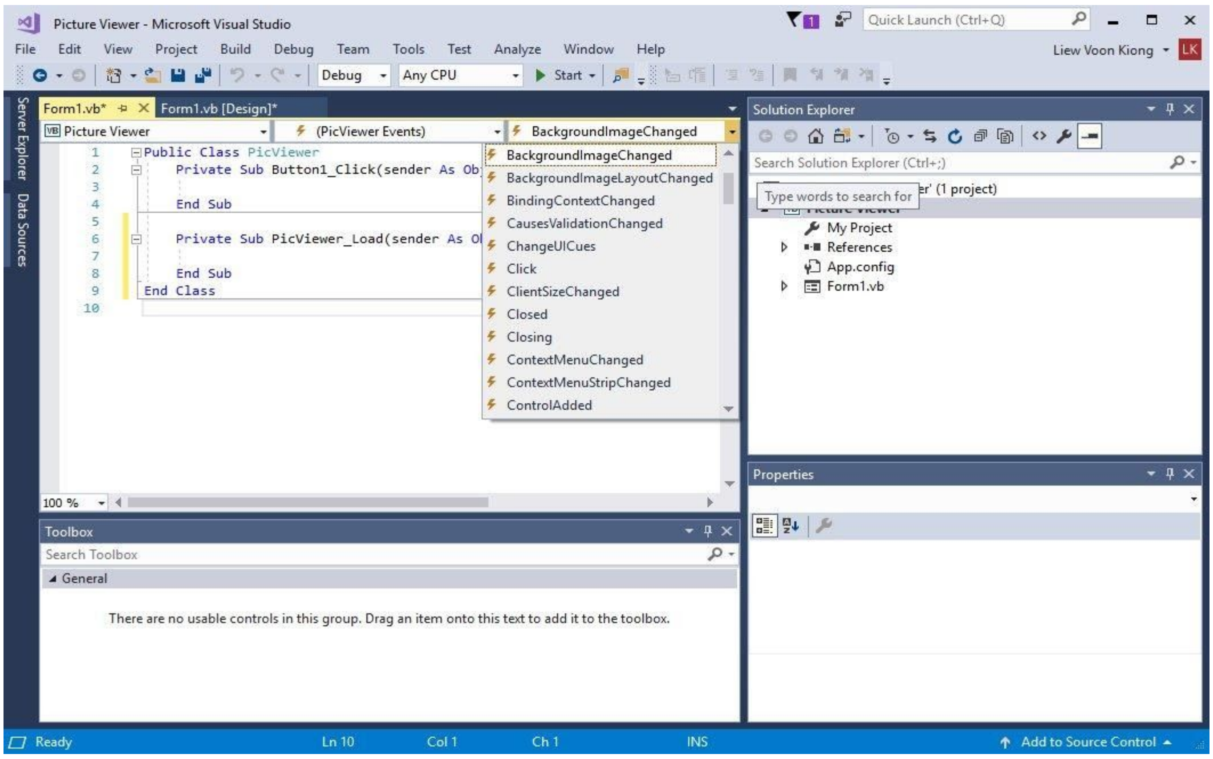
❖ Learn the basics of writing code

In the previous chapter, we have learned how to design the user interface by adding controls to the form and by changing their properties. However, the user interface alone will not work without adding code to them. In this chapter, we shall learn how to write code for all the controls so that they can interact with the events triggered by the users. Before learning how to write code, let us delve into the concept of event-driven programming

3.1 The Concept of Event-Driven Programming

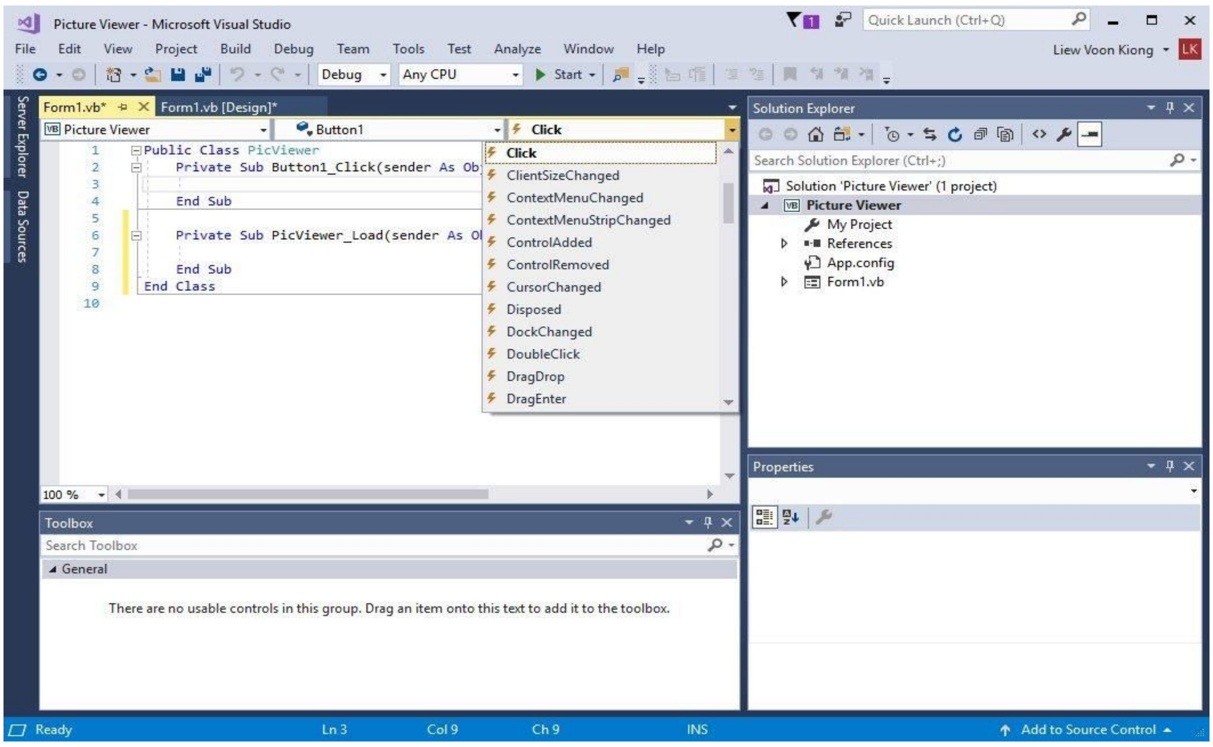
Visual Basic 2017 is an event-driven programming language. Event-driven means that the code is executed in response to events triggered by the user actions like clicking the mouse, pressing a key on the keyboard, selecting an item from a drop-down list, typing some words into textbox and more. It may also be an event that runs in response to some other programs. Some of the common events in Visual Basic 2017 are load, click, double-click, drag and drop, pressing the keys and more.

Every form and every control you place on the form has a set of events related to them. To view the events, double-click the control (object) on the form to enter the code window. The default event will appear at the top part on the right side of the code window. You need to click on the default event to view other events associated with the control. The code appears on the left side is the event procedure associated with the load event. Figure 3.1 illustrates the event procedure load associated with the Form (its name has been changed to PicViewer therefore you can see the words PicViewer events) and Figure 3.2 shows the events associated with button.



```
Public Class PicViewer
    Private Sub Button1_Click(sender As Ob

    End Sub

    Private Sub PicViewer_Load(sender As Ol

    End Sub
End Class
```

Figure 3.1: Events associated with Form



```
Public Class PicViewer
    Private Sub Button1_Click(sender As Ob

    End Sub

    Private Sub PicViewer_Load(sender As Ol

    End Sub
End Class
```

Figure 3.2: Events associated with the button

3.2 Writing the Code

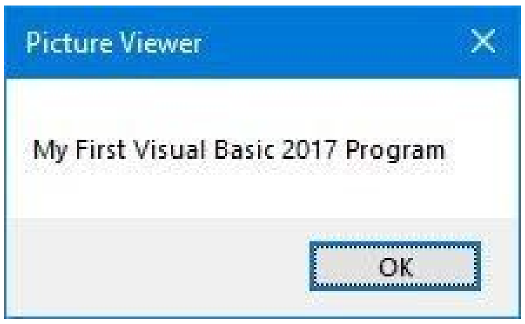
To start writing the code, click on any part of the form to go into the code window as shown in Figure 3.1. The event procedure is to load Form1 and it starts with the keywords **`Private Sub`** and ends with **`End Sub`**. This procedure includes the Form1 class and the event Load, and they are bind together with an underscore, i.e. **`Form_Load`**. It does nothing other than loading an empty form. To make the load event does something, insert the statement.

```
MsgBox ( "Welcome to Visual Basic 2017")
```

The Code

```
Public Class Form1
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
MsgBox ( "My First Visual Basic 2017 Program", ,"My Message")
End Sub
End Class
```

MsgBox is a built-in function in Visual Basic 2017 that displays a message in a message box. The MsgBox function comprises a few arguments, the first is the message and the third one is the title of the message box. When you run the program, a message box displaying the text “My First Visual Basic 2017 Program” will appear, as shown in Figure 3.3.



My First Visual Basic 2017 Program

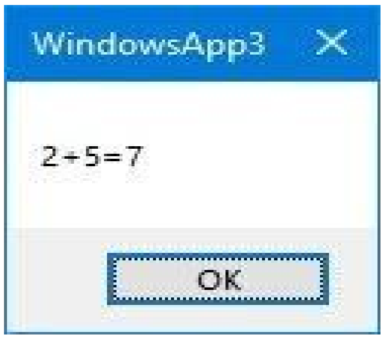
Figure 3.3

You will notice that above Private Sub structure there is a preceding keyword **`Public Class Form1`**. This is the model of an object-oriented programming language. When we start a windows application in Visual Basic 2017, we will see a default form with the name Form1 appears in the IDE, it is actually the Form1 Class that inherits from the Form class **`System.Windows.Forms.Form`**. A class has events as it creates an instant of a class or an object.

You can write code to perform arithmetic calculations. For example, you can use the arithmetic operator plus to perform addition of two numbers, and display the result in a message box, as shown below:

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
MsgBox("2" & "+" & "5" & "=" & 2 + 5)
End Sub
```

*The symbol & (ampersand) is to perform string concatenation. The output is as shown in Figure 3.4



```
2+5=7
```

.

Figure 3.4

Summary

- In section 3.1, you have learned the concepts of event driven programming
- In section 3.2, you have learned how to write code for the controls

Chapter 4 Working with Controls

- ❖ Learn to work with a text box
- ❖ Learn to work with a label control
- ❖ Learn to work with a list box
- ❖ Learn to work with a combo box

In the preceding chapter, we have learned how to write simple Visual Basic 2017 code. In this chapter, we shall learn how to write codes for some common controls. Some of the commonly used controls are label, textbox, button, list box and combo box. However, in this chapter, we shall only deal with textbox , label, list box and combo box. We shall deal with the other controls later chapters.

4.1 TextBox

A Textbox is the standard control for accepting inputs from the user as well as to display the output. It can handle string and numeric data but not images or pictures. String in a textbox can be converted to a numeric data by using the function **`Val(text)`**. The following example illustrates a simple program that processes the input from the user.

Example 4.1 Adding two numbers in two text boxes

In this program, you add two text boxes and a button on the form. The two text boxes are for accepting inputs from the user. Besides that, we can also program a button to calculate the sum of the two numbers using the plus operator. The value in a textbox is stored using the syntax **`TextBox1.Text`** , where Text is one of the properties of textbox.

The following code will add the value in TextBox1 and the value in TextBox2 and displays the sum in a message box. The runtime interface is illustrated in Figure 4.1.

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

MsgBox("The sum is" & Val(TextBox1.Text) + Val(TextBox2.Text))

End Sub
```

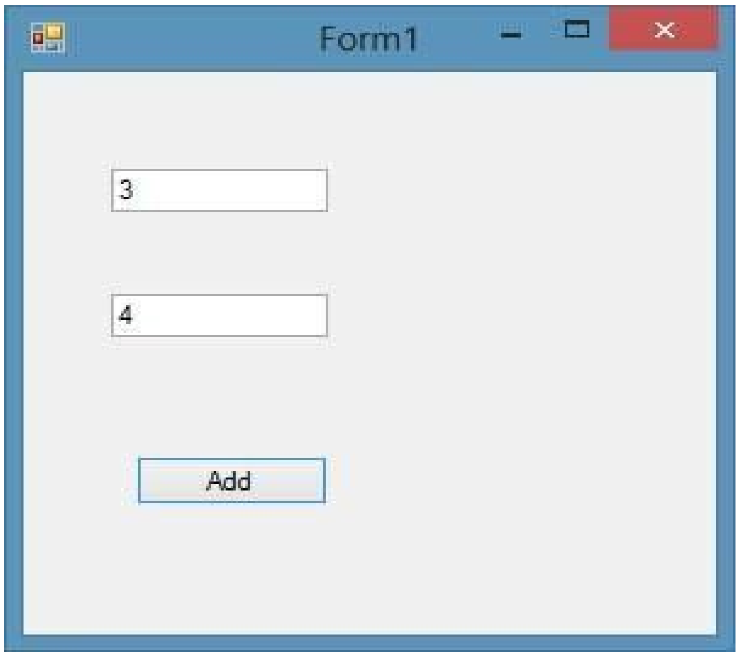
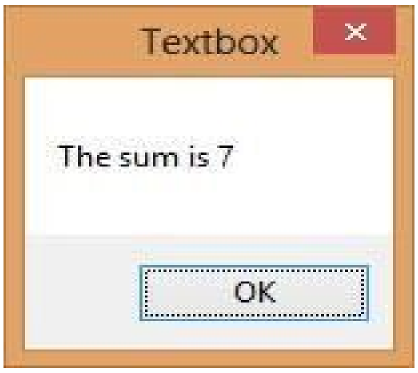


Figure 4.1

After clicking the Add button, you will obtain the answer in a message box, as shown in Figure 4.2



The sum is 7

:

Figure 4.2

4.2 Label

The Label control can be used for multiple purposes like providing instructions and guides to the users, displaying outputs and more. It is different from the textbox because it is read only , which means the user cannot change or edit its content at runtime. Using the syntax

`Label.Text`, it can display string as well as numeric data . You can change its text property in the properties window or at runtime by writing an appropriate code.

Example 4.2 Displaying output on a Label

Based on Example 4.1, we add two labels, one is for displaying the text Sum= and the other label is to display the answer of the Sum. For the first label, change the text property of the label by typing Sum= over the default text Label1. Further, change its font to bold and the font size to 10. For the second label, delete the default text Label2 and change its font to bold and the font size to 10. Besides that, change its background color to white.

In this program, instead of showing the sum in a message box, we display the sum on a label.

The Code

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
Label2.Text = Val(TextBox1.Text) + Val(TextBox2.Text)
End Sub
```

*The function Val is to convert text to numeric value. Without using Val, you will see that two numbers are joined instead of adding them.

The output is as shown in Figure 4.3

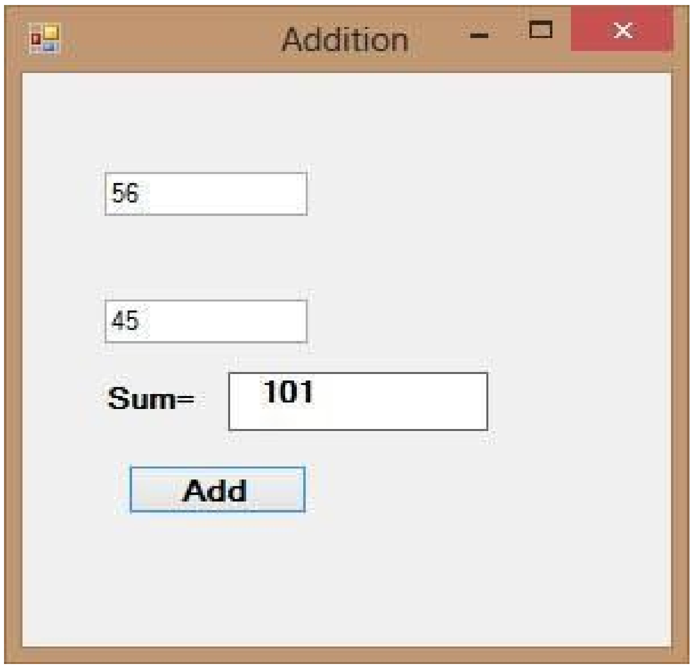


Figure 4.3

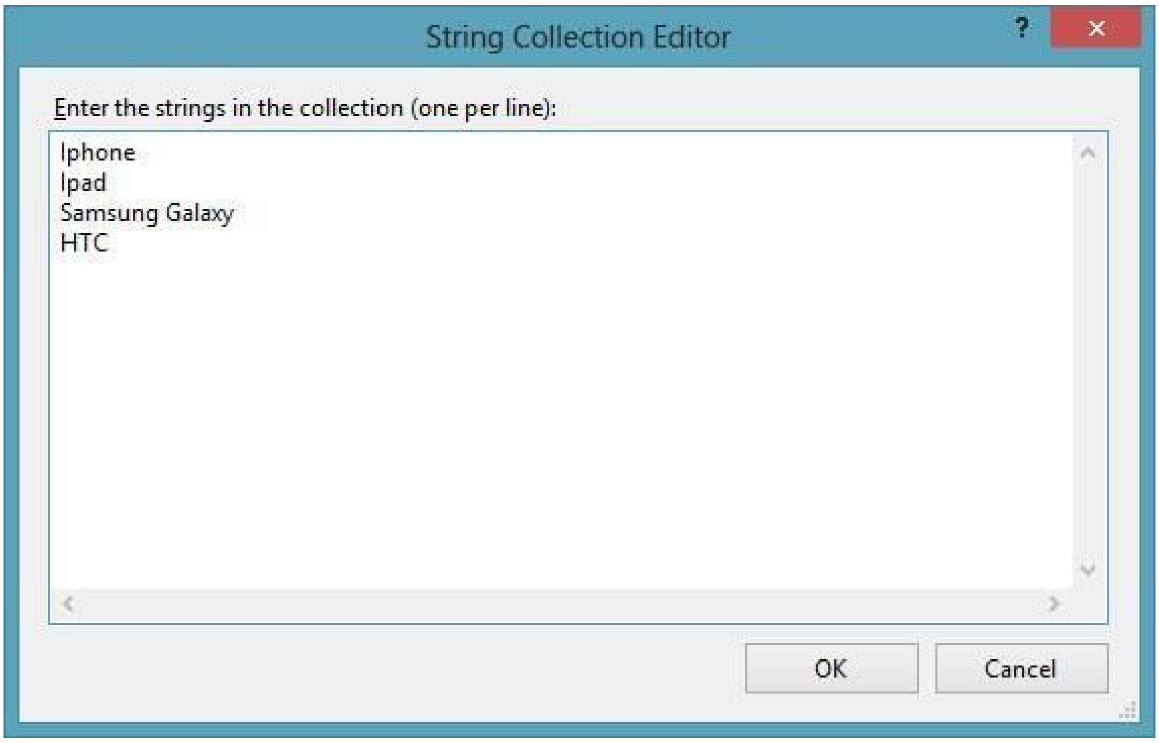
4.3 List Box

The function of the list box is to present a list of items where the user can click and select the items from the list. Items can be added by the programmer at design time or at runtime using a code. We can also write code to allow the user to add items to the list box or remove the items from the list box.

4.3.1 Adding Items to the List Box

a) Adding items using the String Collection Editor

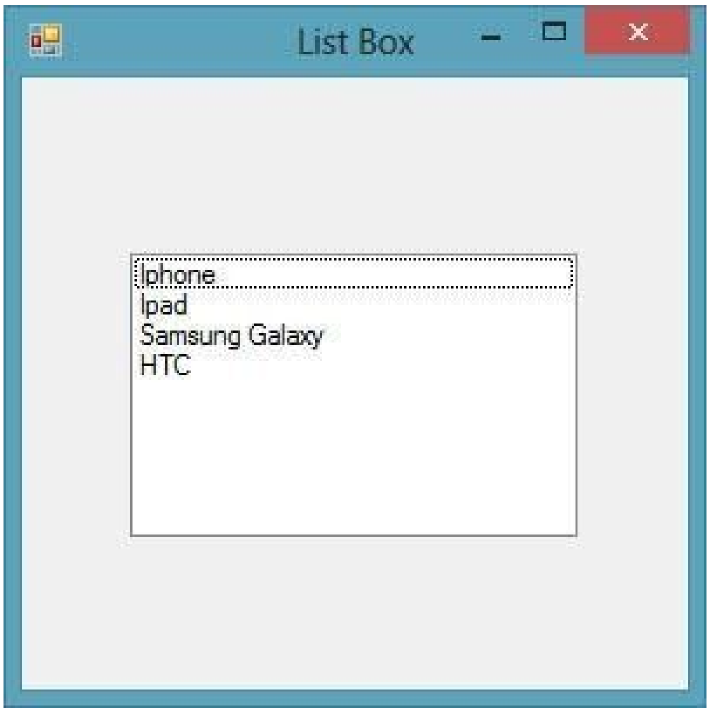
To demonstrate how to add items at design time, start a new project and insert a list box on the form then right-click on the list box to access the properties window. Next, click on collection of the Item property to launch the *String Collection Editor* whereby you can enter the items one by one by typing the text and press the Enter key, as shown in Figure 4.4. After clicking on the OK button, the items will be displayed in the text box, as shown in Figure 4.5



Enter the strings in the collection (one per line):

Iphone
Ipad
Samsung Galaxy
HTC

Figure 4.4



Iphone
Ipad
Samsung Galaxy
HTC

Figure 4.5

b) Adding Items using the Add() Method

Items can also be added at runtime using the **`Add()`** method. Before we proceed further, we should know that Visual Basic 2017 is an object-oriented programming language. Therefore, visual basic 2017 comprises objects. All objects have methods and properties, and they can be differentiated and connected by hierarchy. For a list box, an Item is an object subordinated to the object ListBox . The Item object comprises a method call **`Add()`** that is used to add items to the list box. To add an item to a list box, you can use the following syntax:

```
ListBox.Item.Add("Text")
```

Example 4.3 Adding an Item to a List Box

In this example, the item “Nokia” will be added to the end of the list, as shown in Figure 4.6

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
      ListBox1.Items.Add("Nokia")
End Sub
```

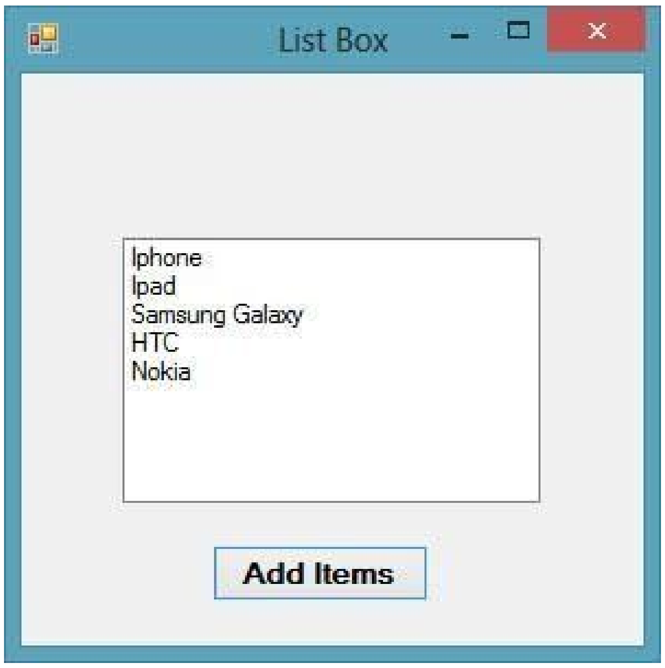


Figure 4.6

Example 4.4 Adding items to a List Box via an input box

In this example, you can allow the user to add items via a popup input box. First, we create a variable `myitem` and then assign a value to `myitem` via the `InputBox` function that store the input from the user. We then use the Add() method to add the user's item into the list box. The code is as follows:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
Dim myitem                              'declare the variable myitem
 myitem = InputBox("Enter your Item")
ListBox1.Items.Add(myitem)
End Sub
```

The runtime interface is as shown in Figure 4.7

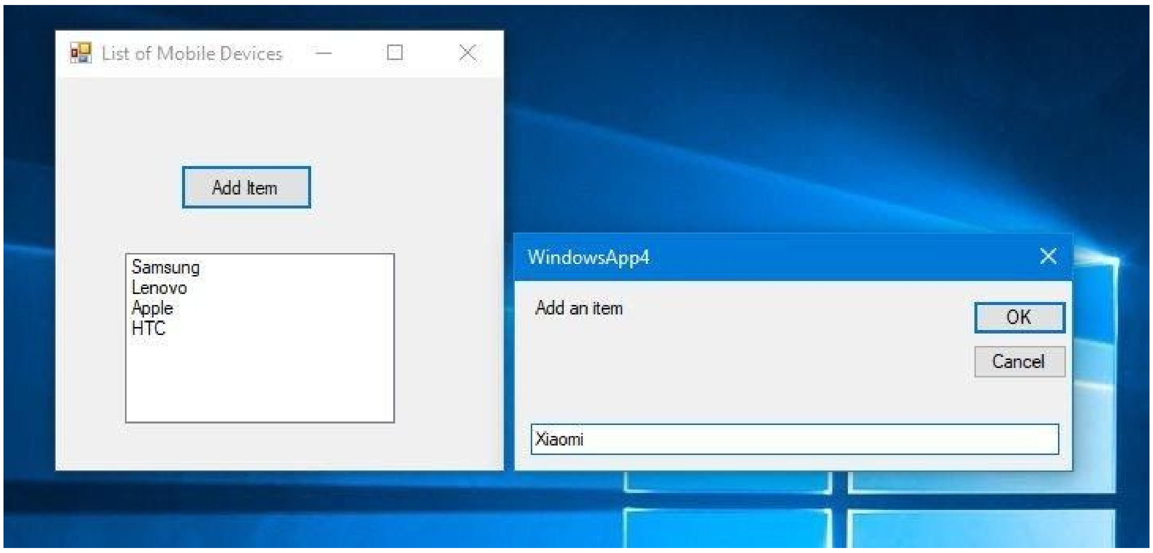
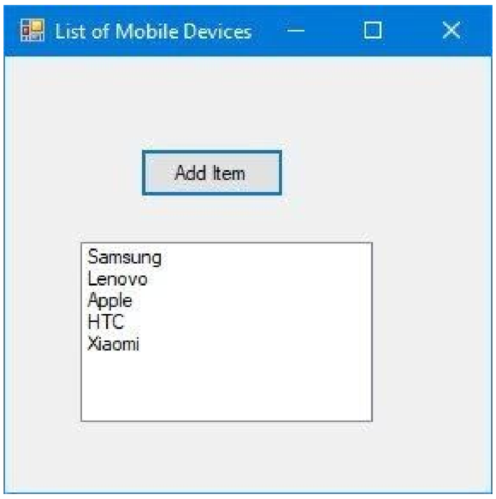


Figure 4.7

After typing the item 'Xiaomi" in the input box, the item will be added to the list box, as shown in Figure 4.8.



.

Figure 4.8

Example 4.5 Creating Geometric Progression

This is a program that generates a geometric progression and displays the results in a list box. Geometric progression is a sequence of numbers where each subsequent number is found by multiplying the previous number with a fixed number. This fixed number is called the common ratio. The common ratio can be a negative number, an integer, a fraction and any number but it must not be a zero or 1.

The formula to find the nth term of the geometric progression is **ar^{n-1}** , where a is the first number and r is the common ratio.

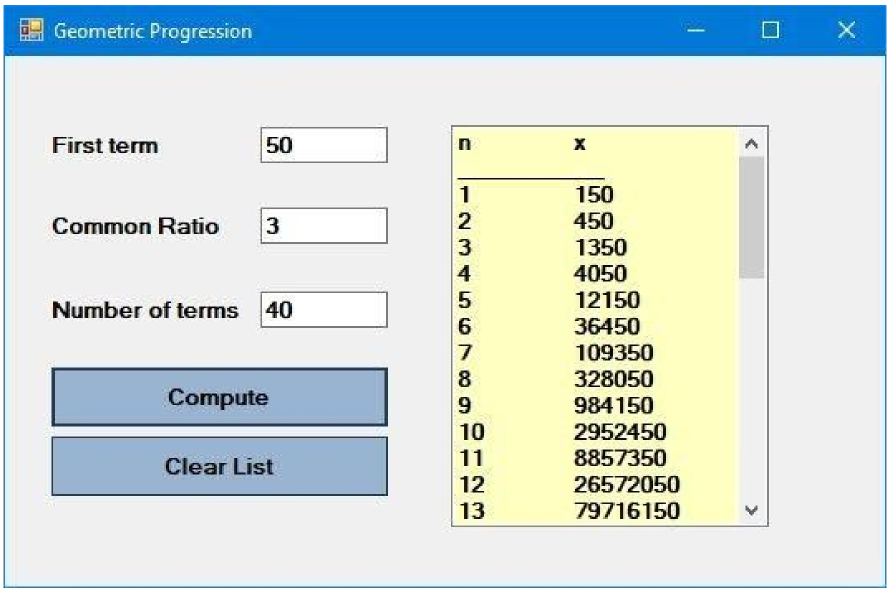
In this program, we employ the **`Do.... Loop Until`** statement to generate the numbers in a geometric progression. To design the UI, we need to insert three text boxes for the user to enter the first number, the common ratio and the number of terms. We also need to insert a list box to list the generated numbers. Besides that, a command button is needed for the user to generate the numbers in the geometric progression. In addition, we also add another button for clearing the list of generated numbers.

To add the numbers to the list box, we use the **`Add()`** method. The syntax is **`ListBox1.Items.Add(x)`**, where x can be any variable.

The code

```
Private Sub BtnComp_Click(sender As Object, e As EventArgs) Handles
```

```
BtnComp.Click
        Dim x, n, num As Double
        Dim r As Double
        x = TxtFirst.Text
        r = TxtCR.Text
        num = TxtN.Text
        MyList.Items.Add("n" & vbTab & "x")
        MyList.Items.Add("___________")

        n = 1
        Do
            x = x * r
            MyList.Items.Add(n & vbTab & x)
            n = n + 1
        Loop Until n = num + 1
End Sub

Private Sub BtnClr_Click(sender As Object, e As EventArgs) Handles
BtnClr.Click
        MyList.Items.Clear()
End Sub
```



First term: 50

Common Ratio: 3

Number of terms: 40

```
n       x
___________
1       150
2       450
3       1350
4       4050
5       12150
6       36450
7       109350
8       328050
9       984150
10      2952450
11      8857350
12      26572050
13      79716150
```

Figure 4.9

4.3.2 Removing Items from a List Box

To remove items at design time, simply open the String Collection Editor and delete the items line by line or all at once using the Delete key. To remove the items at runtime, you can use the `Remove` method, as illustrated in the following Example 4.5.

Example 4.6 Removing an item from a list box

In this example, add a button and label it “Remove Items”. Click on this button and enter the following code

```
Private Sub Button2_Click(sender As Object, e As EventArgs) Handles
Button2.Click
ListBox1.Items.Remove("Ipad")
End Sub
```

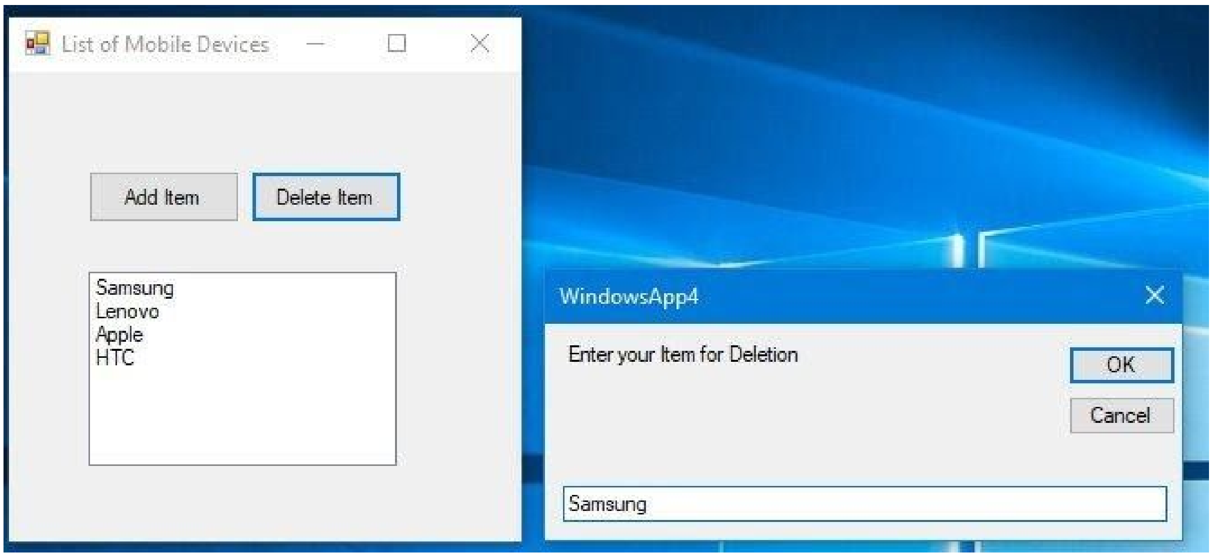
The item “Ipad” will be removed after running the program.

Example 4.7 Deleting an item from a list box via an input box

You can also allow the user to choose an item to delete via an Inputbox. To add this capability, insert a button at design time and change its text to Delete Item. Click on the button and enter the following statements in the code window:

```
Private Sub BtnDelete_Click(sender As Object, e As EventArgs) Handles
BtnDelete.Click
        Dim myitem
        myitem = InputBox("Enter your Item for Deletion")
        MyListBox.Items.Remove(myitem)
End Sub
```

The runtime interface is as shown in Figure 4.10. After entering the item Samsung in the input box and press OK, the item Samsung will be deleted from the list box.



Samsung
Lenovo
Apple
HTC

Enter your Item for Deletion

Samsung

Figure 4.10

To remove a selected item from the list box, using the following syntax:

```
Listbox1.Items.Remove(ListBox1.SelectedItem)
```

Example 4.8 Removing a selected item from a list box

```
Private Sub BtnDelSel_Click(sender As Object, e As EventArgs) Handles
BtnDelSel.Click
        MyListBox.Items.Remove(MyListBox.SelectedItem)
End Sub
```

When the user run the program and select an item to delete, the item will be deleted.

To remove multiple selected items from the list box, you need to use the If...End If structure together with the For…Next loop. Besides that, you also must ensure that the list box allows multiple selection. To enable multiple selection, set the selection mode to `MultipleSimple` in the list box properties windows. The code is as shown in Example 4.7.

Example 4.9 Removing multiple selected items in a list box

In this example, add an extra button to the previous example and label it as Clear Selected Items. Key in the following code:

```
Private Sub BtnDelSelected_Click(sender As Object, e As EventArgs)
Handles BtnDelSelected.Click
        If MyListBox.SelectedItems.Count > 0 Then
            For n As Integer = MyListBox.SelectedItems.Count - 1 To 0
Step -1
                'remove the current selected item from items
                MyListBox.Items.Remove(MyListBox.SelectedItems(n))
            Next n
        End If
    End Sub
```

To clear all the items at once, use the clear method, as illustrated in Example 4.8.

Example 4.10 Removing all items in a list box using the Clear method

In this example, add a button and label it “Clear the List”

```
Private Sub BtnClr_Click(sender As Object, e As EventArgs) Handles
BtnClr.Click

        MyListBox.Items.Clear()

End Sub
```

When you run the program and click the “Clear the List” button, all the items will be cleared.

The design interface for remove the items from the list box is as shown in Figure 4.11

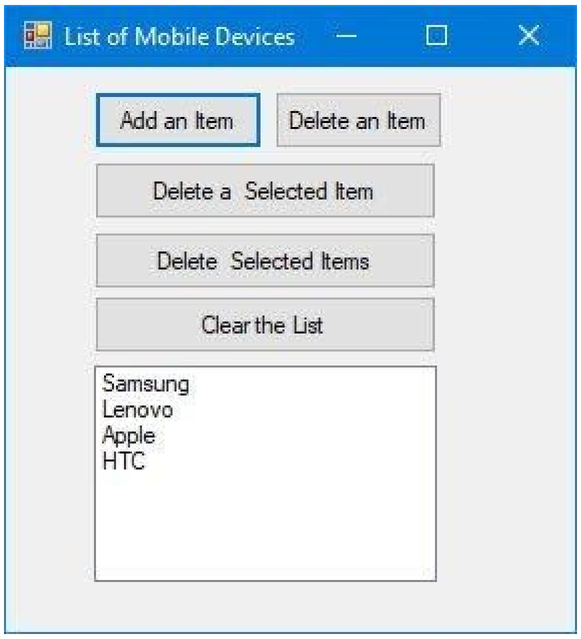


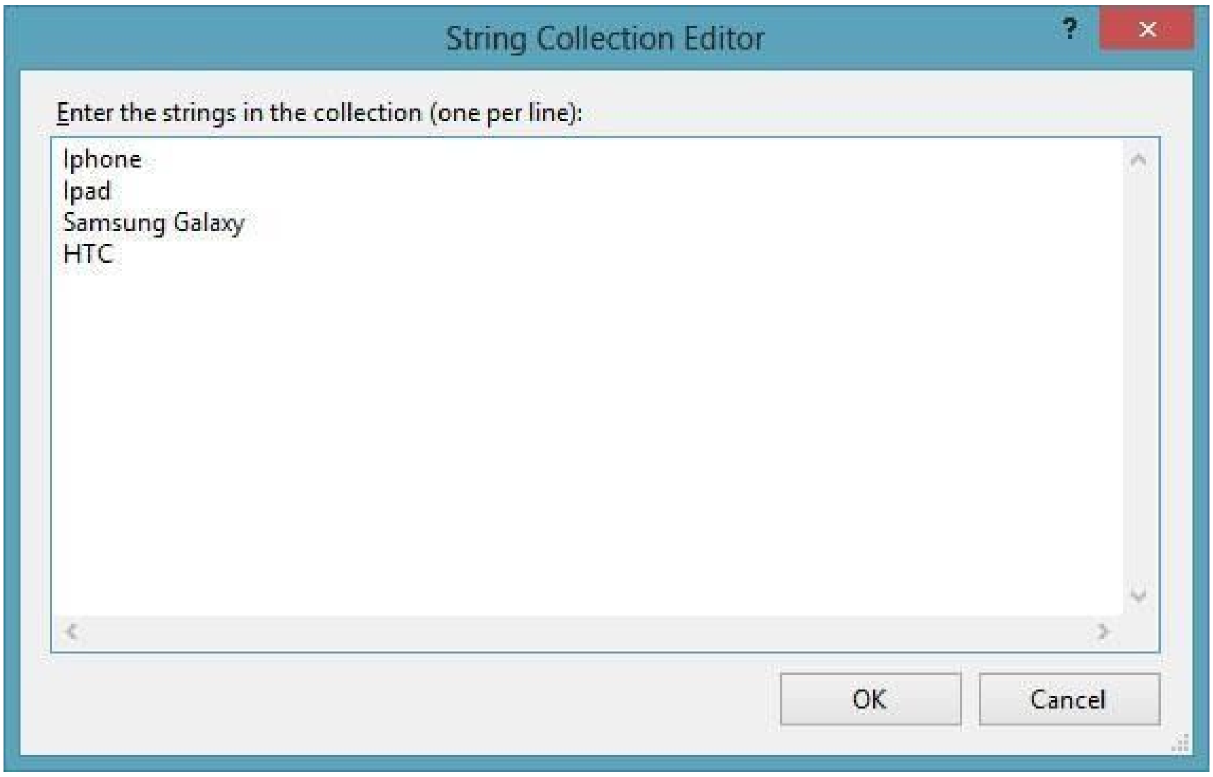
Figure 4.11

4.4 ComboBox

The function of the combo box is also to present a list of items where the user can click and select the items from the list. However, the combo box only displays one item at runtime. The user needs to click on the handle (small arrowhead) on the right of the combo box to see all the items in a drop-down list.

4.4.1 Adding Items to a combo box

To add items to the combo box at design time, use the String Collection Editor as shown in Figure 4.12. Besides that, if you want to display an item as the default text in the combo box when you run the program, enter the name of the item by replacing the text property of the combo box.



Iphone
Ipad
Samsung Galaxy
HTC

Figure 4.12

After clicking the handle of the right side of the combo box, the user will be able to view all the items, as shown in Figure 4.13

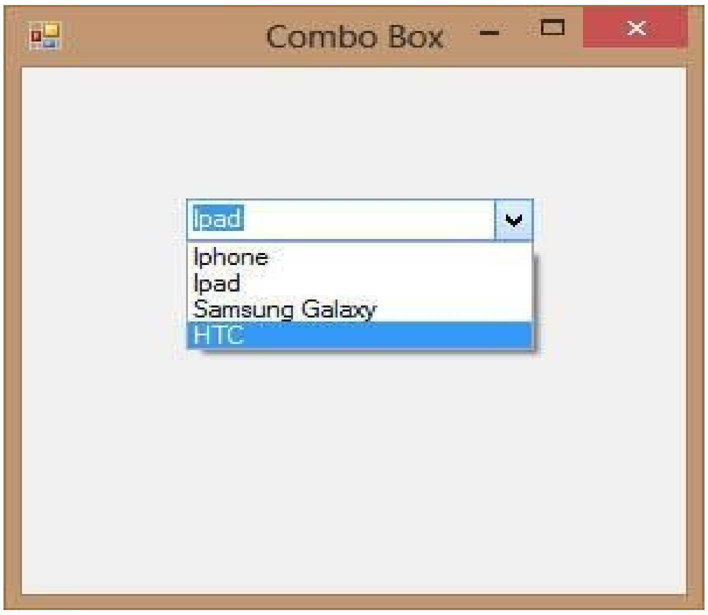


Figure 4.13

Besides that, you may add items using the **Add()** method. For example, if you wish to add an item to ComboBox1, key in the following statement. The output is as shown in Figure 4.14

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
ComboBox1.Items.Add("Nokia")
End Sub
```

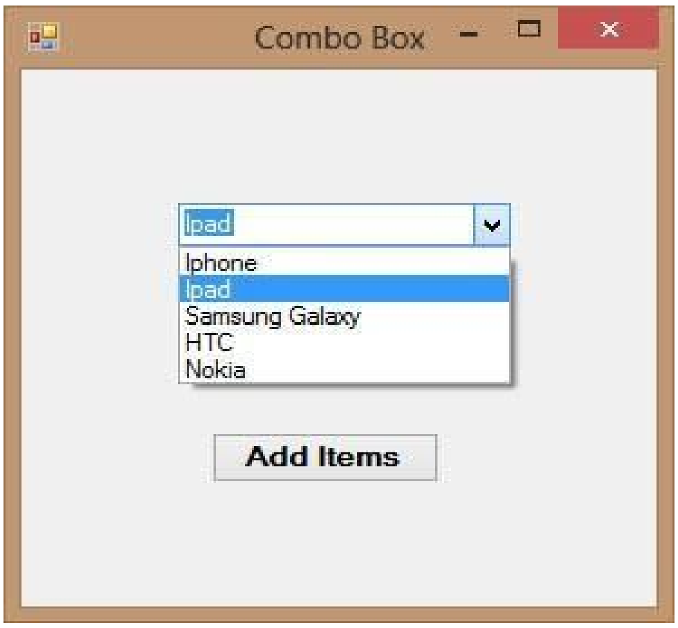


Figure 4.14

You can also allow the user to add items via an input box, as follows:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
Dim myitem
myitem = InputBox("Enter your Item")
ComboBox1.Items.Add(myitem)
End Sub
```

The runtime interface is shown in Figure 4.15

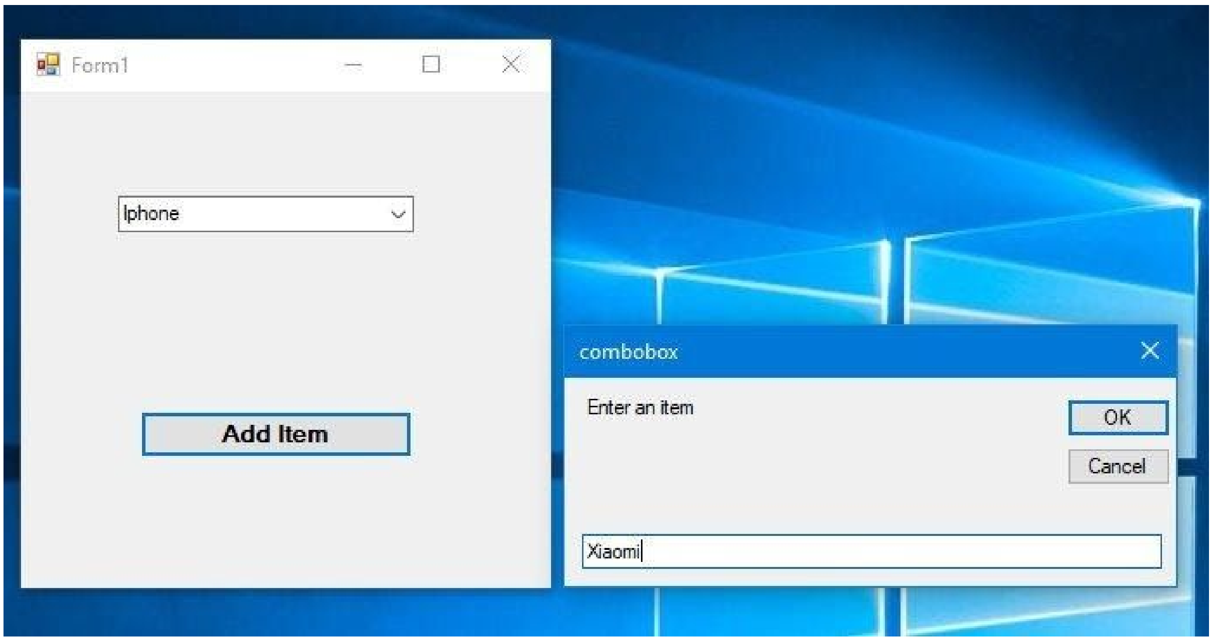


Figure 4.15

After you type the item ‘Xiaomi’ and click Ok, you can see that the item has been added to the combo box, as shown in Figure 4.16.

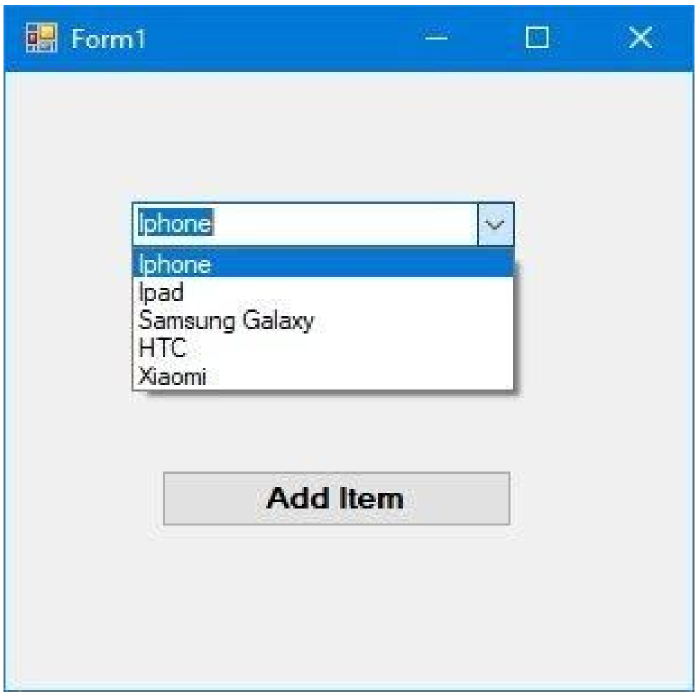


Figure 4.16

4.4.2 Removing Items from a Combo box

To remove items from the combo box at design stage, simply open the String Collection Editor and delete the items line by line or all at once using the Delete key.

To remove the items at runtime, use the Remove method, as illustrated in the following example. In this example, add a second button and label it “Remove Items”. Click on this button and enter the following code:

```
Private Sub Button2_Click(sender As Object, e As EventArgs) Handles
Button2.Click
ComboBox1.Items.Remove("Ipad")
End Sub
```

The item “Ipad” will be removed after running the program. You can also let the user select a specific item to delete, the code is as follows:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
MyCombo.Items.Remove(MyCombo.SelectedItem)
End Sub
```

To clear all the items at once, use the clear method, as illustrated in the following example. In this example, add a button and label it “Clear Items”

```
Private Sub Button3_Click(sender As Object, e As EventArgs) Handles
Button2.Click
ComboBox1.Items.Clear()
End Sub
```

Summary

- In section 4.1, you have learned how to work with a text box
- In section 4.2, you have learned how to work with a label
- In section 4.3.1, you have learned how to add items to a list box
- In section 4.3.2, you have learned how to remove items from a list box
- In section 4.4.1, you have learned how to add items to a combo box
- In section 4.4.2, you have learned how to remove items from a combo box

Chapter 5 Handling Images

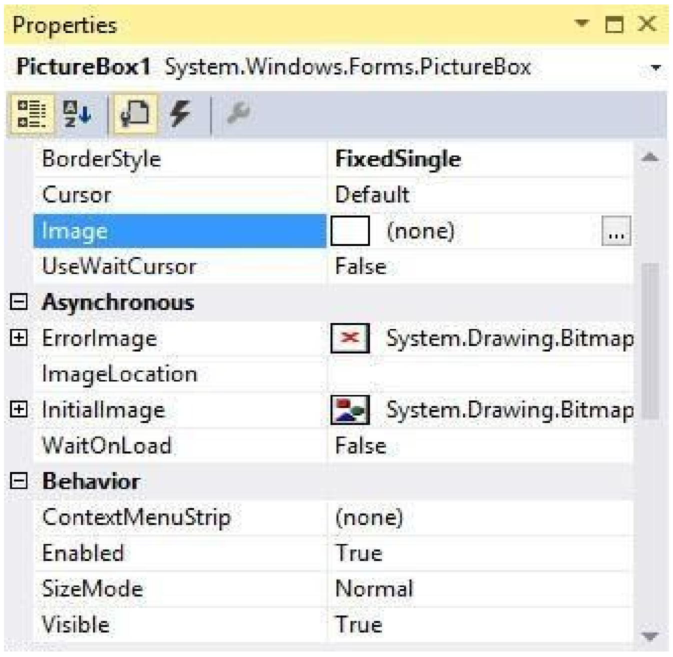
- ❖ Learn to load image in a picture box
- ❖ Learn to browse and load images using the common dialog

In this chapter, we shall learn how to load an image into the picture box at design time and at runtime. In addition, we shall also learn how to use a common dialog control to browse for image files in your local drives and then select and load an image into the picture box.

5.1 Loading an Image in a Picture Box

5.1.1 Loading an Image at Design Time

In this section, let us develop an image viewer . To create an image viewer, we insert a picture box in the form. Next, change its border property to `FixedSingle` and its background color to white. You might also want to change the `SizeMode` property of the image to `StretchImage` so that the image can fit in the picture box. In the properties window, scroll to the Image property, as shown in Figure 5.1. In the properties window, click on the grey button on the right of the Image item to bring out the "Select Source" dialog box , as shown in Figure 5.2



BorderStyle	FixedSingle
Cursor	Default
Image	(none)
UseWaitCursor	False
Asynchronous	
ErrorImage	System.Drawing.Bitmap
ImageLocation	
InitialImage	System.Drawing.Bitmap
WaitOnLoad	False
Behavior	
ContextMenuStrip	(none)
Enabled	True
SizeMode	Normal
Visible	True

Figure 5.1

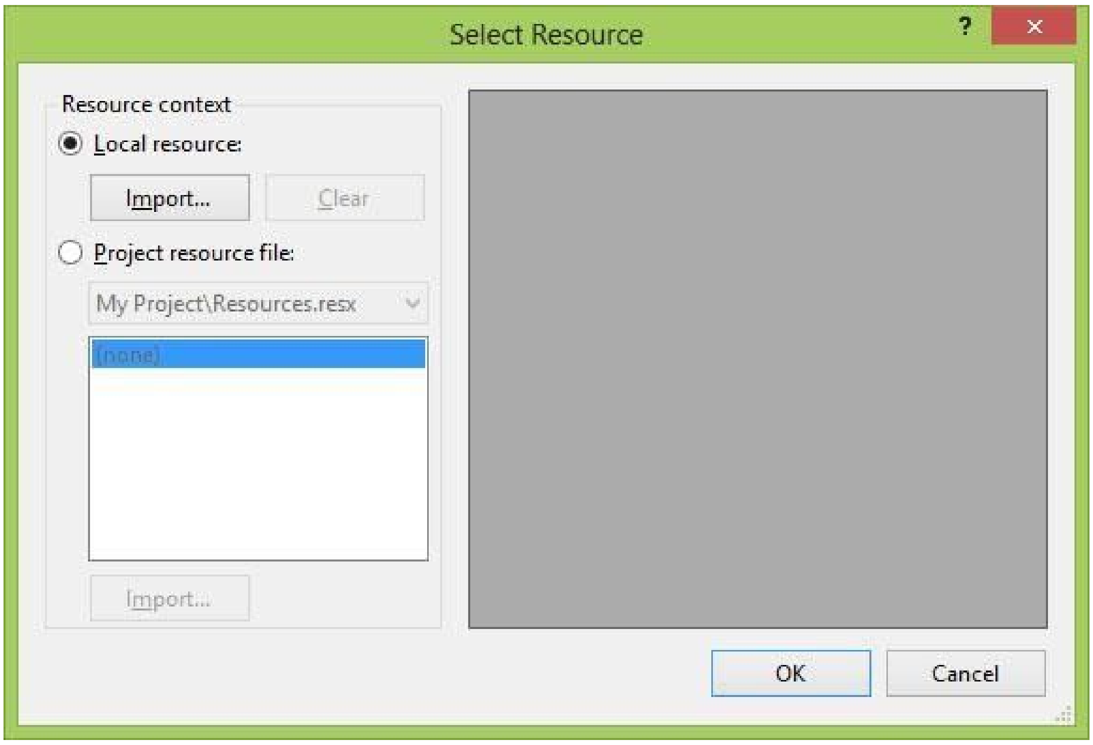


Figure 5.2

The next step is to select local resource and click on the Import button to view the available image files in your local drives, as shown in Figure 5.3. Finally, select the image you like and then click the open button, the image will be displayed in the picture box, as shown in Figure 5.4

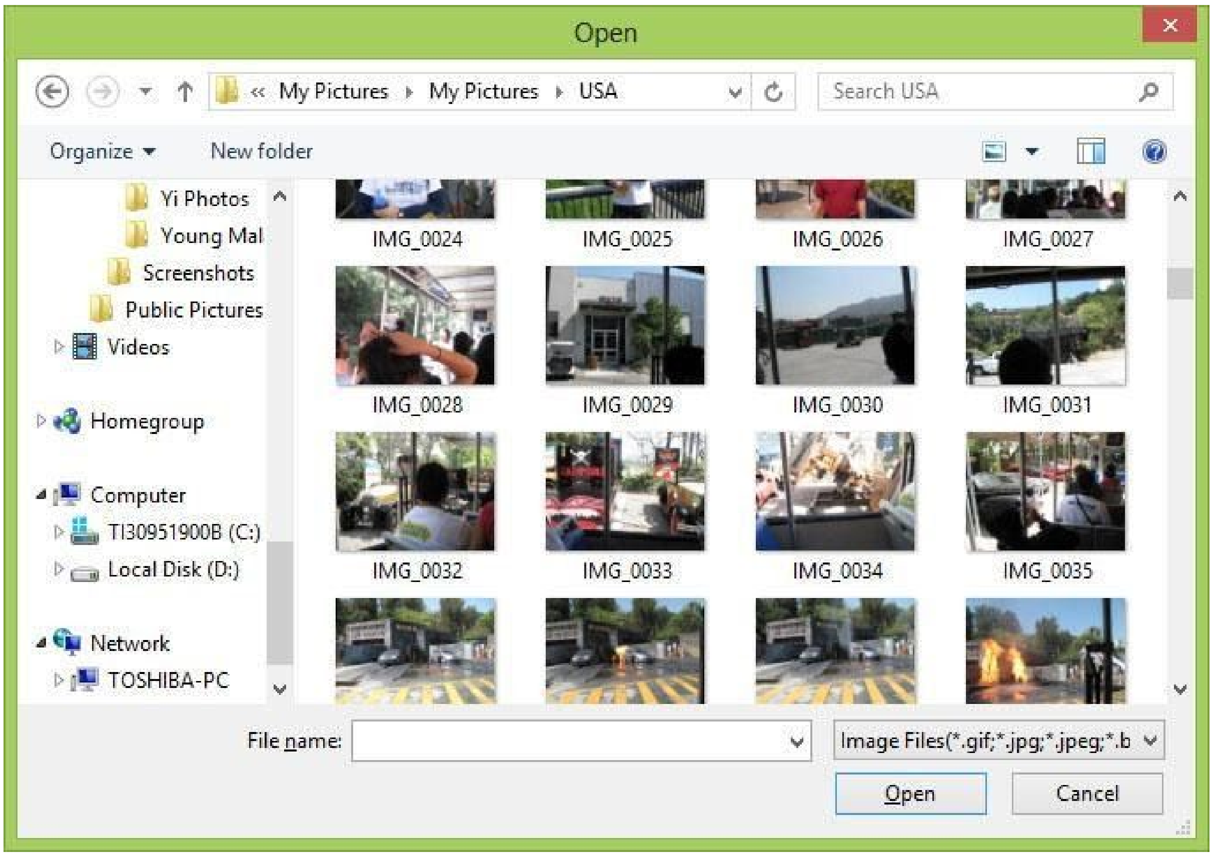


Figure 5.3

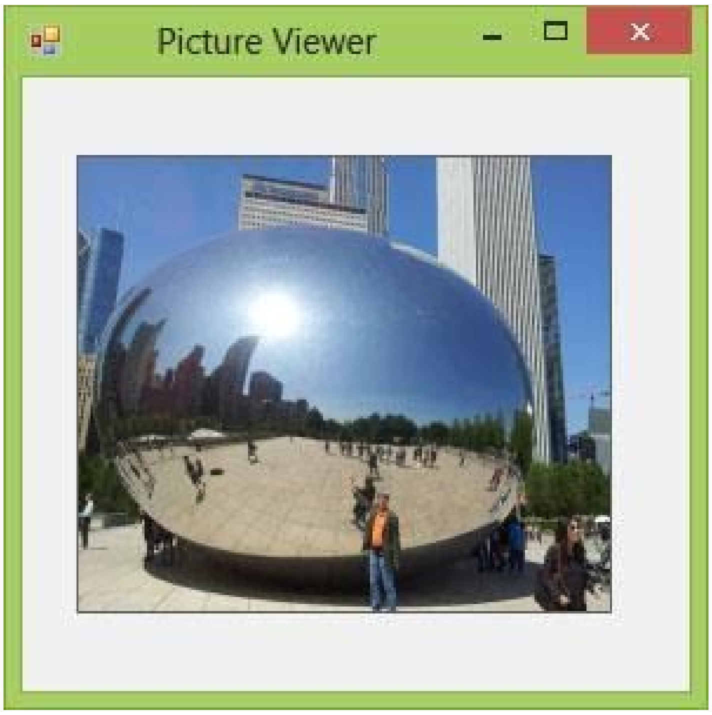


Figure 5.4

5.1.2 Loading an Image at Runtime

We can also load an image at runtime, using the code as follows:

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
PictureBox1.Image = Image.FromFile("C:\Users\Toshiba\Pictures\My
Pictures\USA\Chicago 2012.jpg")
End Sub
```

* You need to search for an image in your local drive and determine its path.

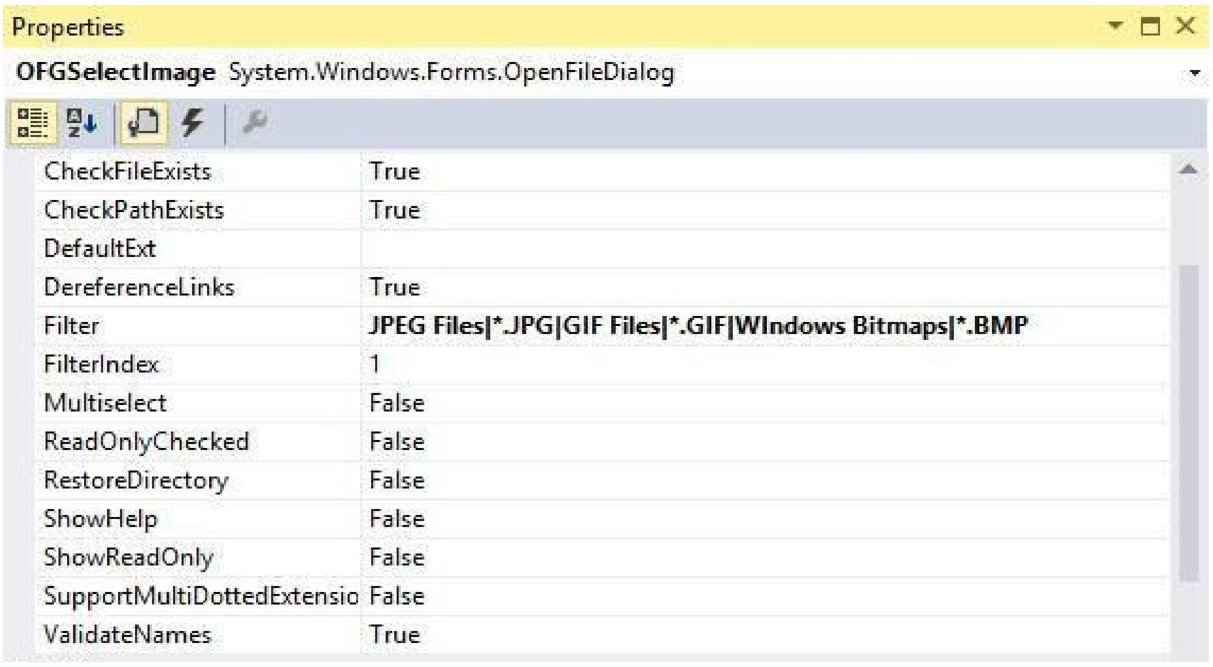
Running the program will display the same image in the picture box as in Figure 5.4

5.2 Loading an Image using Open File Dialog Control

To load an image in a picture box using the **OpenFileDialog** control, we must add the **OpenFileDialog** control on the form. This control will be invisible during runtime, but it facilitates the process of launching a dialog box and let the user browse his or her local drives and then select and open a file. For the **OpenFileDialog** to display all types of image files, we must specify the types of image files under the Filter property. Before that, rename **OpenFileDialog** as **OFGSelectImage**. Next, right click on the **OpenFileDialog** control to access its properties window. Beside the Filter property, specify the image files using the format:

```
JPEG Files| *.JPG|GIF Files|*.GIF|WIndows Bitmaps|*.BMP
```

as shown in Figure 5.5. These are the common image file formats. Besides that, you need to delete the default Filename.



Properties

OFGSelectImage System.Windows.Forms.OpenFileDialog

CheckFileExists	True
CheckPathExists	True
DefaultExt	
DereferenceLinks	True
Filter	JPEG Files|*.JPG|GIF Files|*.GIF|WIndows Bitmaps|*.BMP
FilterIndex	1
Multiselect	False
ReadOnlyChecked	False
RestoreDirectory	False
ShowHelp	False
ShowReadOnly	False
SupportMultiDottedExtensio	False
ValidateNames	True

Figure 5.5

Next, double-click on the View button and enter the following code:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
If OFGSelectImage.ShowDialog = Windows.Forms.DialogResult.OK Then
PictureBox1.Image = Image.FromFile(OFGSelectImage.FileName)
End If
End Sub
```

Press F5 to run the program and click the View button, a dialog box showing all the image files will appear, as shown in Figure 5.6.

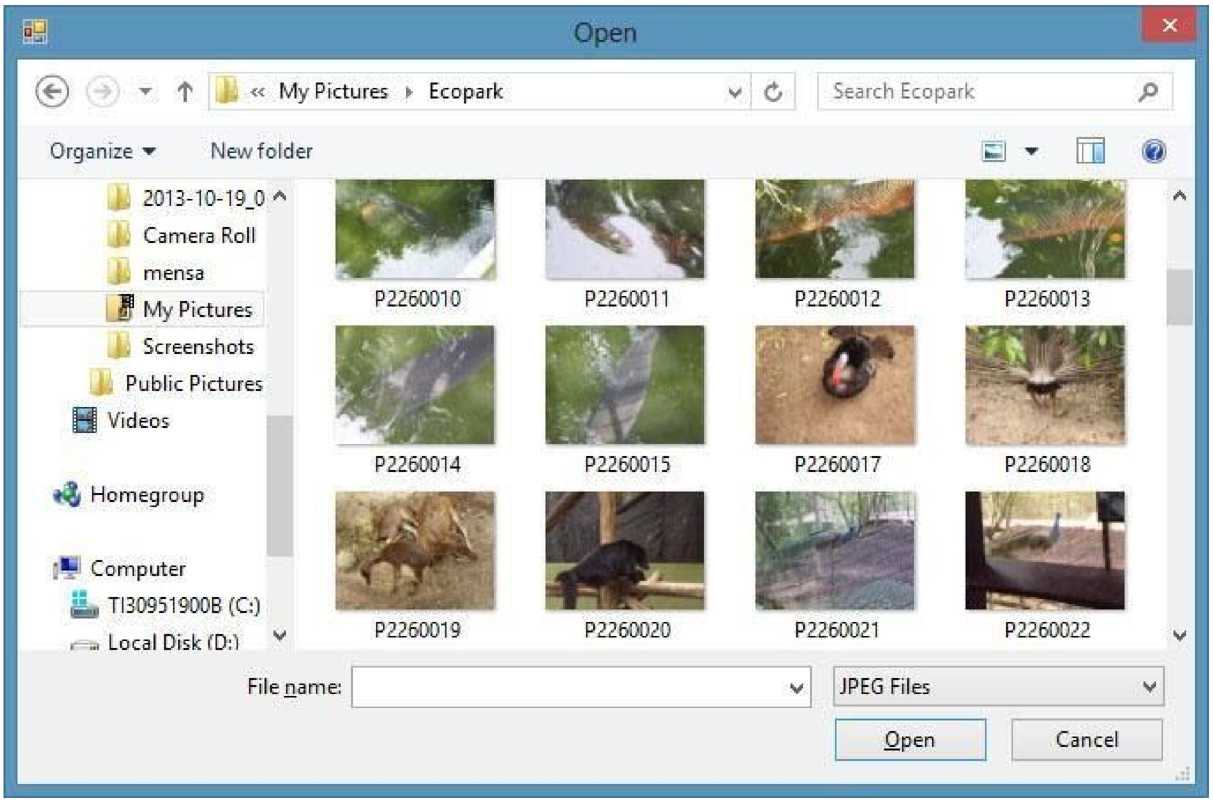


Figure 5.6

Notice that that the default image file is JPEG since we have placed it in the first place in the Filter property. Selecting and opening an image file will load it in the picture box, as shown in Figure 5.7.

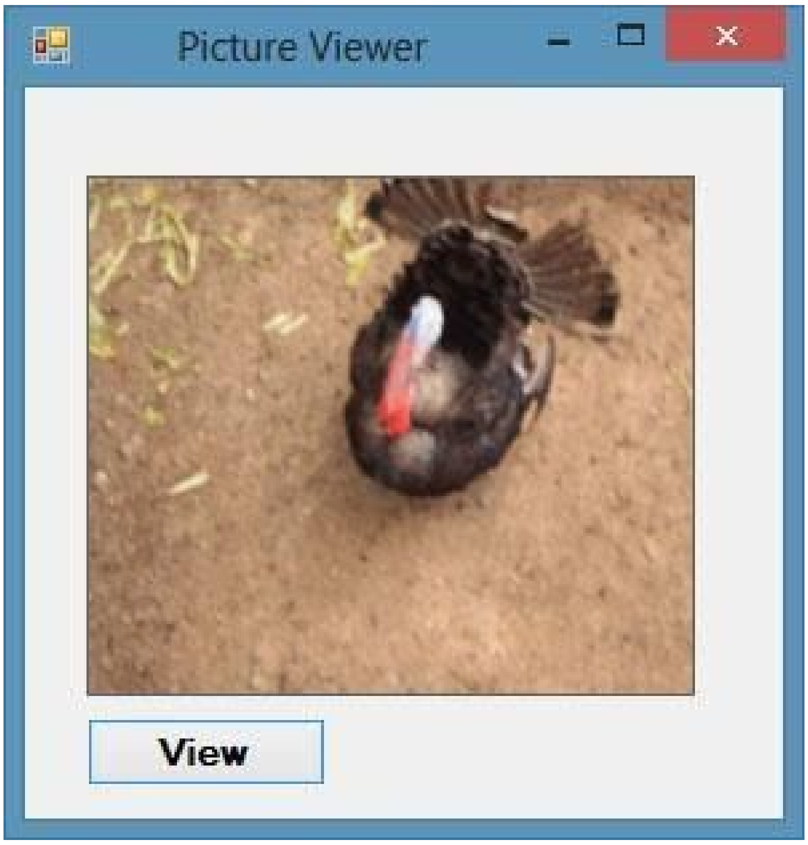


Figure 5.7

Summary

- In section 5.11, you have learned how to load an image at design time using the properties window
- In section 5.1.2, you have learned how to load an image at runtime
- In section 5.2, you have learned how to load an image using the OpenFileDialog control

Chapter 6 Working with Data

- ❖ Understand various data types
- ❖ Learn how to declare variables
- ❖ Learn how to declare constants

We deal with many kinds of data in our daily life like names, phone number, addresses, money, date, stock quotes, statistics and more. Similarly, in Visual Basic 2017, we must deal with all sorts of data, some of them can be mathematically calculated while some are in the form of text or other non-numeric forms. In Visual Basic 2017, data can be stored as variables, constants or arrays. The values of data stored as variables always change, just like the contents of a mailbox or the storage bin while the value of a constant remains the same throughout.

6.1 Data Types

Visual Basic 2017 classifies information into two major data types, the numeric data types and the non-numeric data type

6.1.1 Numeric Data Types

Numeric data types are types of data comprises numbers. Numeric data are divided into seven types based on the range of values they can store.

Calculations that only involve round figures can use Integer or Long integer. Computations that require high precision must use Single and Double precision data types; they are also called floating point numbers. For currency calculation, you can use the currency data types. Lastly, if even more precision is required to perform calculations that involve many decimal points, we can use the decimal data types. These data types are summarized in Table 6.1

Table 6.1 Numeric Data Types

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

6.1.2 Non-numeric Data Types

Non-numeric data comprises string data types, Date data types, Boolean data types, Object data type and Variant data type .

Table 6.1 Non-numeric Data Types

Type	Storage	Range
String(fixed length)	Length of string	1 to 65,400 characters
String(variable length)	Length + 10 bytes	0 to 2 billion characters
Date	8 bytes	January 1, 100 to December 31, 9999
Boolean	2 bytes	True or False

Object	4 bytes	Any embedded object
Variant(numeric)	16 bytes	Any value as large as Double
Variant(text)	Length+22 bytes	Same as variable-length string

6.1.3 Suffixes for Literals

Literals are values that you assign to data. In some cases, we need to add a suffix behind a literal so that VB can handle the calculation more accurately. For example, we can use num =1.3089# for a Double type data. The suffixes are summarized in Table 6.3.

Table 6.3 Suffixes

Suffix	Data type
&	Long
!	Single
#	Double
@	Currency

In addition, we need to enclose string literals within two quotations and date and time literals within two # sign. Strings can contain any characters, including numbers. The following are few examples:

```
memberName="Turban, John."
TelNumber="1800-900-888-777"
LastDay=#31-Dec-00#
ExpTime=#12:00 am#
```

6.2 Variables and Constants

Data can be stored as a variable or as a constant. Variables are like mailboxes in the post office. The content of the variables changes every now and then, just like the mailboxes.

6.2.1 Variable Names

Like the mailboxes, each variable must be given a name. To name a variable, you must follow a set of rules. The following are the rules when naming the variables in Visual Basic:

- It must be less than 255 characters
- No spacing is allowed
- It must not begin with a number
- Period is not permitted

Examples of valid and invalid variable names are displayed in Table 6.4

Table 6.4

Valid Names	Invalid Names
My_Computer	My.Computer
Smartphone123	123Smartphone
Long_Name_Can_beUSE	LongName&Canbe&Use *& is not acceptable

6.2.2 Declaring Variables

We must declare the variables before using them by assigning names and data types. If you fail to do so, the program will show an error. Variables are usually declared in the general section of the code windows using the `Dim` statement. The syntax is as follows:

```
Dim VariableName As DataType
```

If you want to declare more variables, you can declare them in separate lines or you may also combine more in one line, separating each variable with a comma, as follows:

```
Dim VariableName1 As DataType1, VariableName2 As DataType2, VariableName3
As DataType3
```

Example 6.1 Declaring Variables using Dim

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

```
Dim password As String
Dim yourName As String
Dim firstnum As Integer
Dim secondnum As Integer
Dim total As Integer
Dim doDate As Date
End Sub
```

You may also combine the statements in one line, separating each variable with a comma.

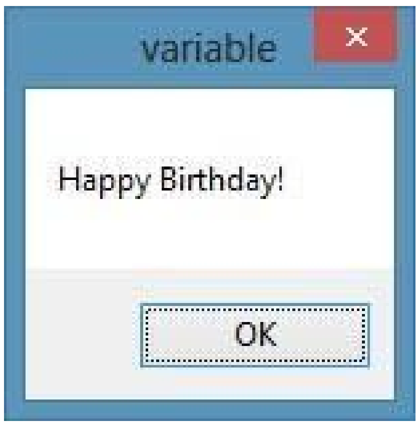
```
Dim password As String, yourName As String, firstnum As Integer......
```

For the string declaration, there are two possible forms, the variable-length string and the fixed-length string.

Example 6.2 Displaying Message using MsgBox

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
Dim YourMessage As String
YourMessage = "Happy Birthday!"
MsgBox(YourMessage)
End Sub
```

When you run the program, a message box that shows the text "Happy Birthday!" will appear, as shown in Figure 6.1



Happy Birthday!

Figure 6.1

For the fixed-length string, you must use the syntax as shown below:

```
Dim VariableName As String * n
```

Where n defines the number of characters the string can hold.

```
Dim yourName As String * 10
```

yourName can holds no more than 10 Characters.

6.2.3 Assigning Values to Variables

After declaring various variables using the **`Dim`** statements, we can assign values to those variables. The syntax 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, as illustrated in the following examples:

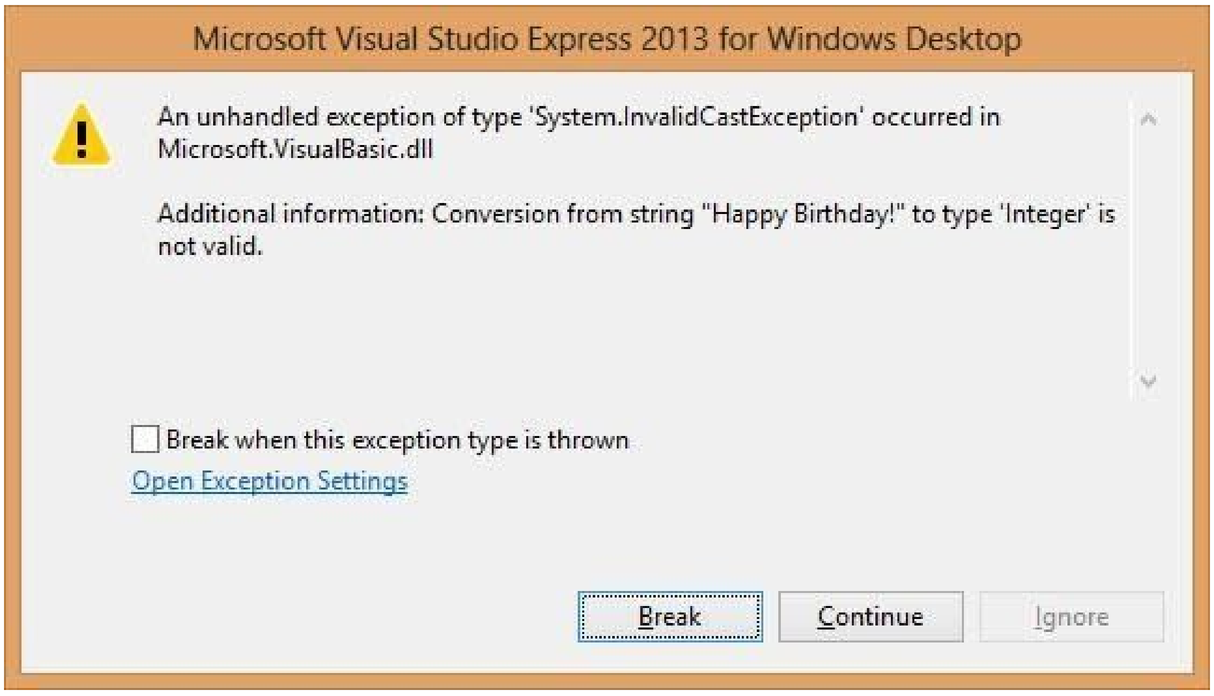
```
firstNumber=100
secondNumber=firstNumber-99
userName="John Lyan"
userpass.Text = password
Label1.Visible = True
Command1.Visible = false
Label4.text = textbox1.Text
ThirdNumber = Val(usernum1.Text)
total = firstNumber + secondNumber+ThirdNumber
MeanScore% = SumScores% / NumSubjects%
X=sqr (16)
TrimString= Ltrim ("Visual Basic", 4)
Num=Int(Rnd*6)+1
```

An error occurs when you try to assign a value to a variable of incompatible data type. For example, if you have declared a variable as an integer but you assigned a string value to it, an error occurred, as shown in Example 6.4.

Example 6.3 Incompatible Data Type

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
Dim YourMessage As Integer
YourMessage = "Happy Birthday!"
MsgBox(YourMessage)
End Sub
```

When you run the program, the following error messages will appear in a dialog box, as shown in Figure 6.2.


Microsoft Visual Studio Express 2013 for Windows Desktop

An unhandled exception of type 'System.InvalidCastException' occurred in Microsoft.VisualBasic.dll

Additional information: Conversion from string "Happy Birthday!" to type 'Integer' is not valid.

Figure 6.2

6.2.4 Scope of Declaration

Other than using the **Dim** keyword to declare the data, you can also use other keywords to declare the data. Three other keywords are **Private, Static** and **Public**. The forms are as shown below:

```
Private VariableName As Datatype
Static VariableName As Datatype
Public VariableName As Datatype
```

The keywords indicate the scope of declaration. **Private** declares a local variable, or a variable that is local to a procedure or module. However, Private is rarely used; we normally use Dim to declare a local variable. The **Static** keyword declares a variable that is being used multiple times, even after a procedure has been terminated. Most variables created inside a procedure are discarded by Visual Basic when the procedure is completed, **static** keyword preserve the value of a variable even after the procedure is terminated. **Public** is the keyword that declares a global variable, which means it can be used by all the procedures and modules of the whole program.

6.2.5 Declaring Constants

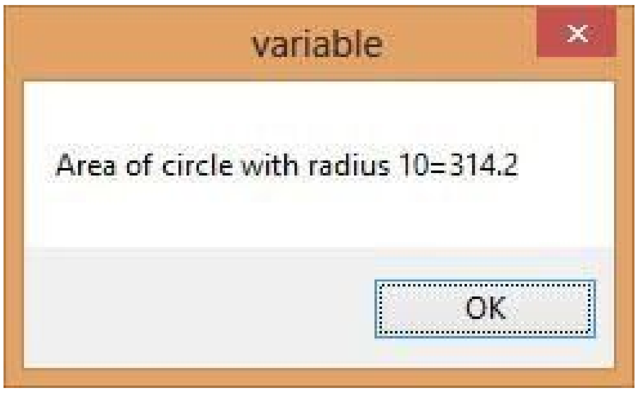
Constants are different from variables in the sense that their values do not change during the running of the program. The syntax to declare a constant is

```
Const ConstantName As Single=K
```

Example 6.4 Calculating the Area of Triangle

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
Const Pi As Single = 3.142
Dim R As Single = 10
Dim AreaCircle As Single
AreaCircle = Pi * R ^ 2
MsgBox("Area of circle with " & "radius" & R & "=" & AreaCircle)
End Sub
```

Running the program and clicking the OK button will produce the following message.



```
Area of circle with radius 10=314.2
```

Figure 6.3

Summary

- In section 6.11, you have understood numeric data types
- In section 6.1.2, you have understood non-numeric data types
- In section 6.1.3, you have learned how to use suffixes for literals
- In section 6.2.1, you have understood rules that govern variable names
- In section 6.2.2, you have learned how to declare variables
- In section 6.2.3, you have understood the scope of declaration of variables
- In section 6.2.3, you have learned how to declare a constant

Chapter 7 Array

- ❖ Understand the concept of array
- ❖ Learn how to declare a one-dimensional array
- ❖ Learn how to declare a two-dimensional array

7.1 Introduction to Arrays

An array is a group of variables with the same data type. When we work with a single item, we only need to use one variable. However, if we deal with a list of items of similar type , it is better to declare an array of variables instead of using a variable for each item

Imagine if we need to enter one thousand names, it would be very tedious to declare one hundred different names. Therefore, instead of declaring one thousand different variables, we just need to declare only one array. We differentiate each item in the array by using a subscript, the index value of each item, for example name(1), name(2),name(3) …….etc. , which will make declaring variables streamline and much more systematic.

7.2 Dimension of an Array

An array can be one dimensional or multidimensional. One dimensional array is a list of items that consists of one row of items or one column of items. A two-dimensional array is a table of items that is made up of rows and columns. The way to reference an element in a one-dimensional array is `ArrayName(x)`, where `x` is the index of the element. The way to reference an element in a two-dimensional array is `ArrayName(x,y)` , where (x,y) is the index of the element. Usually it is sufficient to use one dimensional and two-dimensional arrays, we only need to use higher dimensional arrays if we need to deal with more complex problems. Let me illustrate the arrays with tables.

Table 7.1 One-Dimensional Array

Student Name	SName(0)	SName(1)	SName(2)	SName(3)	SName(4)	SName(5)

Table 7.1 Two-Dimensional Array

SName(0,0)	SName(0,1)	SName(0,2)	SName(0,3)
SName(1,0)	SName(1,1)	SName(1,2)	SName(1,3)
SName(2,0)	SName(2,1)	SName(2,2)	SName(2,3)
SName(3,0)	SName(3,1)	SName(3,2)	SName(3,3)

7.3 Declaring Arrays

We use **`Public`** or **`Dim`** statement to declare an array, just as the way we declare a single variable. The **`Public`** statement declares an array that can be used throughout an application while the **`Dim`** statement declares an array that could be used only in a local procedure or module. The statement to declare a one-dimensional array is as follows:

```
Dim arrayName(n) As dataType
```

Please note that n does not indicate the number of elements in the array, it is one less than the number of elements (n-1) because the first element is always the zeroth element. The first element is arrayName(0), the second element is arrayName(1), the third element is arrayName(2) and so forth. The number of elements in an array is also known as length, we can retrieve the length of an array using the syntax **`arrayName.length`**

For example, the following statement declares an array that consists of 11 elements starting from CusName(0) through to CusName(10)

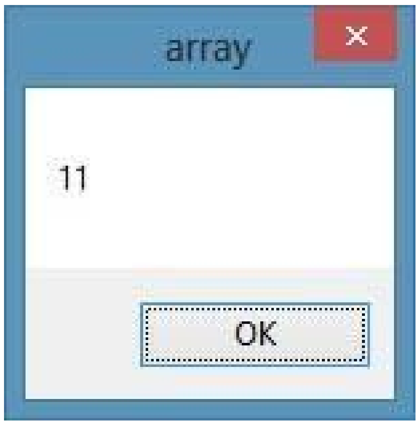
```
Dim CusName(10) As String
```

To find out the length of the array, you can write the following code:

Example 7.1 Find the Length of an Array

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
Dim CusName As String()
CusName = New String() {1, 2, 3}
MsgBox(CusName.Length)
End Sub
```

Running the program displays the length of the array i.e. 11, as shown in Figure 7.1



```
11
```

Figure 7.1

You might also declare an array with a non-zero starting index by initializing an index value other than zero, as follows:

```
Dim arrayname As DataType()
arrayName = New String(){1,2,3,....,n)
```

This array consists of n elements, starting with arrayName(1)

Example 7.2 Using the Length Property

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
Dim CusName As String()
CusName = New String() {1, 2, 3}
MsgBox(CusName.Length)
End Sub
```

The message box will display the length as 3.

he statement to declare a two-dimensional array is as follow, where m and n indicate the last indices in the array. The number of elements or the length of the array is (m+1) x (n+1)

```
Dim ArrayName(m,n) As dataType
```

Example 7.3 Find the Length of a Two-Dimensional Array

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
Dim CusName(5,6) As String
MsgBox(CusName.Length)
End Sub
```

The program produces a message box will display the length of the array, i.e.42, as shown in Figure 7.2

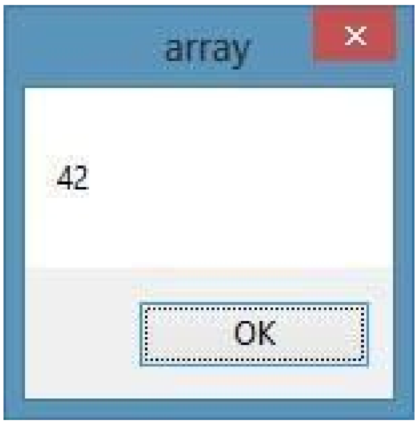


Figure 7.2

Example 7.4 Populating a List Box Involving an Array

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
Dim num As Integer
Dim CusName(5) As String
For num = 0 To 5
CusName(num) = InputBox("Enter the customer name", "Enter Name")
ListBox1.Items.Add(CusName(num))
Next
End Sub
```

This program will prompt the user to enter names in an input box for a total of 6 times and the names will be entered into a list box, as shown in Figure 7.3 and Figure 7.4

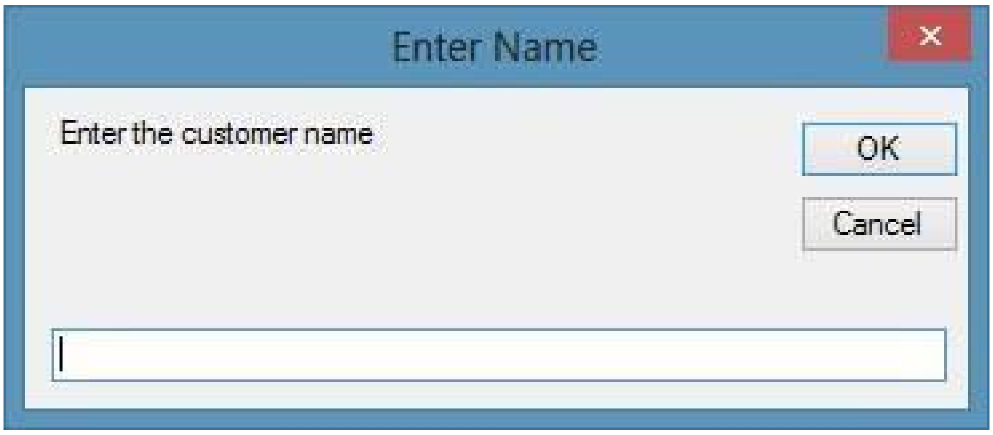
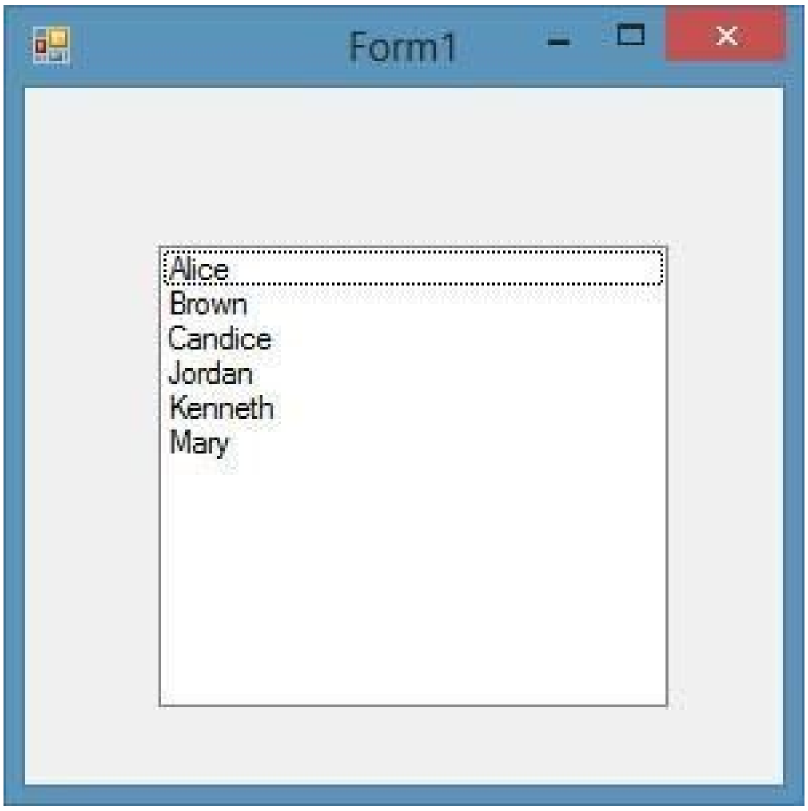


Figure 7.3



Alice
Brown
Candice
Jordan
Kenneth
Mary

Figure 7.4

Summary

- In section 7.1, you have understood the concept of arrays
- In section 7.2, you have understood dimension in arrays
- In section 7.3, you have learned how to declare an array

Chapter 8 Performing Mathematical Operations

- ❖ Recognize various mathematical operators
- ❖ Learn to write code that perform mathematical operations

Computer execute mathematical calculations much faster than human beings do. However, computer itself cannot perform any mathematical calculations without receiving instructions from the user. In Visual Basic 2017, we can write code to instruct the computer to perform mathematical calculations such as addition, subtraction, multiplication, division and many other kinds of mathematical operations.

8.1 Mathematical Operators

For Visual Basic 2017 to perform mathematical calculations, we need to write code that involves the use of various mathematical operators. The mathematical operators are remarkably like the normal arithmetic operators, only with some slight variations. The plus and minus operators are the same while the multiplication operator use the * symbol and the division operator use the / symbol. The list of Visual Basic 2017 mathematical operators is shown in table 8.1.

Table 8.1 Mathematical Operators

Operator	Mathematical function	Example
+	Addition	1+2=3
-	Subtraction	10-4=6
^	Exponential	3^2=9
*	Multiplication	5*6=30
/	Division	21/7=3
Mod	Modulus(returns the remainder of an integer division)	15 Mod 4=3
\	Integer Division(discards the decimal places)	19/4=4

8.2 Writing Code that Performs Mathematical Operations

Once you can recognize all the mathematical operators , it is quite easy to write code that can perform mathematical operations. First you need to think of a mathematical problem and the equations as well as formulas that are required to for solving it then write the code using those formulas and equations.

Example 8.1 Standard Arithmetic Calculations

In this program, you need to insert two text boxes, four labels and one button. Click the button and enter the code as shown below. This program performs standard arithmetic operations involving addition, subtraction, multiplication and division. The Code is as follows:

```
Private Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs)
Handles Button1.Click
Dim num1, num2, difference, product, quotient As Single
num1 = TextBox1.Text
num2 = TextBox2.Text
sum=num1+num2
difference=num1-num2
product = num1 * num2
quotient=num1/num2
LblSum.Text=sum
LblDiff.Text=difference
LblPro.Text = product
LblQt.Text = quotient
End Sub
```

Example 8.2 Pythagorean Theorem

This program uses Pythagorean Theorem to calculate the length of hypotenuse c given the length of the adjacent side a and the opposite side b. In case you have forgotten the formula for the Pythagorean Theorem, it is written as:

c^2=a^2+b^2

The code is as follows:

```
Private Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs)
Handles Button1.Click
Dim a, b, c As Single
a = TextBox1.Text
b = TextBox2.Text
c= (a^2+b^2)^(1/2)
Label3.Text=c
End Sub
```

Example 8.3: BMI Calculator

A lot of people are obese now and that could affect their health seriously. Obesity has proven by the medical experts to be a one of the main factors that brings many adverse medical problems, including the cardiovascular disease. If your BMI is more than 30, you are considered obese. You can refer to the following range of BMI values for your weight status.

Underweight = <18.5

Normal weight = 18.5-24.9

Overweight = 25-29.9

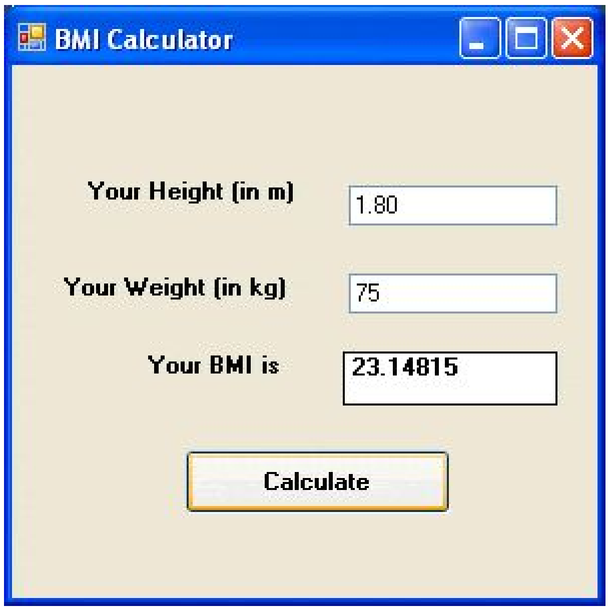
Obesity = BMI of 30 or greater

To calculate your BMI, you just need use a calculator or create a homemade computer program, and this is exactly what I am showing you here. The BMI calculator is a Visual Basic program that can calculate the BMI of a person based on the body weight in kilogram and the body height in meter. BMI can be calculated using the formula weight/(height)^2, where weight is measured in kg and height in meter. If you only know your weight and height in lb. and feet, then you need to convert them to the metric system.

```
Private Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs)
Handles Button1.Click
Dim height, weight, bmi As Single
height = TextBox1.Text
weight = TextBox2.Text
bmi = (weight) / (height ^ 2)
```

```
Label4.Text = bmi
End Sub
```

The output is shown in Figure 8.1. In this example, height is 1.80m (about 5 foot 11), your weight is 75 kg (about 168lb), and the BMI is 23.14815. The reading suggests that you are healthy. (Note: 1 foot=0.3048, 1 lb. =0.45359237 kilogram)



```
1.80
75
23.14815
```

Figure 8.1

Summary

- In section 8.1, you have recognized all mathematical operators
- In section 8.2, you have learned to write code that perform arithmetic operations

Chapter 9 String Manipulation

- ❖ Learn how to manipulate string using + and & signs
- ❖ Learn how to manipulate string using built-in functions

String manipulation means writing code to process characters like names, addresses, gender, cities, book titles, sentences, words, text, alphanumeric characters (@,#,$,%,^,&,*, etc.) and more. String manipulation is best demonstrated in the area of word processing which deals with text editing. A string is a single unit of data that made up of a series of characters that includes letters, digits, alphanumeric symbols etc. It is treated as the String data type and therefore it is non-numeric in nature which means it cannot be manipulated mathematically though it might consist of numbers.

9.1 String Manipulation Using + and & signs

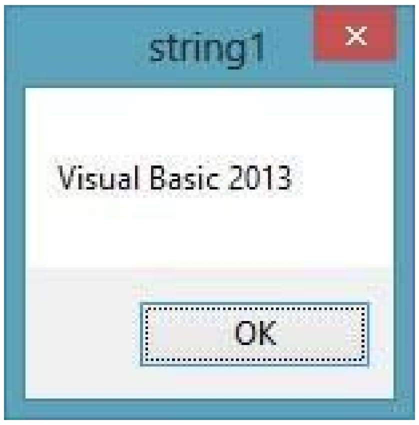
Strings can be manipulated using the & sign and the + sign, both perform the string concatenation which means combining two or more smaller strings into a larger string. For example, we can join "Visual" ,"Basic" and "2017" into "Visual Basic 2017" using "Visual" & "Basic" or "Visual "+"Basic", as shown in the Examples below:

Example 9.1 String Concatenation

```
Private Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs)
Handles Button1.Click
Dim text1, text2, text3, text4 As String
text1 = "Visual"
text2 = "Basic"
text3 = "2017"
text4 = text1 + text2 + text3
MsgBox(text4)
End Sub
```

The line **`text4=text1+ text2 + text3`** can be replaced by **`text4=text1 & text2 & text3`** and produces the same output. However, if one of the variables is declared as numeric data type, you cannot use the + sign, you can only use the & sign.

The output is shown in Figure 9.1


string1

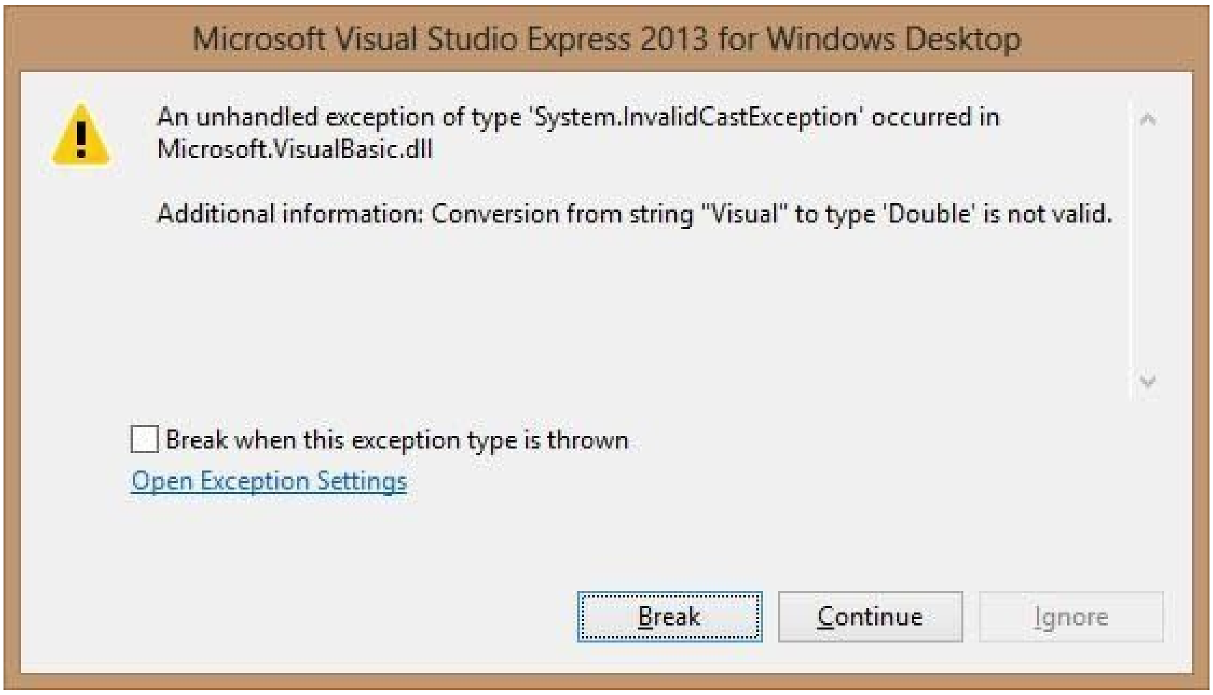
Visual Basic 2013

Figure 9.1

Example 9.2 Data Mismatch

```
Private Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs)
Handles Button1.Click
Dim text1, text3 As string
Dim Text2 As Integer
text1 = "Visual"
text2 = 22
text3 = text1 + text2
MsgBox(text3)
End Sub
```

This code will produce an error because of data mismatch. The error message appears as shown in Figure 9.2.


Microsoft Visual Studio Express 2013 for Windows Desktop

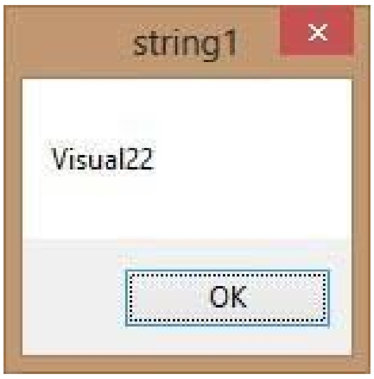
```
An unhandled exception of type 'System.InvalidCastException' occurred in
Microsoft.VisualBasic.dll

Additional information: Conversion from string "Visual" to type 'Double' is not valid.
```

Figure 9.2

However, using & instead of + will be alright.

```
Dim text1, text3 As string
Dim Text2 As Integer
text1 = "Visual"
text2 = 22
text3 = text1 & text2
MsgBox(text3)
```

The output is shown in Figure 9.3


string1

```
Visual22
```

Figure 9.3

9.2 String Manipulation Using Built-in Functions

A function is like a normal procedure, but the main purpose of the function is to accept a specific input and return a value which is passed on to the main program to finish the execution. There are numerous string manipulation functions that are built into Visual Basic 2017.

9.2.1 Len Function

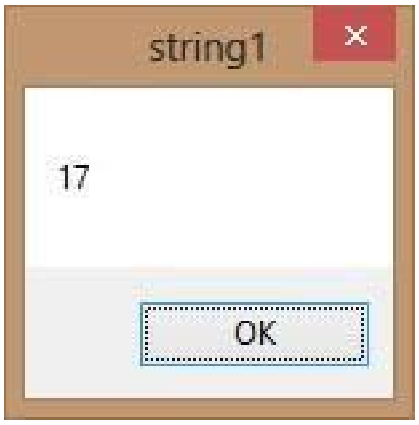
The Len function returns an integer value which is the length of a phrase or a sentence, including the empty spaces. The syntax is

```
Len("Phrase")
```

Example 9.3 Finding the Length of a Phrase

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
Dim MyText As String
MyText="Visual Basic 2017"
MsgBox(Len(MyText))
End Sub
```

The output is shown in Figure 9.4



```
17
```

Figure 9.4

9.2.2 Right Function

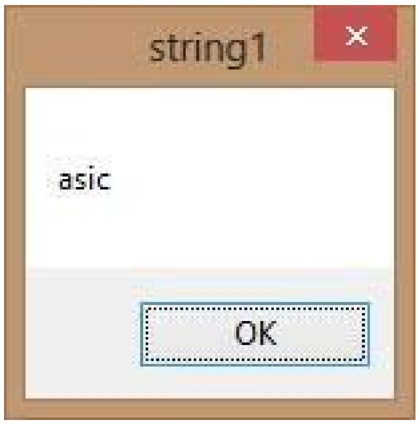
The Right function extracts the right portion of a phrase. The syntax is

```
Microsoft.VisualBasic.Right("Phrase",n)
```

Example 9.4 Extracting the Right Portion of a Phrase

```
Private Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs)
Handles Button1.Click
Dim MyText As String
MyText = "Visual Basic"
MsgBox(Microsoft.VisualBasic.Right(MyText, 4))
End Sub
```

The program returns four right most characters of the phrase , as shown in Figure 9.5



```
asic
```

Figure 9.5

9.2.3 Left Function

The Left function extract the left portion of a phrase. The syntax is

```
Microsoft.VisualBasic.Left("Phrase",n)
```

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

```
Microsoft.VisualBasic.Left ("Visual Basic", 4) = Visu
```

9.2.4 Mid Function

The Mid function is used to retrieve a part of text form a given phrase. The syntax of the Mid Function is

```
Mid(phrase, position,n)
```

* position is the starting position of the phrase from which the retrieving process begins, and n is the number of characters to retrieve.

Example 9.5 Retrieve Part of a Text Using Mid Function

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
Dim myPhrase As String
myPhrase = InputBox("Enter your phrase")
LblPhrase.Text = myPhrase
blExtract.Text = Mid(myPhrase, 2, 6)
End Sub
```

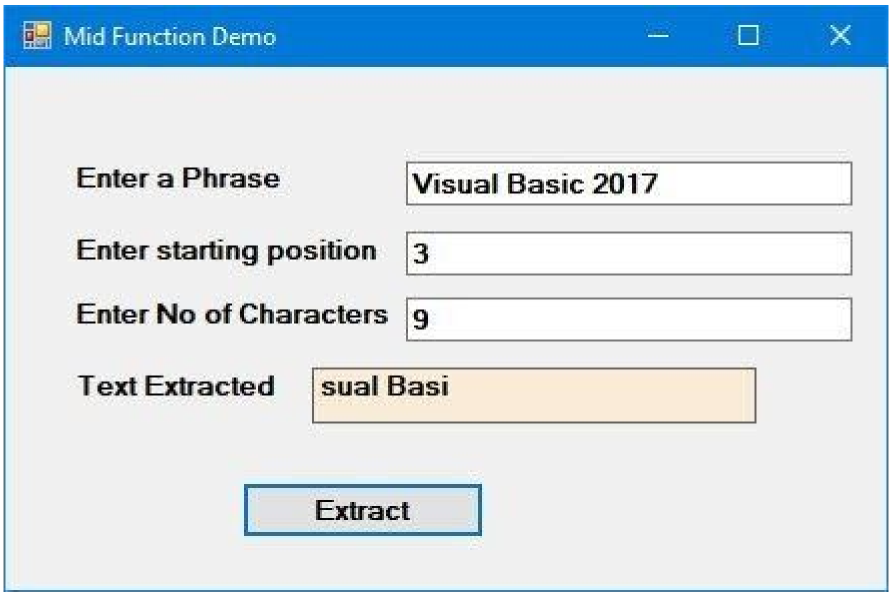
This program extracts six characters starting from position 2 of the phrase. For example, if you enter the phrase “Visual Basic 2017”, the extracted text is `isual`.

Example 9.6 Extracting Text from a Phrase

You can let the user decide the starting position of the text to be extracted as well as the number of characters to be extracted, as shown in the following code:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
BtnExtract.Click
        Dim myPhrase As String
        Dim pos, n As Integer
        myPhrase = TxtPhrase.Text
        pos = TxtPos.Text
        n = TxtNumber.Text
        LblExtract.Text = Mid(myPhrase, pos, n)
 End Sub
```

The runtime interface is shown in Figure 9.6



Enter a Phrase: Visual Basic 2017

Enter starting position: 3

Enter No of Characters: 9

Text Extracted: sual Basi

Figure 9.6

9.2.5 Trim Function

The Trim function trims the empty spaces on both side of the phrase. The syntax is

`Trim("Phrase")`

For example, Trim (" Visual Basic ") = Visual basic

Example 9.7 Trimming Both Side of a Phrase

```
Private Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs)
Handles Button1.Click
Dim myPhrase As String
myPhrase = InputBox("Enter your phrase")
Label1.Text = Trim(myPhrase)
End Sub
```

9.2.6 Ltrim Function

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

```
Ltrim("Phrase")
```

For example,

```
Ltrim("Visual Basic 2017") = Visual basic 2017
```

9.2.7 The Rtrim Function

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

```
Rtrim("Phrase")
```

For example,

```
Rtrim("Visual Basic 2017") = Visual Basic 2017
```

9.2.8 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 syntax 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 2017 ","Basic")=8
```

9.2.9 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.

The syntaxes are

```
Microsoft.VisualBasic.UCase(Phrase)
Microsoft.VisualBasic.LCase(Phrase)
```

For example,

```
Microsoft.VisualBasic.Ucase("Visual Basic") = VISUAL BASIC
Microsoft.VisualBasic.Lcase("Visual Basic") = visual basic
```

9.2.10 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. Please refer to the ASCII table in the Appendix for a complete list of the codes. 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 syntax of the Chr function is

```
Chr(charcode)
```

and the syntax of the Asc function is

```
Asc(Character)
```

The followings are some examples:

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

Summary

- In section 9.1, you have learned how to manipulate string using the & and + signs
- In section 9.2, you have learned how to manipulate string using various built-in function

Chapter 10 Using If…Then…Else

- ❖ Learn about the conditional operators
- ❖ Learn about the logical operators
- ❖ Learn how to use If…Then…Else

In this chapter, we shall learn how to write code that can make decisions and control the program flow in the process. The decision-making process is an important part of programming because it can solve problems in a smart way and provide useful output or feedback to the user.

For example, we can write program that instruct the computer to perform some tasks until some conditions are met. To control the program flow, we must use the conditional operators and the logical operators together with the **If...Then...Else** control structure.

10.1 Conditional Operators

Conditional operators allow a program to compare values and then decide what actions to take, whether to execute a program or terminate the program and more. They compare two values to see whether they are equal, one value is greater or less than the other value. The comparison will return a true or a false result. These operators are shown in Table 10.1.

Table 10.1 Conditional Operators

Operators	Description
=	Equal to
>	Greater than
<	Less than
>=	More than and equal to
<=	Less than and equal to

<>	Not equal to

10.2 Logical Operators

Sometimes we might need to make more than one comparison before a decision can be made and an action taken. In this case, using numerical comparison operators alone is not sufficient, we need to the logical operators. The logical operators are shown in Table 10.2

Table 10.2 Logical Operators

Operators	Description
And	Both sides must be true
Or	One side or other must be true
Xor	One side or other must be true but not both
Not	Negates true

The logical operators can be used to compare numerical data as well as non-numeric data . In making strings comparison, there are specific rules to follows: Upper case letters are less than lowercase letters, “A”<”B”<”C”<”D”…….<”Z” and number are less than letters.

10.3 Using If...Then...Else

To effectively control flow, we shall use the `If` control structure together with the conditional operators and logical operators. There are basically three types of `If` control structures, namely `If...Then` statement, `If...Then...Else` statement and `If...Then...ElseIf` statement.

10.3.1 If…Then Statement

This is the simplest control structure which instructs the computer to perform a specific action specified by the expression if the condition is true. However, when the condition is false, no action will be performed. The syntax for the `If…Then` statement is

```
If condition Then
Visual Basic 2017 expressions
End If
```

Example 10.1 Lucky Draw

```
Private Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs)
Handles Button1.Click
Dim myNumber As Integer
myNumber = TextBox1.Text
If myNumber > 100 Then
Label2.Text = " You win a lucky prize"
End If
End Sub
```

* When you run the program and enter a number that is greater than 100, you will see the “You win a lucky prize” message.

10.3.2 If...Then…Else Statement

Using only `If...Then` statement does not provide choices for the users. To provide a choice, we can use the `If...Then...Else` Statement. This control structure will ask the computer to perform a specific action specified by the expression if the condition is met. And when the condition is false, an alternative action will be executed. The syntax for the `If...Then...Else` statement is

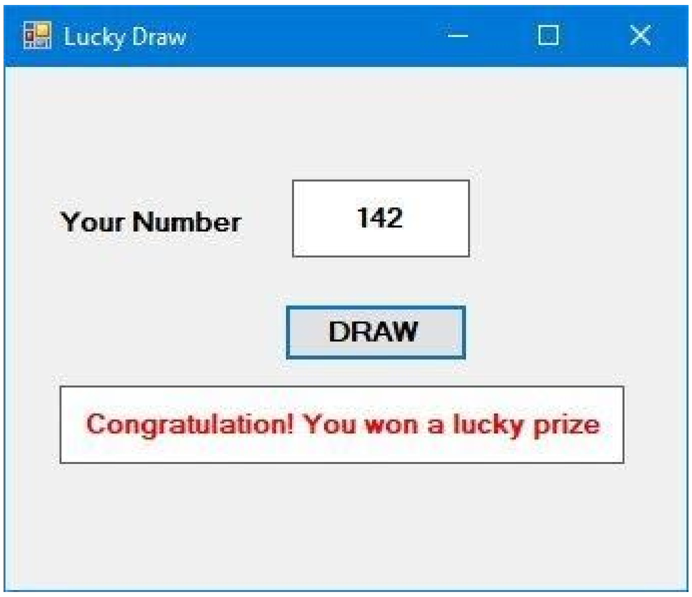
Example 10.2 Lucky Draw Simulation

This is a lucky draw simulation program. We use the **`Rnd()`** function to generate a random number between 0 and 1. In addition, we use the **`Int()`** function in the formula **`Int(Rnd() * 200) + 1`** to generate a random integer between 0 and 200. Next, we use the **`If...Then...Else`** statement to determine the condition for striking a lucky draw.

```
Private Sub BtnDraw_Click(sender As Object, e As EventArgs) Handles
BtnDraw.Click
        Dim myNumber As Integer
        myNumber = Int(Rnd() * 200) + 1
        LblNum.Text = myNumber
        If myNumber > 120 Then
            LblMsg.Text = " Congratulation! You won a lucky prize"
        Else
```

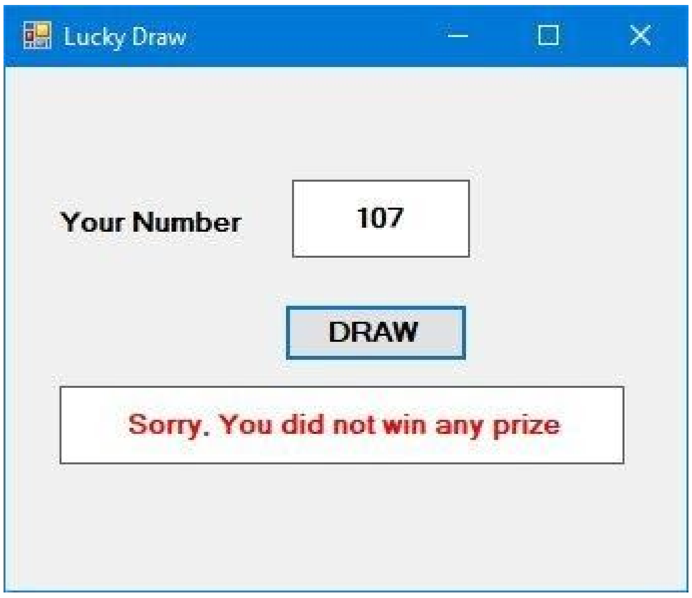
```
            LblMsg.Text = " Sorry, You did not win any prize"
        End If

    End Sub
```

* When you run the program and click the 'Draw" button, if the generated number is greater than 120, the message "Congratulation! You won a lucky prize" will be shown on the label . Otherwise, the label will show the "Sorry, you did not win any prize" message. The outcomes are shown in Figure 10.1 and 10.2



```
Congratulation! You won a lucky prize
```

Figure 10.1



```
Sorry. You did not win any prize
```

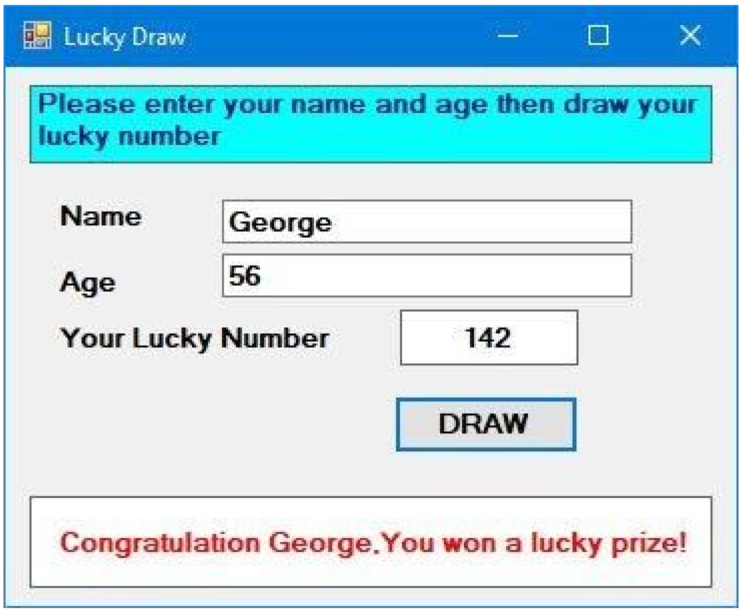
Figure 10.2

Example 10.3 Lucky Draw

Now we modified Example 10.2 and add in an additional constraint , age. In the program, we use the logical operator **And** beside the conditional operators. This means that both the conditions must be fulfilled for the conditions to be true, otherwise the second block of code will be executed. In this example, the lucky number must be more than 120 and the age must be more than 50 in order to win a lucky prize, any one of the above conditions not fulfilled will disqualify the user from winning a lucky prize. In addition, we make the program more interactive by adding name in the message. The code is as follows:

```
Dim myAge As Integer
Dim myName As String

Private Sub BtnDraw_Click(sender As Object, e As EventArgs) Handles
BtnDraw.Click
 Dim myNumber As Integer
 myAge = TxtAge.Text
 myName = TxtName.Text
 myNumber = Int(Rnd() * 200) + 1
 LblNum.Text = myNumber

  If myNumber > 120 And myAge > 50 Then
LblMsg.Text = " Congratulation " & myName & ",You won a lucky prize!"
 Else
LblMsg.Text = myName & " Sorry" & myName & ", you did not win any prize"
 End If

End Sub
```

The outcomes are shown in Figure 10.3 and Figure 10.4



Please enter your name and age then draw your lucky number

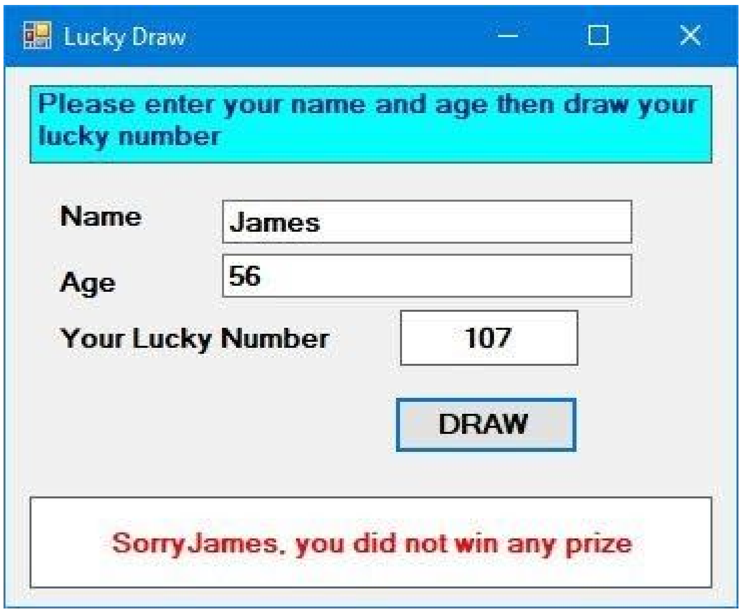
Name George

Age 56

Your Lucky Number 142

Congratulation George.You won a lucky prize!

Figure 10.3



Please enter your name and age then draw your lucky number

Name James

Age 56

Your Lucky Number 107

SorryJames, you did not win any prize

Figure 10.4

10.3.3 If….Then…ElseIf 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 structure of **`If…Then...ElseIf`** statement is

```
If condition Then

Visual Basic 2017 expression

ElseIf condition 1 Then

Visual Basic 2017 expression

ElseIf condition 2 Then

Visual Basic 2017 expression

.

.

Else

Visual Basic 2017 expression

End If
```

Example 10.4 Grade Generator

This program uses the **`If... ElseIf`** structure and the **`And`** logical operator to compute the grade for a specific mark.

```
Private Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs)
Handles Button1.Click
Dim Mark As Integer
Dim Grade as String
Mark = Val(TextBox1.Text)
If Mark>=80 Then
Grade="A"
ElseIf Mark>=60 And Mark<80 Then
Grade="B"
ElseIf Mark>=50 And Mark<60 Then
Grade="C"
Else
Grade="D"
End If
MsgBox("You grade is " & Grade)
End Sub
```

Running the program and will produce a form when the user can enter the mark. After entering the mark and clicking the ‘Compute Grade’ button, the grade will be displayed in a message box, as shown in Figure 10.5

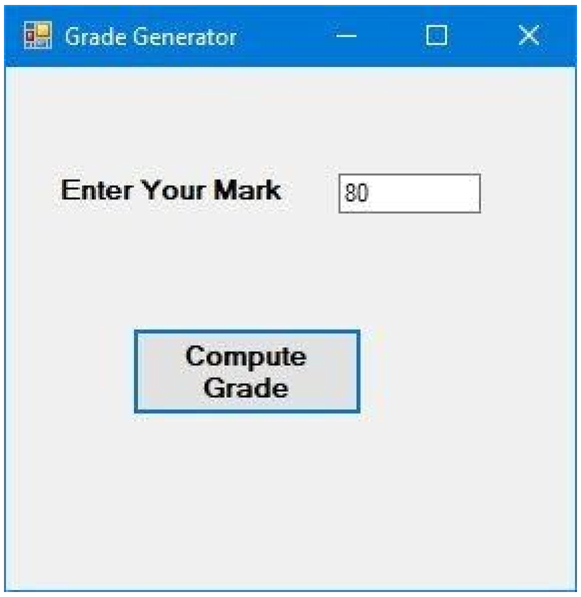
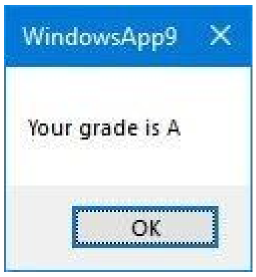


Figure 10.5

Clicking the OK produces a message box that shows the grade, as shown in Figure 10.6



Your grade is A

Figure 10.6

Summary

- In section 10.1, you have learned about the conditional operators
- In section 10.2, you have learned about the logical operators
- In section 10.3, you have learned how to write code involving If….Then…Else

Chapter 11 Using Select Case

- ❖ Learn how to write code for Select Case

The **`Select Case`** control structure is slightly different from the **`If...ElseIf`** control structure. The difference is that the **`Select Case`** control structure only make decision on one expression or dimension while the **`If…ElseIf`** statement control structure may evaluate only one expression, each **`If….ElseIf`** statement may also compute entirely different dimensions. **`Select Case`** is preferred when there exist multiple conditions.

The structure of the Select Case control structure in Visual Basic 2017 is as follows:

```
Select Case test expression
Case expression list 1
Block of one or more Visual Basic 2017 statements
Case expression list 2
Block of one or more Visual Basic 2017 Statements
.
.
Case Else
Block of one or more Visual Basic 2017 Statements
End Select
```

Example 11.1: Examination Grades

This program displays the examination results based on the grade obtained. The test expression here is grade. In this program, we insert a textbox for entering the grade, rename it as txtgrade. Next, insert a label for display the result, rename it as LblResult. Lastly, we insert a button, rename it as BtnCompute then enter the following code:

```
Private Sub BtnCompute_Click( )
Dim grade As String
grade=txtgrade.Text
Select Case grade
Case "A"
  LblResult.Text="High Distinction"
Case "A-"
```

```
  LblResult.Text="Distinction"
Case "B"
  LblResult.Text="Credit"
Case "C"
  LblResult.Text="Pass"
Case Else
  LblResult.Text="Fail"
End Select
End Sub
```

When the user runs the program, enters grade and clicks the 'Compute' button, the output is as shown in Figure 11.1

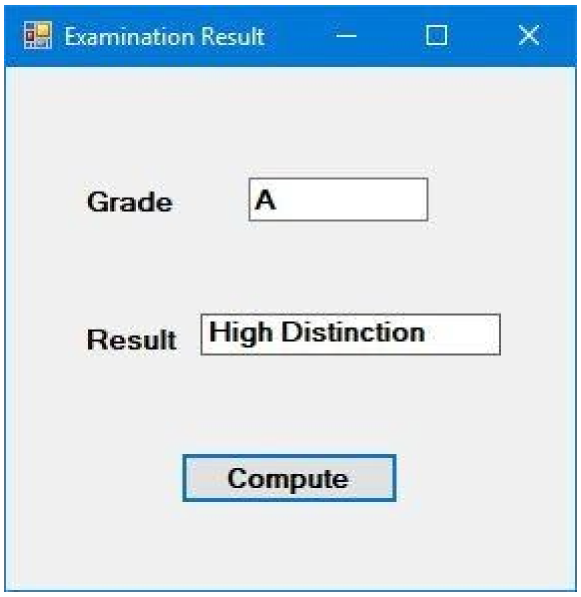


Figure 11.1

Example 11.2 Using Case Is

This example is like the previous example, but we use the **`Case Is`** keyword and the conditional operator >= to compute the results.

```
Private Sub BtnCompute_Click( )
Dim mark As integer
mark = TxtMark.Text
Select Case mark
  Case Is >= 85
```

```
LblComment.Text= "Excellence"
  Case Is >= 70
LblComment.Text= "Good"
  Case Is >= 60
LblComment.Text = "Above Average"
  Case Is >= 50
LblComment.Text= "Average"
  Case Else
LblComment.Text = "Need to work harder"
End Select
End Sub
```

The output is shown in Figure 11.2

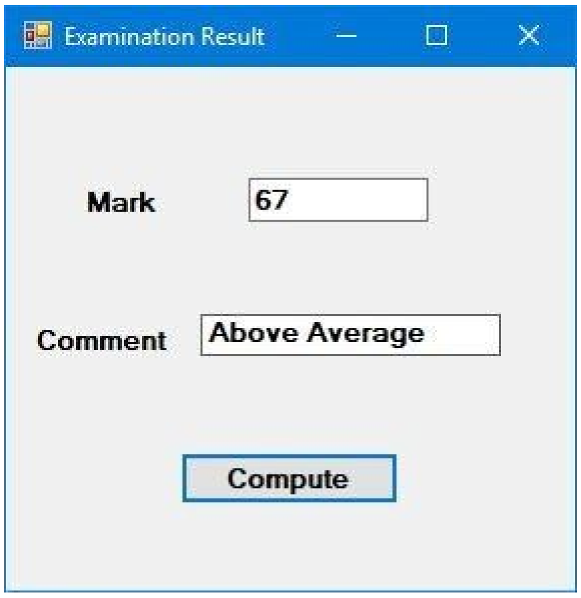


Figure 11.2

Example 11.3 Select Case using a Range of Values

Example 11.2 can be rewritten as follows:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
'Examination Marks
Dim mark As Single
mark = Textbox1.Text
Select Case mark
Case 0 to 49
  Label1.Text = "Need to work harder"
Case 50 to 59
```

```
  Label1.Text = "Average"
Case 60 to 69
  Label1.Text= "Above Average"
Case 70 to 84
  Label1.Text = "Good"
Case 85 to 100
  Label1.Text= "Excellence"
Case Else
Label1.Text= "Wrong entry, please reenter the mark"
End Select
End Sub
```

Example 11.4 Examination Grade

Grades in high school are usually presented with a single capital letter such as A, B, C, D or E. The grades can be computed as follow:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
'Examination Marks
Dim mark As Single
mark = TextBox1.Text
Select Case mark
Case 0 To 49
Label1.Text = "E"
Case 50 To 59
Label1.Text = "D"
Case 60 To 69
Label1.Text = "C"
Case 70 To 79
Label1.Text = "B"
Case 80 To 100
Label1.Text = "A"
Case Else
Label1.Text = "Error, please reenter the mark"
```

```
End Select
End Sub
```

The output is as shown in Figure 11.3

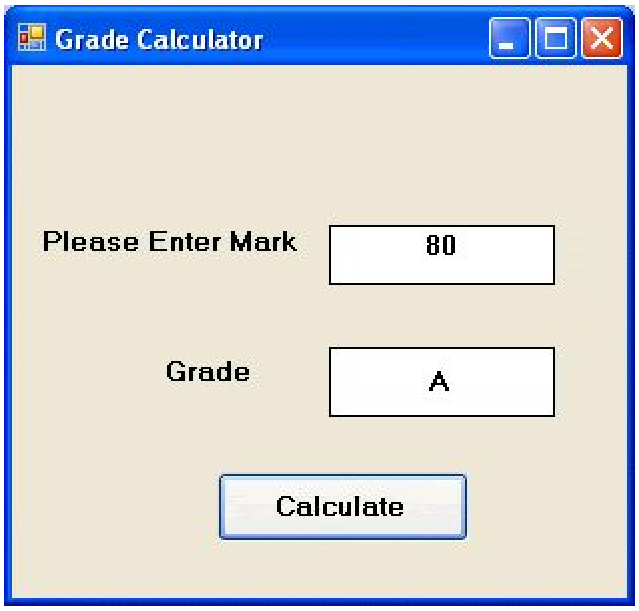


Figure 11.3

Summary

- In section 11.1, you have learned about the Select Case structure
- In section 11.2, you have learned how to write code using Select Case structure together with the conditional operators.

Chapter 12 Looping

- ❖ Learn how to write code using For Next Loop
- ❖ Learn how to write code using Do Loop
- ❖ Learn how to write code using While End…While Loop

In programming, we often need to write code that does a job repetitively until a specific condition is met, this process is called looping. For example, we can design a program that adds a series of numbers until the sum exceeds a specific value, or a program that asks the user to enter data repeatedly until he or she enters the word 'Finish'. There are three types of Loops, the **For...Next** loop, the **Do...loop** and the **While...End While** loop

12.1 For…Next Loop

The structure of a **For…Next** loop is as shown below:

```
For counter = startNumber to endNumber (Step increment)

One or more Visual Basic 2017 statements

Next
```

To exit a **For...Next** Loop, you can place the **Exit For** statement within the loop; and it is usually used together with the **If...Then** statement.

Example 12.1 Creating a Counter

```
Dim counter As Integer
For counter=1 to 10
ListBox1.Items.Add (counter)
Next
```

* This loop will enter number 1 to 10 into the list box.

Example 12.2 Sum of Numbers

```
Dim counter, sum As Integer
For counter=1 to 100 step 10
sum += counter
ListBox1.Items.Add (sum)
Next
```

* This loop will calculate the sum of the numbers as follows:

sum=0+10+20+30+40+……

Example 12.3 Step-down For Next Loop

```
Dim counter, sum As Integer
sum = 1000
For counter = 100 To 5 Step -5
sum - = counter
ListBox1.Items.Add(sum)
Next
```

*Notice that increment can be negative.

The program will compute the subtraction as follow:

1000-100 - 95 - 90 -……….

Example 12.4 Demonstrate Exit For

```
Dim n as Integer
For n=1 to 10
If n>6 then
Exit For
End If
Else
ListBox1.Items.Add (n)
Next
End If
Next
```

The process will stop when n is greater than 6.

12.2 Do Loop

The **`Do Loop`** structures are

a)

```
Do While condition
Block of one or more Visual Basic 2017 statements
Loop
```

b)

```
Do
Block of one or more Visual Basic 2012 statements
Loop While condition
```

c)

```
Do Until condition
Block of one or more Visual Basic 2012 statements
Loop
```

d)

```
Do
Block of one or more Visual Basic 2012 statements
Loop Until condition
```

Sometimes we need exit to exit a loop prematurely because a specific condition is fulfilled. The syntax to use is `Exit Do`. Let us examine the following examples:

Example 12.5 Do While…Loop

```
Do while counter <=1000
TextBox1.Text=counter
counter +=1
Loop
```

* The above example will keep on adding until counter >1000.

The above example can be rewritten as

```
Do
TextBox1.Text=counter
counter+=1
Loop until counter>1000
```

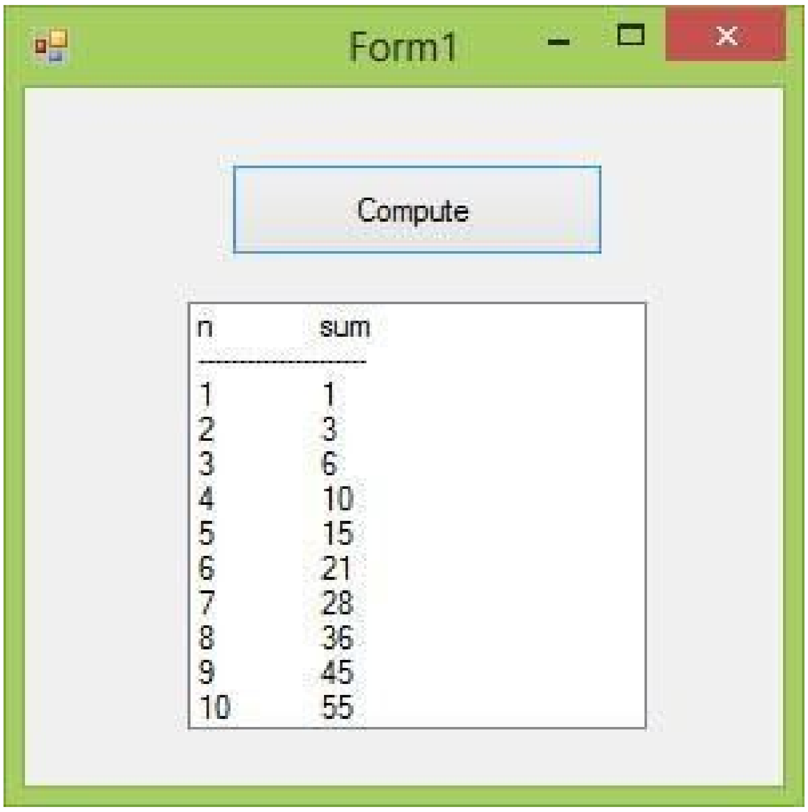
Example 12.6 Summation of Numbers

```
Private Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs)
Handles Button1.Click
Dim sum, n As Integer
ListBox1.Items.Add("n" & vbTab & "Sum")
ListBox1.Items.Add("---------------")
Do
n += 1
sum += n
ListBox1.Items.Add(n & vbTab & sum)
If n = 100 Then
Exit Do
End If
Loop
End Sub
```

* The loop in the above example can be replaced by the following loop:

```
Do Until n = 10
n += 1
sum += n
ListBox1.Items.Add(n & vbTab & sum)
Loop
```

The output is as shown in Figure 12.1



```
n       sum
----------------
1       1
2       3
3       6
4       10
5       15
6       21
7       28
8       36
9       45
10      55
```

Figure 12.1

12.3 While…End While Loop

The structure of a `While....End While` Loop is like the `Do` Loop.

```
While conditions
Visual Basic 2017 statements
End While
```

The loop is illustrated in Example 12.3

Example 12.3 Demonstrating While…End While Loop

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
Dim sum, n As Integer
ListBox1.Items.Add("n" & vbTab & "sum")
```

```
ListBox1.Items.Add("---------------")
While n <> 10
n += 1
sum += n
ListBox1.Items.Add(n & vbTab & sum)
End While
End Sub
```

Summary

- In section 12.1, you have learned how to write code using the For…Next Loop
- In section 12.2, you have learned how to write code using the Do Loop
- In section 12.3. you have learned how to write code using the While...End While Loop

Chapter 13 Sub Procedures

- ❖ Understand the concept of sub procedure
- ❖ Learn how to write code for a sub procedure

13.1 What is a Sub Procedure

A sub procedure is a procedure that performs a specific task and to return values, but it does not return a value associated with its name. Sub procedures is a program code by itself and but it is not an event procedure because it is not associated with a runtime procedure or a control. It is called by the main procedure to perform a specific task.

A sub procedure begins with a Sub keyword and ends with an End Sub keyword. The program structure of a sub procedure is as follows:

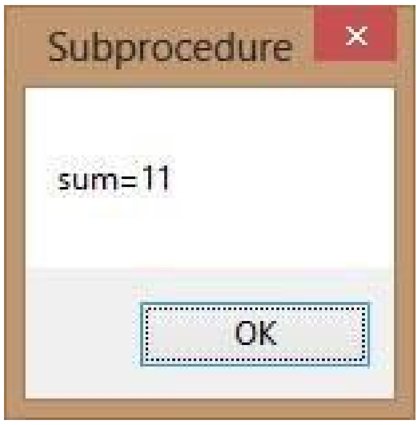
```
Sub ProcedureName(arguments)

Statements

End Sub
```

Example 13.1 A Sub Procedure that Adds Two Numbers

This sub procedure adds two values that are specified as the arguments. The main program can reference a procedure by using its name together with the arguments in the parentheses.

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
sum(5, 6)
End Sub
Sub sum(a As Single, b As Single)
MsgBox("sum="& a + b)
End Sub
```

Running the program produces a message box as shown in Figure 13.1



```
sum=11
```

Figure 13.1

Example 13.2: Password Cracker

This password cracking program that can generate possible passwords and then compares each of them with the actual password. If the generated password found to be equal to the actual password, login will be successful. In this program, a timer is inserted into the form and it is used to do a repetitive job of generating the passwords.

We create a password generating procedure **`generate ()`** and it is called by the **`Timer1_Tick()`** event so that the procedure is repeated after every interval. The interval of the timer can be set in its properties window where a value of 1 is 1 millisecond, so a value of 1000 is 1 second: the smaller the value, the shorter the interval. However, do not set the timer to zero because if you do that, the timer will not start. We shall set the Timer interval at 100 which is equivalent to 0.1 second.

The **`Timer1.Enabled`** property is set to false so that the program will only start generating the passwords after the user clicks on the Generate button. `Rnd` is a function that generates a random number between 0 and 1. Multiplying Rnd by 100 will obtain a number between 0 and 100. Int is a function that returns an integer by ignoring the decimal part of that number.

Therefore, **`Int(Rnd*100)`** will produce a number between 0 and 99, and the value of **`Int(Rnd*100)+100`** will produce a number between 100 and 199. Finally, the program uses **`If…Then…Else`** to check whether the generated password is equal the actual password or not; and if they are equal, the passwords generating process will be terminated by setting the **`Timer1.Enabled`** property to false. The code is as follows:

```
Dim password As Integer Dim crackpass As Integer
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
Timer1.Enabled = True
End Sub
Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles
Timer1.Tick
generate()
If crackpass = password Then
Timer1.Enabled = False
Label1.Text = crackpass
MsgBox("Password Cracked!Login Successful!")
Else Label1.Text = crackpass
Label2.Text = "Please wait..."
End If
End Sub
Sub generate()
crackpass = Int(Rnd() * 100) + 100
End Sub
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
password = 123
End Sub
```

The output is shown in Figure 13.2

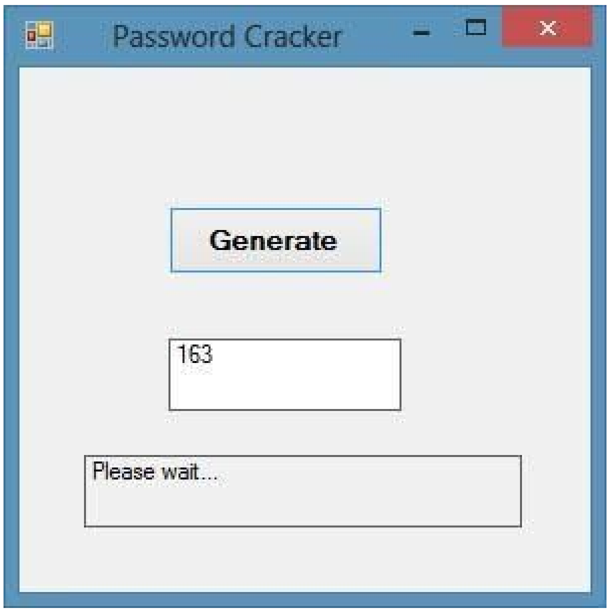
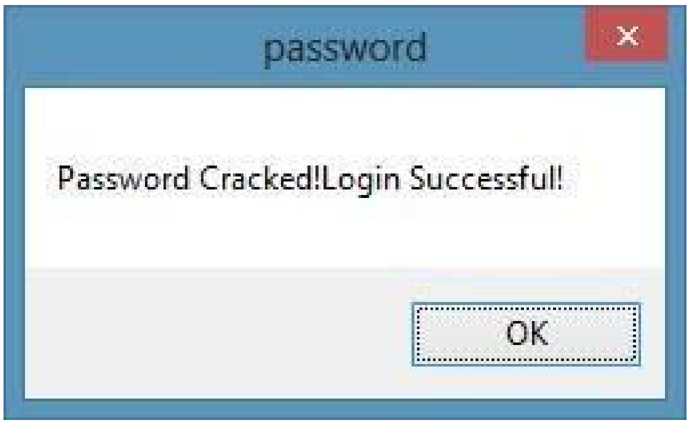


Figure 13.2: Passwords generating phase



Password Cracked!Login Successful!

Figure 13.3: Login Phase

Summary

- In section 13.1, you have learned about the concept of a sub procedure
- In section 13.2, you have learned how to write code for a sub procedure

Chapter 14 Creating Functions

- ❖ Understand the concept of Function
- ❖ Learn how to create user-defined functions

A function is like a sub procedure in the sense that both are called by the main procedure to fulfil specific tasks. However, there is one difference; a function returns a value whereas a sub procedure does not. There are two types of functions, the built-in functions (or internal functions) and the functions created by the programmers, or simply called user-defined functions.

14.1 Creating User-Defined Functions

To create a user defined function, type the function procedure directly into the code window as follows:

```
Public Function functionName (Argument As dataType,..........) As
dataType
```

or

```
Private Function functionName (Argument As dataType,..........) As
dataType
```

The keyword Public indicates that the function is applicable to the whole project whereas the keyword Private indicates that the function is only applicable to a specific module or procedure. Argument is a parameter that can pass a value back to the function. You can include as many arguments as you like.

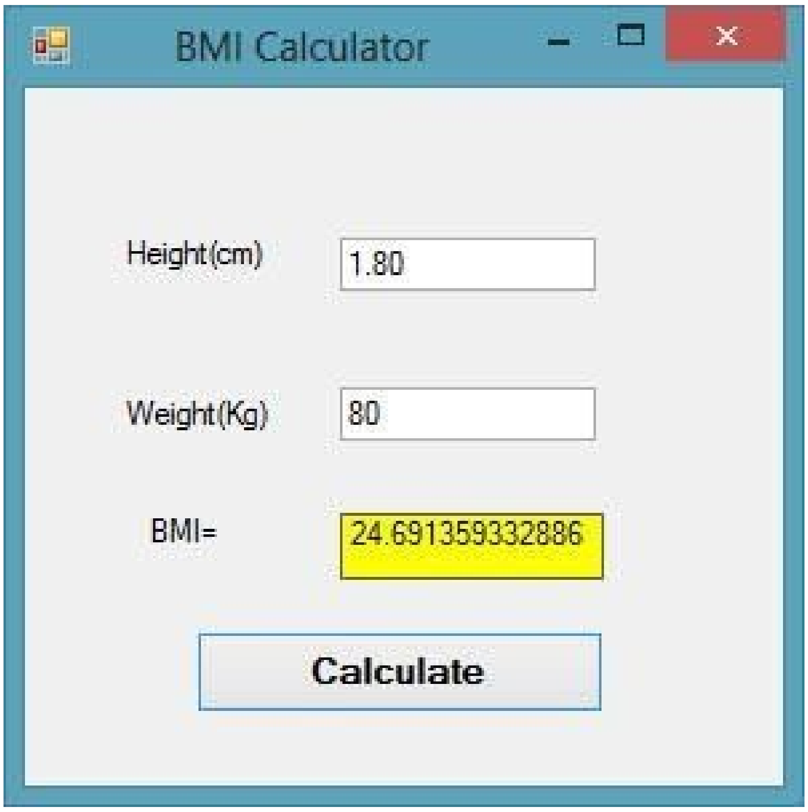
Example 14.1: BMI Calculator

This BMI calculator is a program that calculates the body mass index, or BMI of a person based on the body weight in kilogram and the body height in meter. BMI is calculated using the formula weight/(height $)^2$, where weight is measured in kg and height in meter. If you only know your weight and height in lb. and feet, then you need to convert them to the metric system. If your BMI is more than 30, you are considered obese

The Code

```
Private Function BMI(Height As Single, weight As Single) As Double
     BMI = weight / Height ^ 2
End Function

Private Sub BtnCal_Click(sender As Object, e As EventArgs) Handles
BtnCal.Click
Dim h As Single, w As Single
h = Val(TextBox1.Text)
w = Val(TextBox2.Text)
LblBMI.Text = BMI(h, w)
End Sub
```

The output is shown in Figure 14.1


BMI Calculator

Height(cm) 1.80

Weight(Kg) 80

BMI= 24.691359332886

Figure 14.1

Example 14.2 Future Value Calculator

The concept of future value is related to time value of money. For example, if you deposit your money in a bank account for a specific period, you will earn an interest based on the compound interest rate computed periodically. This profit is added to the principal if you continue to keep the money in the bank. Interest for the following period is now computed based on the initial principal plus the interest (the amount which becomes your new principal). Subsequent interests are computed in the same way.

For example, let us say you deposited \$1000 in a bank and the bank is paying you 5% compound interest annually. After the first year, you will earn an interest of \$1000×0.05=\$50 . Your new principal will be

\$1000+\$1000×0.05=\$1000(1+0.05)=\$1000(1.05)=\$1050.

After the second year, your new principal is \$1000(1.05) x1.05=\$1000(1.05)2 =\$1102.50. This new principal is called the future value.

Following the above calculation, the future value after n years will be

`FV = PV * (1 + i / 100)^n`

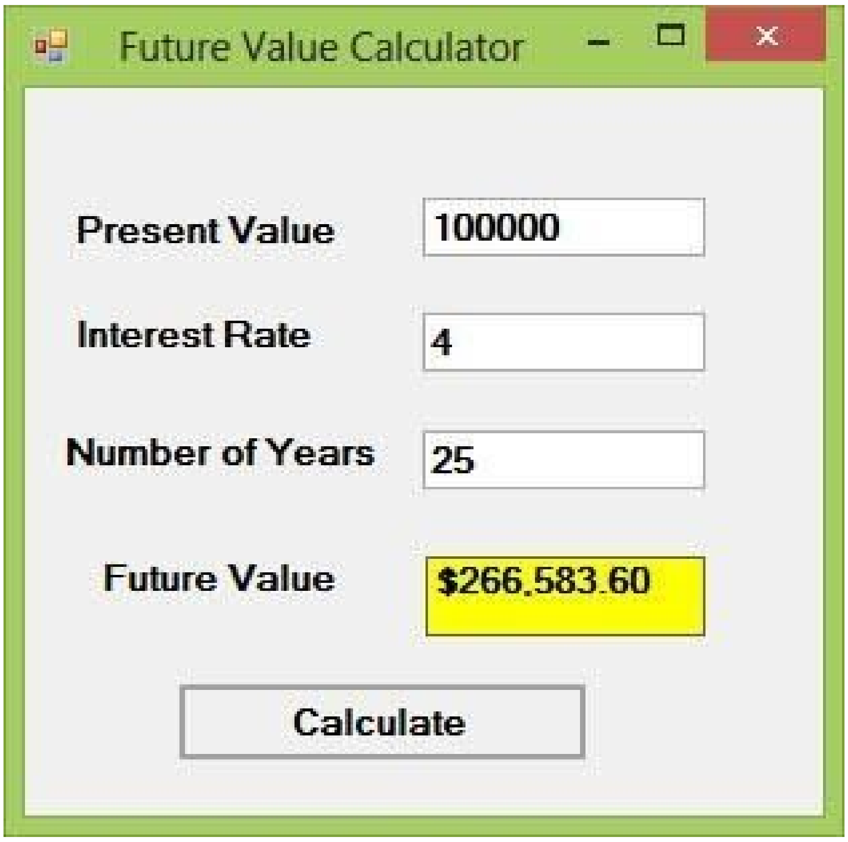
Where PV represents the present value, FV represents the future value, i is the interest rate and n is the number of periods (Normally months or years).

The Code

```
Private Function FV(pv As Single, i As Single, n As Integer) As Double
FV = pv * (1 + i / 100) ^ n
End Function
Private Sub BtnCal_Click(sender As Object, e As EventArgs) Handles
BtnCal.Click
Dim FutureVal As Single
Dim PresentVal As Single
Dim interest As Single
Dim period As Integer
PresentVal = TxtPV.Text
interest = TxtInt.Text
```

```
period = TxtN.Text
FutureVal = FV(PresentVal, interest, period)
LblFV.Text = Format(FutureVal, "$#,##0.00")
End Sub
```

The Output is shown in Figure 14.2



Future Value Calculator

Present Value 100000

Interest Rate 4

Number of Years 25

Future Value $266,583.60

Figure 14.2

14.2 Passing Arguments by Value and by Reference

Functions can be called by value or by reference. By default, the arguments in the function are passed by reference. If arguments are passed by reference, the original data will be modified and no longer preserved. On the one hand, if arguments are passed by value, the original data will be preserved. The keyword to pass arguments by reference is `ByRef` and the keyword to pass arguments by value is `ByVal`.

For example,

```
Private Function FV(ByVal pv As Single, ByRef i As Single, n As Integer)
As Double
```

The function FV receives pv by value, i by reference and n by reference. Notice that although ByRef is not used to pass n, by default it is passed by reference.

Example 14.3 ByRef and ByVal

In this example, we created two functions that compute the square root of a number , the first uses the keyword ByRef and the second uses the keyword ByVal.

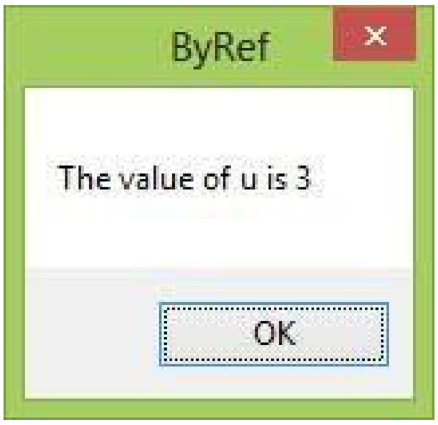
The Code

```
Private Function sqroot(ByRef x As Single) As Double
x = x ^ 0.5
sqroot = x
End Function

Private Function sqroot1(ByVal y As Single) As Double
y = y ^ 0.5
sqroot1 = y
End Function
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
Dim u As Single
u = 9
MsgBox(3 * sqroot(u), , "ByRef")
MsgBox("Value of u is " & u, , "ByRef")
End Sub
Private Sub Button2_Click(sender As Object, e As EventArgs) Handles
Button2.Click
Dim u As Single
u = 9
MsgBox(3 * sqroot1(u), , "ByVal")
MsgBox("Value of u is " & u, , "ByVal")
```

```
End Sub
```

The Outputs

Case 1: Passing argument using ByRef . The output is shown in Figure 14.3.

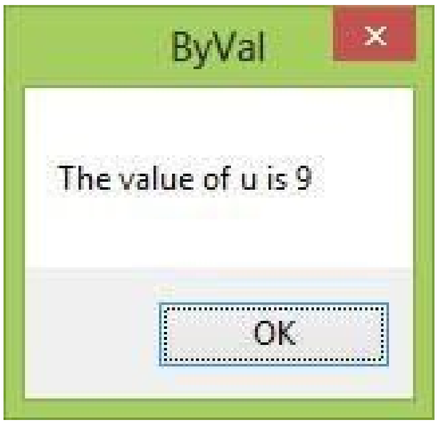


The value of u is 3

Figure 14.3

Notice that the value of u has been changed to 3

Case 2: Passing argument using ByVal(as shown in Figure 14.4)



The value of u is 9

Figure 14.4

Notice that the value of u remains unchanged.

Summary

- In section 14.1, you have learned how to create a user-defined function
- In section 14.2, you have learned about the difference between passing arguments using ByVal and ByRef

Chapter 15 Mathematical Functions

- ❖ Learn how to use various mathematical functions

In previous chapters, we have learned how to write code that performs mathematical operations using standard mathematical operators. However, for more complex mathematical calculations, we need to use the built-in math functions in Visual Basic 2017.

15.1 The Abs Function

The Abs function returns the absolute value of a given number. The syntax is

```
Math.Abs(number)
```

* The Math keyword here indicates that the Abs function belong to the Math class. However, not all mathematical functions belong to the Math class.

Example 15.1 Compute Absolute Number

In this example, we shall add a textbox control for the user to input his or her number and a label control to display the absolute value of the number. We need to use the Val function to convert text to numeric value. Rename the text box as TxtNum and the label as LblAbs.

The Code

```
Private Sub BtnComp_Click(sender As Object, e As EventArgs) Handles
BtnComp.Click
LblAbs.Text = Math.Abs(Val(TxtNum.Text))
End Sub
```

The output is shown in Figure 15.1

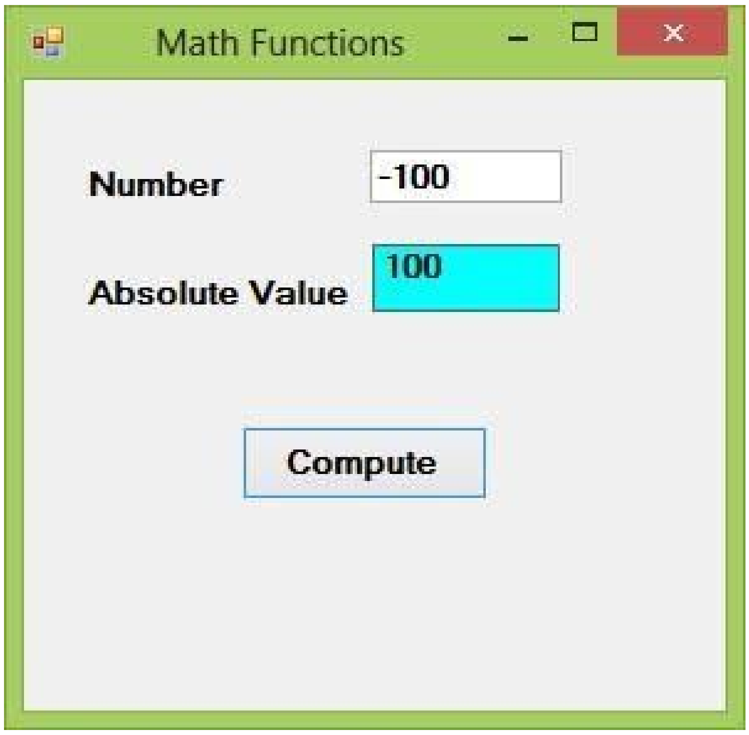


Figure 15.1

15.2 The Exp function

The Exp function returns the exponential value of a given number. For example,

Exp(1)=e=2.71828182

The syntax is

```
Math.Exp(number)
```

Example 15.2 Compute Exponential Value

In this example, we shall add a textbox control for the user to input his or her number and a label control to display the exponential value of the number. Rename the text box as TxtNum and the label as LblAbs.

The Code

```
Private Sub BtnComp_Click(sender As Object, e As EventArgs) Handles
BtnComp.Click
LblExp.Text = Math.Exp(Val(TxtNum.Text))
End Sub
```

The Output is shown in Figure 15.2

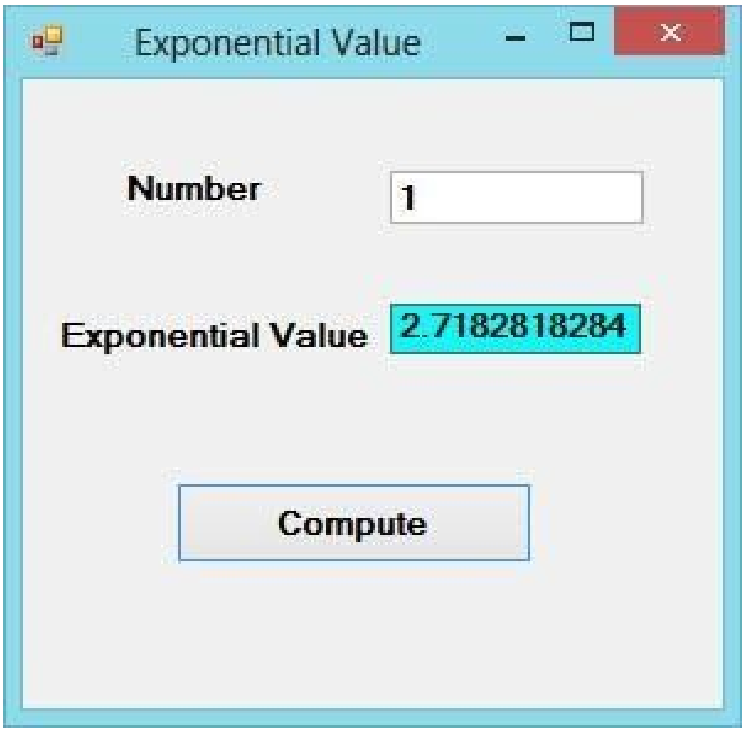


Figure 15.2

15.3 The Fix Function

The Fix function truncates the decimal part of a positive number and returns the largest integer smaller than the number. However, when the number is negative, it returns the smallest integer larger than the number. Fix does not belong to the Math class therefore we do not use the Math keyword.

Example 15.3 Truncate Decimals using Fix

```
Private Sub BtnComp_Click(sender As Object, e As EventArgs) Handles
BtnComp.Click
LblFixNum1.Text = Fix(Val(TxtPosNum.Text))
LblFixNum2.Text = Fix(Val(TxtNegNum.Text))
End Sub
```

The Output is shown in Figure 15.3

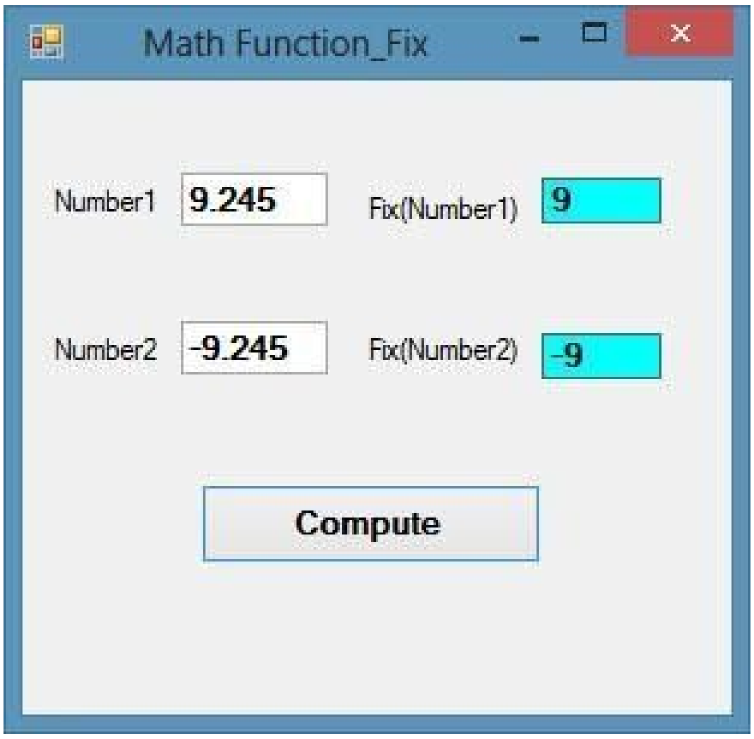


Figure 15.3

15.4 The Int Function

The Int is a function that converts a number into an integer by truncating its decimal part and the resulting integer is the largest integer that is smaller than the number. For example

`Int(2.4)=2, Int(6.9)=6 , Int(-5.7)=-6, Int(-99.8)=-100`

15.5 The Log Function

The Log function is the function that returns the natural logarithm of a number.

Example 15.4 Calculate Logarithm of a Number

```
Private Sub BtnComp_Click(sender As Object, e As EventArgs) Handles
BtnComp.Click
LblLog.Text = Math.Log(Val(TxtNum.Text))
End Sub
```

The Output is shown in Figure 15.4

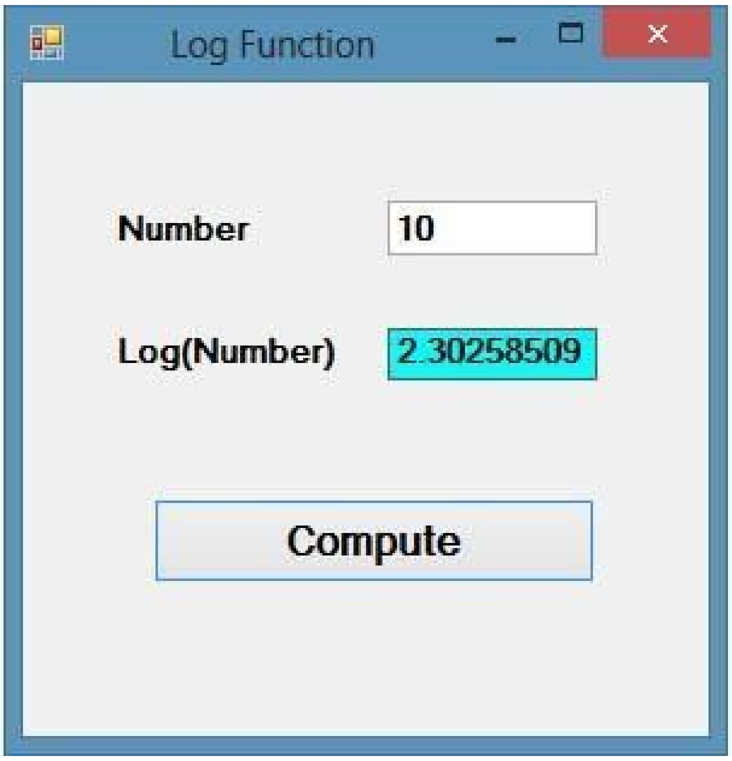


Figure 15.4

15.6 The Rnd( ) Function

We use the `Rnd` function to write code that involves chance and probability. The Rnd function returns a random value between 0 and 1. To make program behaves like a virtual dice, we need to convert the random numbers to integers using the formula `Int(Rnd*6)+1`.

Example 15.5 Generate Random Numbers

```
Private Sub BtnGen_Click(sender As Object, e As EventArgs) Handles
BtnGen.Click
LblRnd.Text = Int(VBMath.Rnd() * 6) + 1
End Sub
```

Notice that the **`Rnd()`** function belongs to the **`VBMath`** class in Visual Basic 2017. This is different from Visual Basic 2012, where you can omit the `VBMath` keyword.

In this example, **`Int(Rnd*6)`** will generate a random integer between 0 and 5 because the function Int truncates the decimal part of the random number and returns an integer. After adding 1, you will get a random number between 1 and 6 every time you click the command button. For example, let say the random number generated is 0.98, after multiplying it by 6, it becomes 5.88, and using the integer function Int(5.88) will convert the number to 5; and after adding 1 you will get 6.

The Output is shown in Figure 15.5

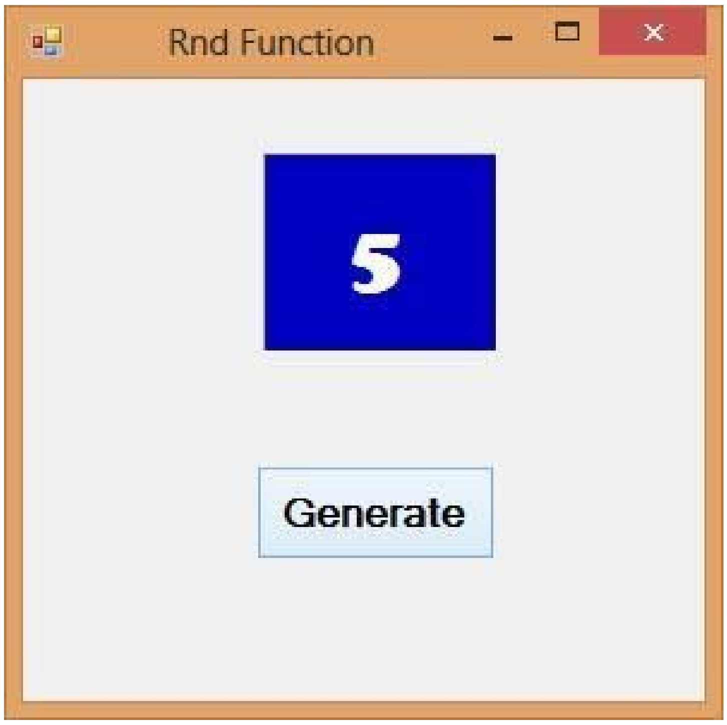


Figure 15.5

15.7 The Round Function

The Round function is the function that rounds up a number to a specific number of decimal places. The Format is Round (n, m) which means to round a number n to m decimal places. For example, **`Math.Round (7.2567, 2) =7.26`**

Example 15.6 Rounding a Number

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
Label1.Text = Math.Round(Val(TextBox1.Text), 2)
End Sub
```

The Output is shown in Figure 15.6

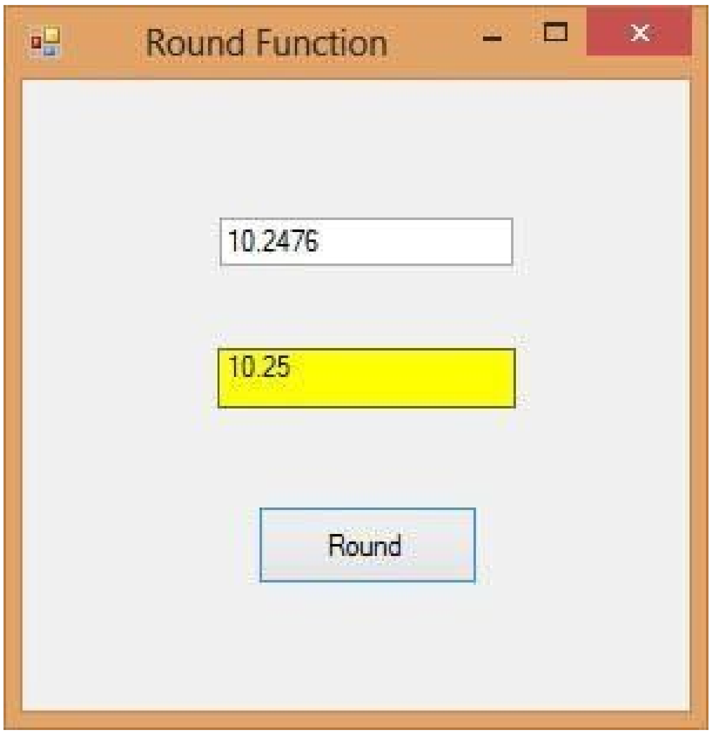


Figure 15.6

Summary

- In section 15.1, you have learned how to use the Abs function
- In section 15.2, you have learned how to use the Exp function
- In section 15.3, you have learned how to use the Fix function
- In section 15.4, you have learned how to use the Int function
- In section 15.5, you have learned how to use the Log function
- In section 15.6, you have learned how to use the Rnd function
- In section 15.7, you have learned how to use the Round function

Chapter 16 The Format Function

- ❖ Learn how to use the Format function

16.1 Format Function for Numbers

There are two types of Format functions for numbers; one of them is the built-in or predefined format while another one can be defined by the user.

16.1.1 Built-in Format function for Numbers

The syntax of the built-in Format function is

```
Format(n, "style argument")
```

Where n is a number.

The list of style arguments shown in Table 16.1.

Table 16.1

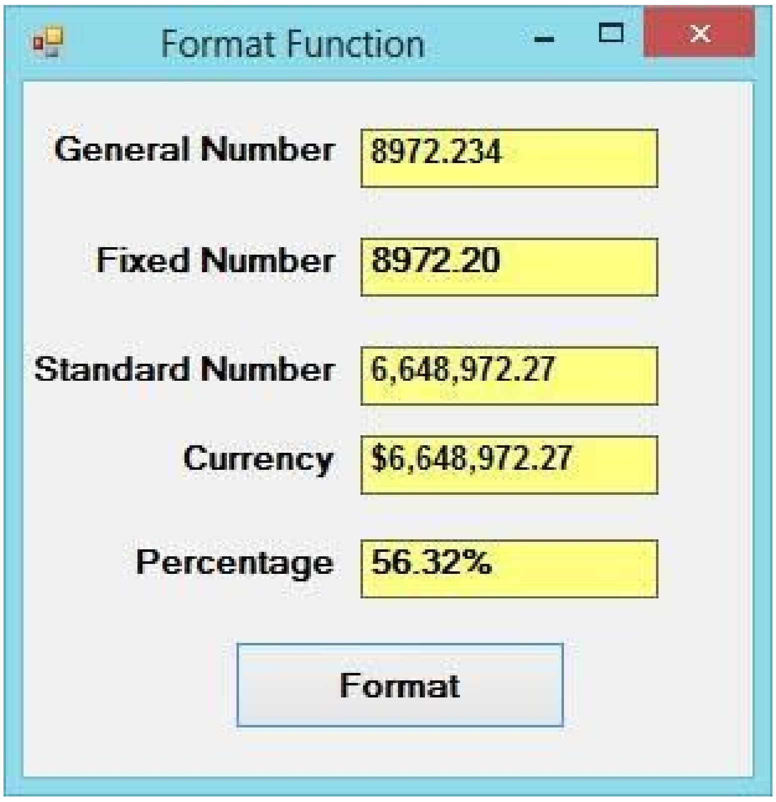
Style argument	Explanation	Example
General Number	To display the number without having separators between thousands.	Format(8972.234, “General Number”)=8972.234
Fixed	To display the number without having separators between thousands and rounds it up to two decimal places.	Format(8972.2, “Fixed”)=8972.23
Standard	To display the number with separators or separators between thousands and rounds it up to two decimal places.	Format(6648972.265, “Standard”)= 6,648,972.27
Currency	To display the number with the dollar sign in front, has separators between thousands as well as rounding it up to two decimal places.	Format(6648972.265, “Currency”)= $6,648,972.27
Percent	Converts the number to the percentage form and	Format(0.56324, “Percent”)=56.32 %

	displays a % sign and rounds it up to two decimal places.	

Example 16.1 Formatting Numbers

```
Private Sub BtnFormat_Click(sender As Object, e As EventArgs) Handles
BtnFormat.Click
Label1.Text = Format(8972.234, "General Number")
Label2.Text = Format(8972.2, "Fixed")
Label3.Text = Format(6648972.265, "Standard")
Label4.Text = Format(6648972.265, "Currency")
Label5.Text = Format(0.56324, "Percent")
End Sub
```

The output is displayed in Figure 16.1



General Number	8972.234
Fixed Number	8972.20
Standard Number	6,648,972.27
Currency	$6,648,972.27
Percentage	56.32%

Figure 16.1

16.1.2 User-Defined Format

The syntax of the user-defined Format function is

```
Format(n, "user's format")
```

Although it is known as user-defined format, we still need to follow specific formatting styles. Examples of user-defined formatting style are listed in Table 16.2

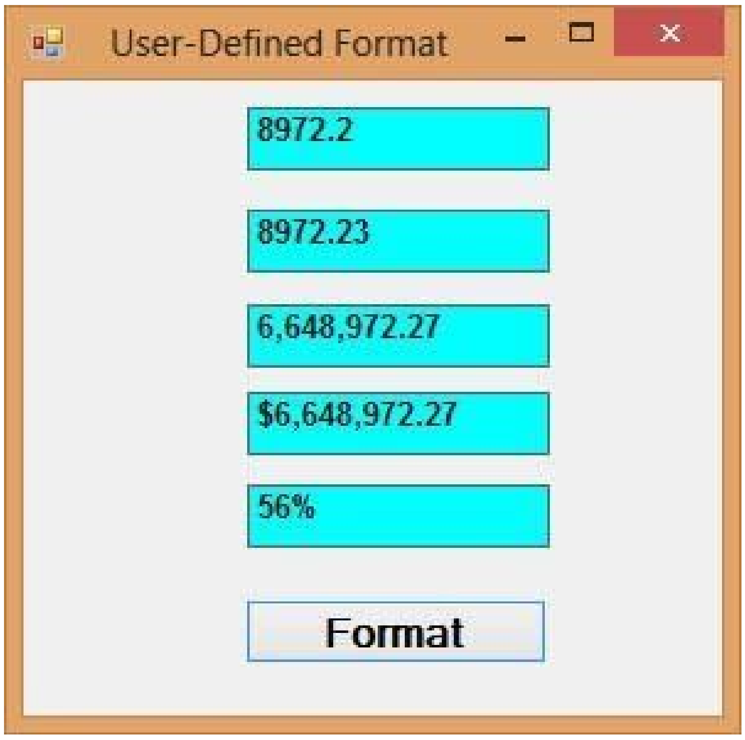
Table 16.2

Format	Description	Output
Format(781234.576,”0")	Rounds to whole number without separators between thousands	781235
Format(781234.576,”0.0")	Rounds to 1 decimal place without separators between thousands	781234.6
Format(781234.576,”0.00")	Rounds to 2 decimal place without separators between thousands	781234.58
Format(781234.576,”#,##0.00")	Rounds to 2 decimal place with separators between thousands	781,234.58
Format(781234.576,”$#,##0.00")	Displays dollar sign and Rounds to 2 decimal place with separators between thousands	$781,234.58
Format(0.576,”0%”)	Converts to percentage without decimal place	58%
Format(0.5768,”0%”)	Converts to percentage form with two decimal places	57.68%

Example 16.2 User-Defined Formats

```
Private Sub BtnFormat_Click(sender As Object, e As EventArgs) Handles
BtnFormat.Click
Label1.Text = Format(8972.234, "0.0")
Label2.Text = Format(8972.2345, "0.00")
Label3.Text = Format(6648972.265, "#,##0.00")
Label4.Text = Format(6648972.265, "$#,##0.00")
Label5.Text = Format(0.56324, "0%")
End Sub
```

The Output is shown in Figure 16.2



```
8972.2
8972.23
6,648,972.27
$6,648,972.27
56%
```

Figure 16.2

16.2 Formatting Date and Time

There are two types of Format functions for Date and time one of them is the built-in or predefined format while another one can be defined by the user.

16.2.1 Formatting Date and time using predefined formats

We can format date and time using predefined formats or user-defined formats. The predefined formats of date and time are shown in Table 16.3

Table 16.3

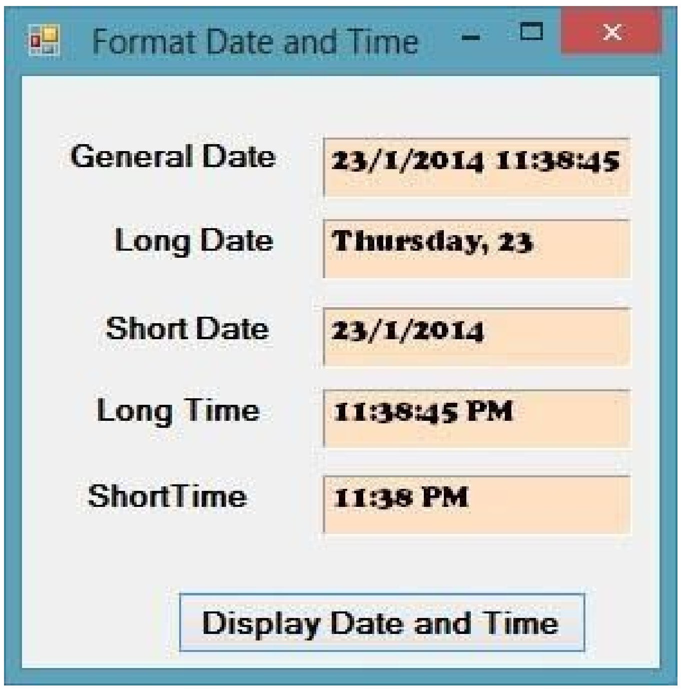
Format	Description
Format(Now, “General Date”)	Displays current date and time

Format(Now, “Long Date”)	Displays current date in long format
Format (Now, “Short date”)	Displays current date in short format
Format (Now, “Long Time”)	Displays current time in long format.
Format (Now, “Short Time”)	Displays current time in short format.

Example 16.3 Formating Date and Time

```
Private Sub BtnDisplay_Click(sender As Object, e As EventArgs) Handles
BtnDisplay.Click
Label1.Text = Format(Now, "General Date")
Label2.Text = Format(Now, "Long Date")
Label3.Text = Format(Now, "short Date")
Label4.Text = Format(Now, "Long Time")
Label5.Text = Format(Now, "Short Time")
End Sub
```

The Output is shown in Figure 16.3



```
General Date   23/1/2014 11:38:45
Long Date      Thursday, 23
Short Date     23/1/2014
Long Time      11:38:45 PM
ShortTime      11:38 PM
```

Figure 16.3

You can display dates and time in real-time using a timer and set its property Enabled to true and interval 100. The code is as follows:

```
Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles
Timer1.Tick
Label1.Text = Format(Now, "General Date")
Label2.Text = Format(Now, "Long Date")
Label3.Text = Format(Now, "short Date")
Label4.Text = Format(Now, "Long Time")
Label5.Text = Format(Now, "Short Time")
End Sub
```

16.2.2 Formatting Date and time using user-defined formats

Besides using the predefined formats, you can also use the user-defined formatting functions. The syntax of a user-defined format for date and time is

```
  Format(expression,style)
```

Table 16.4

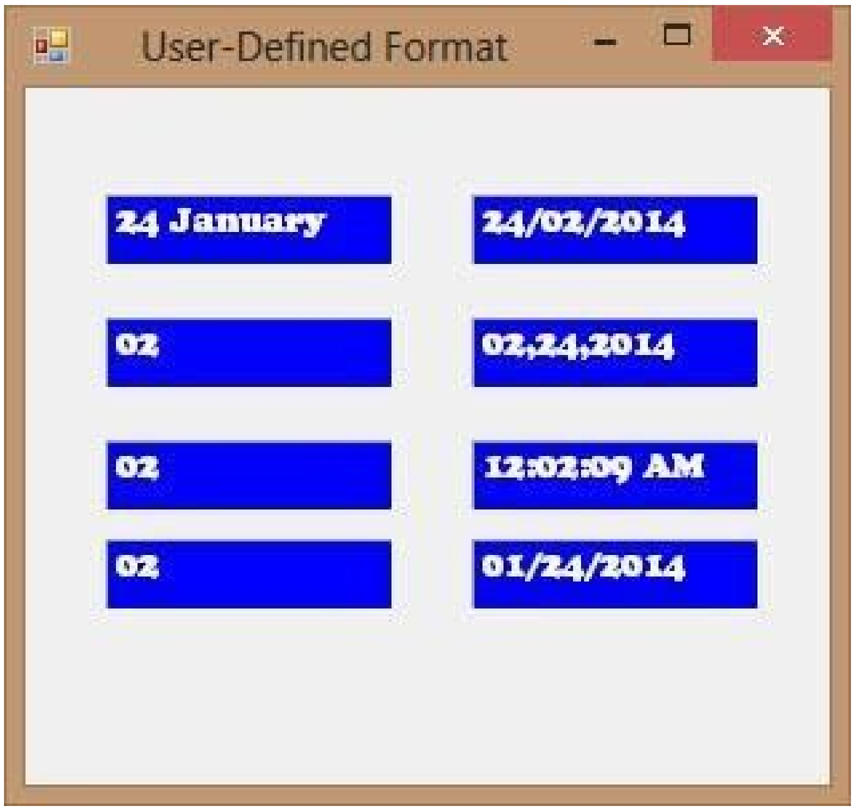
Format	Description
Format (Now, “m”)	Displays current month and date
Format (Now, “mm”)	Displays current month in double digits.
Format (Now, “mmm”)	Displays abbreviated name of the current month
Format (Now, “mmmm”)	Displays full name of the current month.
Format (Now, “dd/mm/yyyy”)	Displays current date in the day/month/year format.
Format (Now, “mmm,d,yyyy”)	Displays current date in the Month, Day, Year Format
Format (Now, “h:mm:ss tt”)	Displays current time in hour:minute:second format and show am/pm
Format (Now, “MM/dd/yyyy h:mm:ss)	Displays current date and time in hour:minute:second format

Example 16.4 Formatting Date and Time

```
Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles
Timer1.Tick
Label1.Text = Format(Now, "m")
```

```
Label2.Text = Format(Now, "mm")
Label3.Text = Format(Now, "mmm")
Label4.Text = Format(Now, "mmmm")
Label5.Text = Format(Now, "dd/mm/yyyy")
Label6.Text = Format(Now, "mmm,d,yyyy")
Label7.Text = Format(Now, "h:mm:ss tt")
Label8.Text = Format(Now, "MM/dd/yyyy h:mm:ss tt")
End Sub
```

The output is shown in Figure 16.4



```
24 January      24/02/2014
02              02,24,2014
02              12:02:09 AM
02              01/24/2014
```

Figure 16.4

Summary

- In section 16.1(a), you have learned how to use the built-in format function
- In section 16.1(b), you have learned how to use the user-defined format function
- In section 16.2(a), you have learned how to format date and time using predefined formats

- In section 16.2(b), you have learned how to format date and time using user-defined formats

Chapter 17 Using Checkbox and Radio Button

- ❖ Learn how to use check box
- ❖ Learn how to use radio button

17.1 Check Box

A Check box allows the user to select one or more items by checking the checkbox or check boxes concerned. In Visual Basic 2017, you may create a shopping cart where the user can click on checkboxes that correspond to the items they intend to buy, and the total payment can be computed at the same time.

Example 17.1: Shopping Cart

In this example, we add a few labels, two buttons and six checkboxes. We declare the price of each item using the Const keyword. If a checkbox is being ticked, its state is True else its state is False. To calculate the total amount of purchase, we use the mathematical operator +=. For example, `sum+=BN` is sum=`sum+BN`. Finally, we use the `ToString` method to display the amount in currency.

The Code

```
Private Sub BtnCal_Click(sender As Object, e As EventArgs) Handles
BtnCal.Click
Const LX As Integer = 100
Const BN As Integer = 500
Const SD As Integer = 200
Const HD As Integer = 80
Const HM As Integer = 300
Const AM As Integer = 150
Dim sum As Integer
If CheckBox1.Checked = True Then
sum += LX
End If
If CheckBox2.Checked = True Then
sum += BN
End If
If CheckBox3.Checked = True Then
```

```
sum += SD
End If
If CheckBox4.Checked = True Then
sum += HD
End If
If CheckBox5.Checked = True Then
sum += HM
End If
If CheckBox6.Checked = True Then
sum += AM
End If
LblTotal.Text = sum.ToString("c")
End Sub
Private Sub BtnReset_Click(sender As Object, e As EventArgs) Handles
BtnReset.Click
CheckBox1.Checked = False
CheckBox2.Checked = False
CheckBox3.Checked = False
CheckBox4.Checked = False
CheckBox5.Checked = False
CheckBox6.Checked = False
End Sub
```

The Runtime Interface is shown in Figure 17.1

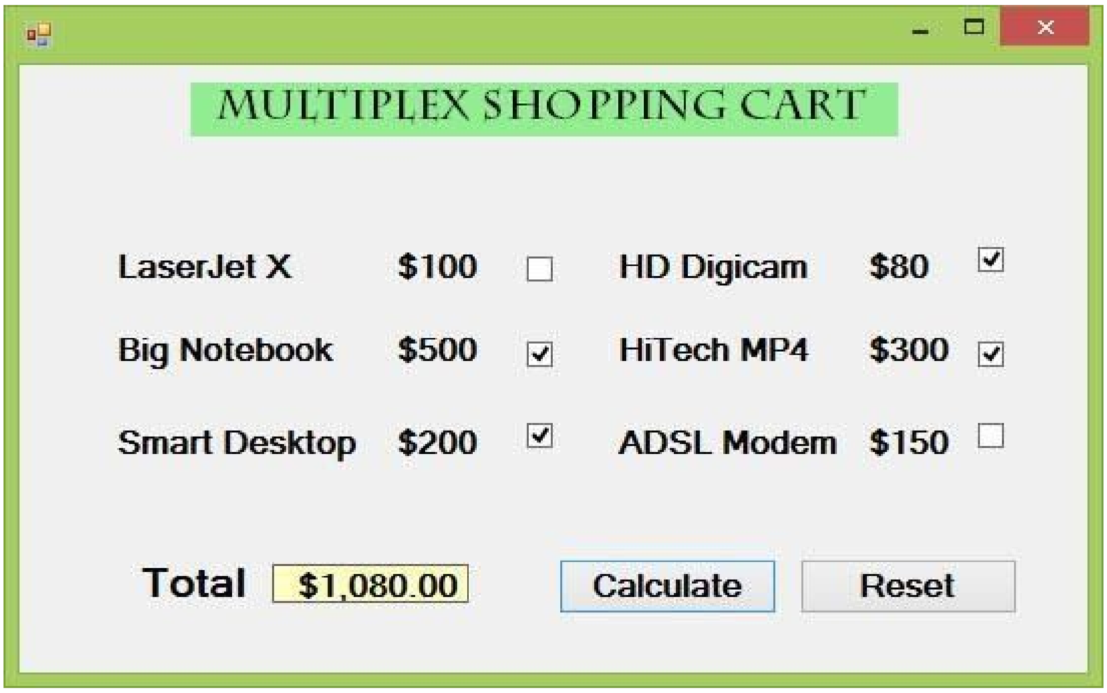


Figure 17.1 Shopping Cart

Example 17.2 Another Shopping Cart

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
Const large As Integer = 10.0
Const medium As Integer = 8
Const small As Integer = 5
Dim sum As Integer
If CheckBox1.Checked = True Then
sum += large
End If

If CheckBox2.Checked = True Then
sum += medium
End If
If CheckBox3.Checked = True Then
sum += small
End If
Label5.Text = sum.ToString("c")
End Sub
```

Example 17.3 Formatting Text

In this example, the text on the label can be formatting using the three check boxes that represent bold, italic and underline.

The Code

```
Private Sub ChkBold_CheckedChanged(sender As Object, e As EventArgs)
Handles ChkBold.CheckedChanged
If ChkBold.Checked Then
LblDisplay.Font = New Font(LblDisplay.Font, LblDisplay.Font.Style Or
FontStyle.Bold)
Else
LblDisplay.Font = New Font(LblDisplay.Font, LblDisplay.Font.Style And Not
FontStyle.Bold)
End If
End Sub

Private Sub ChkItalic_CheckedChanged(sender As Object, e As EventArgs)
Handles ChkItalic.CheckedChanged

If ChkItalic.Checked Then
LblDisplay.Font = New Font(LblDisplay.Font, LblDisplay.Font.Style Or
FontStyle.Italic)
Else
LblDisplay.Font = New Font(LblDisplay.Font, LblDisplay.Font.Style And Not
FontStyle.Italic)
End If
End Sub

Private Sub ChkUnder_CheckedChanged(sender As Object, e As EventArgs)
Handles ChkUnder.CheckedChanged
If ChkUnder.Checked Then
LblDisplay.Font = New Font(LblDisplay.Font, LblDisplay.Font.Style Or
FontStyle.Underline)
Else
LblDisplay.Font = New Font(LblDisplay.Font, LblDisplay.Font.Style And Not
FontStyle.Underline)
End If
End Sub
```

* The above program uses the CheckedChanged event to respond to the user selection by checking a checkbox; it is like the click event. The statement

```
LblDisplay.Font = New Font(LblDisplay.Font, LblDisplay.Font.Style Or
FontStyle.Italic)
```

will retain the original font type but change it to italic font style.

```
LblDisplay.Font = New Font(LblDisplay.Font, LblDisplay.Font.Style And Not
FontStyle.Italic)
```

will also retain the original font type but change it to regular font style.

The Output interface is shown in Figure 17.2

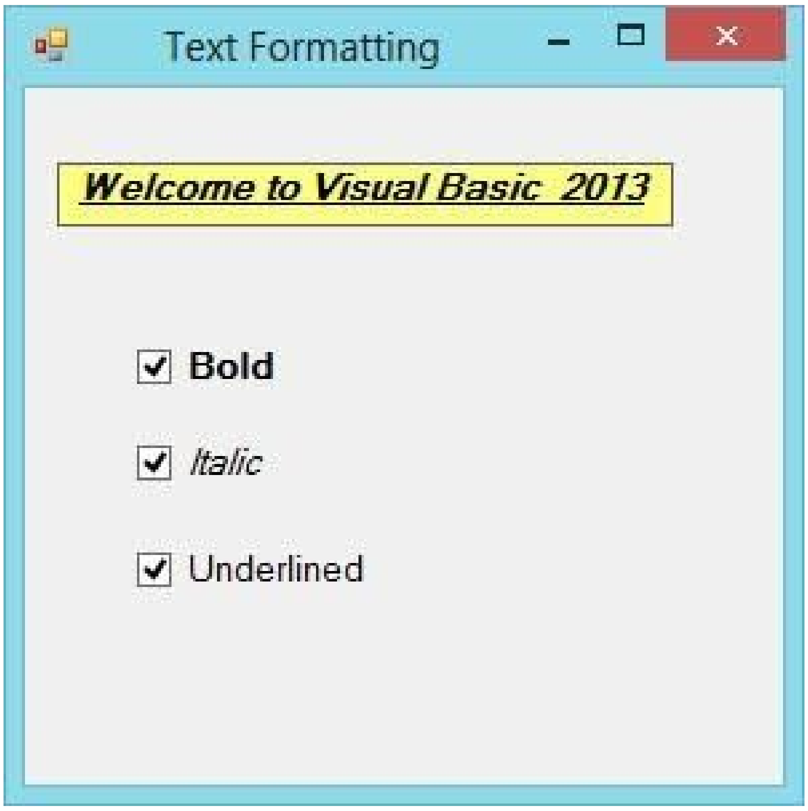


Figure 17.2

17.2 Radio Button

Radio buttons operate differently from the check boxes. While the check boxes work independently and allow the user to select one or more items, radio buttons are mutually exclusive, which means the user can only choose one item only out of several choices.

Example 17.4 Shopping Cart

In this example, the user can only choose one T-shirt color. To design the interface, add three radio buttons and name them as RadioRed, RadioGreen and RadioYellow respectively. Besides that, add a button to confirm the chosen color and a label control to display the chosen color. Name the button as BtnConfirm and the label as LblDisplay. We use the **If…Then…Else** decisions making structure to construct the program. The state of the radio button is indicated by its checked property.

The code

```
Private Sub BtnConfirm_Click(sender As Object, e As EventArgs) Handles
BtnConfirm.Click
Dim Tcolor As String
If RadioRed.Checked  Then
Tcolor = "Red Color"
LblDisplay.ForeColor = Color.Red
ElseIf RadioGreen.Checked  Then
Tcolor = "Green Color"
LblDisplay.ForeColor = Color.Green
Else
Tcolor = "Yellow Color"
LblDisplay.ForeColor = Color.Yellow
End If
LblDisplay.Text = Tcolor
End Sub
```

The Runtime Interface is shown in Figure 17.3

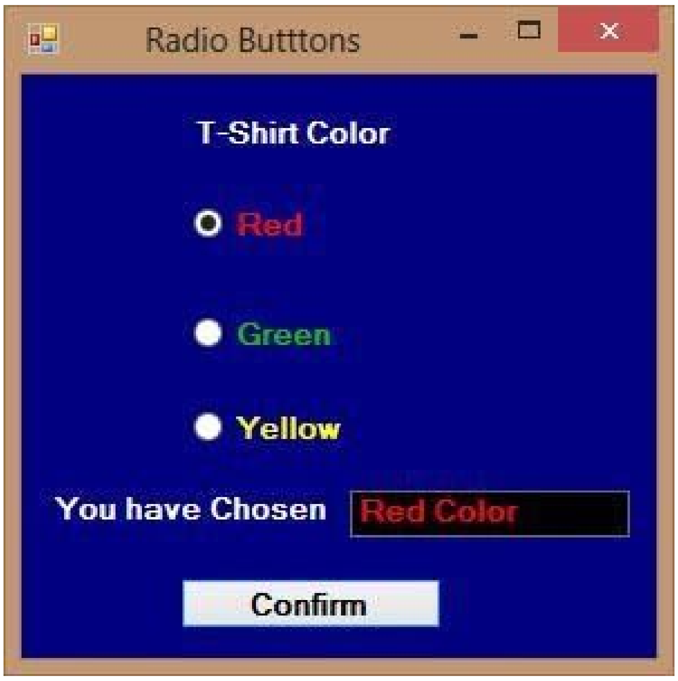


Figure 17.3

Example 17.2 Using Groupbox

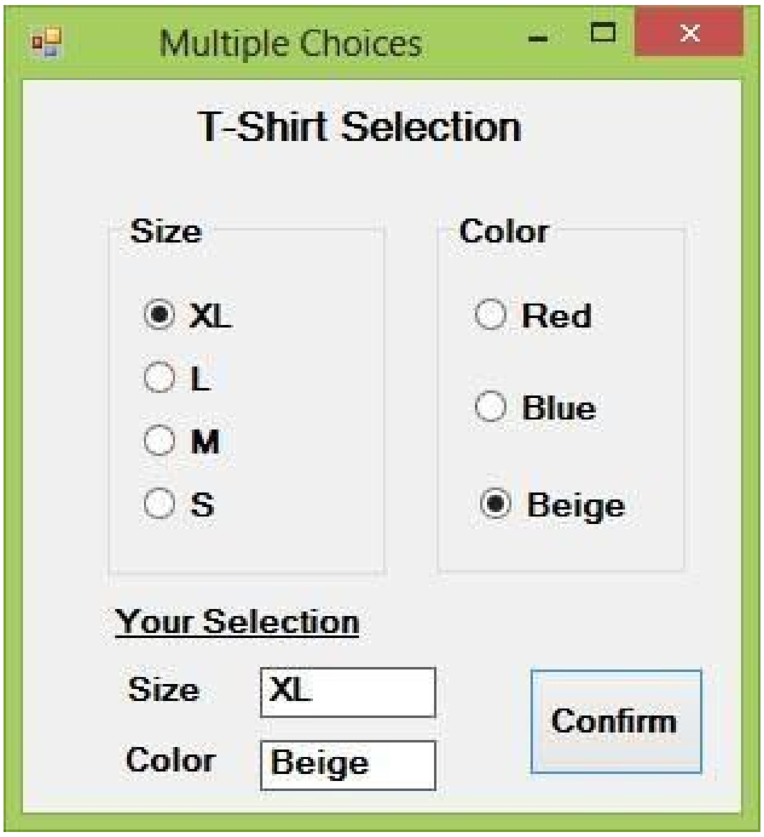
Although the user may only select one item at a time, he may make more than one selection if those items belong to different categories. For example, the user wishes to choose T-shirt size and color, he needs to select one color and one size, which mean one selection in each category. In this case, we need to group the radio buttons together according to the categories. This is easily achieved in Visual Basic 2017 using the Groupbox control under the container's categories.

In the Visual Basic 2017 IDE, after inserting the Groupbox from the toolbox into the form, you can proceed to insert the radio buttons into the Groupbox. Only the radio buttons inside the Groupbox are mutually exclusive, they are not mutually exclusive with the radio buttons outside the Groupbox. In this example, the user can select one color and one size of the T-shirt. To design the interface, insert two group boxes. In the first group box, add four radio buttons and name them as RadioXL, RadioL, RadioM and Radio S respectively. In the second group box, add three radio buttons and name them RadioRed, RadioBlue and RadioBeige respectively. Besides that, insert two label control to display the chosen size and color, name them LblSize and LblColor respectively. Finally, add a button and name it as BtnConfirm. In the code, we shall declare two variables, TSize to indicate the T-shirt size and TColor ro indicate the T-shirt color.

The Code

```
Private Sub BtnConfirm_Click(sender As Object, e As EventArgs) Handles
BtnConfirm.Click
Dim TSize, TColor As String
If RadioXL.Checked Then
TSize = "XL"
ElseIf RadioL.Checked Then
TSize = "L"
ElseIf RadioM.Checked Then
TSize = "M"
Else : TSize = "S"
End If
If RadioRed.Checked Then
TColor = "Red"
ElseIf RadioBlue.Checked Then
TColor = "Blue"
Else : TColor = "Beige"
End If

LblSize.Text = TSize
Lblcolor.Text = TColor

End Sub
```

The Runtime Interface is shown in Figure 17.5



Size: XL
Color: Beige

Figure 17.5

Summary

- In section 17.1, you have learned how to use the check box
- In section 17.2, you have learned how to use the radio button

Chapter 18 Errors Handling

- ❖ Understand the concept of errors handling
- ❖ Learn how to handle errors using On Error Goto Syntax
- ❖ Learn how to handle errors using Try….Catch….End Try Structure

18.1 Introduction

Errors often occur due to incorrect input from the user. For example, the user might make the mistake of attempting to enter text (string) to a box that is designed to handle only numeric values such as the weight of a person, the computer will not be able to perform arithmetic calculation for text therefore will create an error. These errors are known as synchronous errors.

Therefore, a good programmer should be more alert to the parts of program that could trigger errors and should write errors handling code to help the user in managing the errors. Writing errors handling code is a good practice for programmers, so do not try to finish a program fast by omitting the errors handling code. However, there should not be too many errors handling code in the program as it creates problems for the programmer to maintain and troubleshoot the program later. Visual Basic 2017 has improved a lot in its built-in errors handling capabilities compared to Visual Basic 6. For example, when the user attempts to divide a number by zero, Visual Basic 2017 will not return an error message but gives the 'infinity' as the answer.

18.2 Using On Error GoTo Syntax

Visual Basic 2017 still supports the VB6 errors handling syntax that is the On Error GoTo program_label structure. Although it has a more advanced error handling method, we shall deal with that later. We shall now learn how to write errors handling code in Visual Basic 2017. The syntax for errors handling is

```
On Error GoTo program_label
```

* program_label is the section of code that is designed by the programmer to handle the error committed by the user. Once an error is detected, the program will jump to the program_label section for error handling.

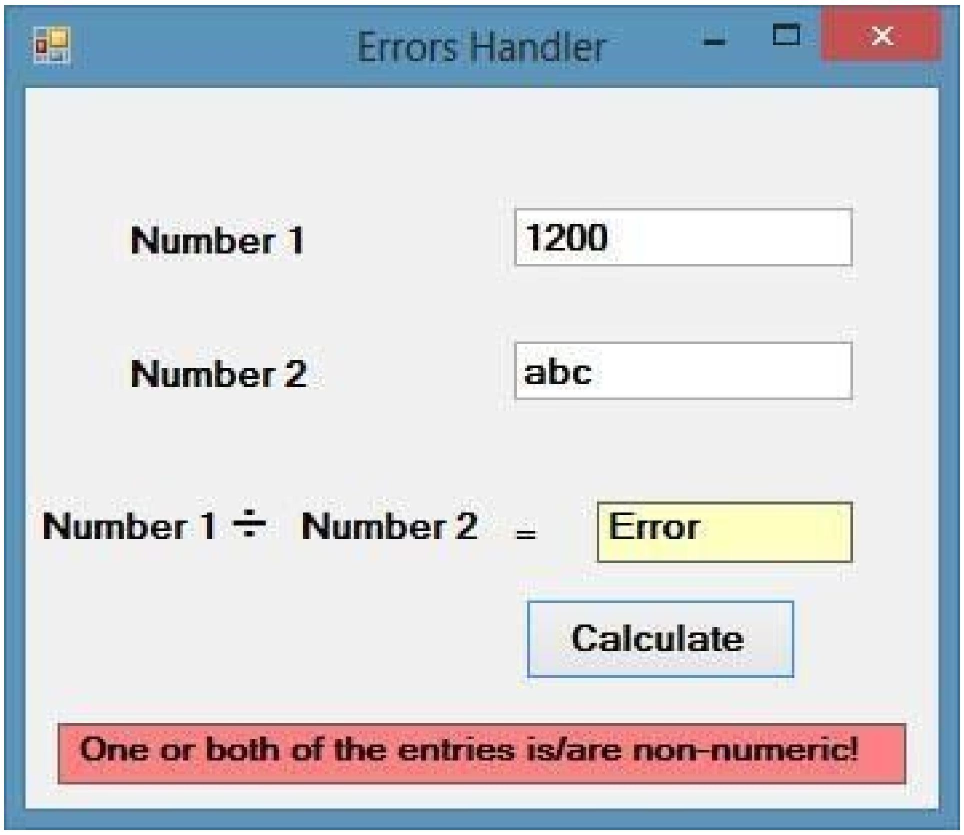
Example 18.1 Division Errors

In this example, we will deal with the error of entering non-numeric data into the text boxes that supposed to hold numeric values. The program_label here is error_handler. when the user enters a non-numeric values into the text boxes, the error message will display the text ”One or both of the entries is/are non-numeric!”. If no error occurs, it will display the correct answer. Try it out yourself.

The Code

```
Private Sub BtnCal_Click(sender As Object, e As EventArgs) Handles
BtnCal.Click
Lbl_ErrMsg.Visible = False
Dim firstNum, secondNum As Double
On Error GoTo error_handler
firstNum = TxtNum1.Text
secondNum = TxtNum2.Text
Lbl_Answer.Text = firstNum / secondNum
Exit Sub  'To prevent error handling even the inputs are valid
error_handler:
Lbl_Answer.Text = "Error"
Lbl_ErrMsg.Visible = True
Lbl_ErrMsg.Text = "One or both of the entries is/are non-numeric! Try
again!"
End Sub
```

The runtime interface is shown in Figure 18.1



1200

abc

Error

One or both of the entries is/are non-numeric!

Figure 18.1

*Please note that division by zero in Visual Basic 2017 no longer gives an error message, but it displays the answer as Infinity.

18.3 Errors Handling using Try...Catch...End Try Structure

Visual Basic 2017 has adopted a new approach in handling errors, or rather exceptions handling. It is supposed to be more efficient than the old On Error Goto method, where it can handle various types of errors within the Try…Catch…End Try structure.

The structure looks like this

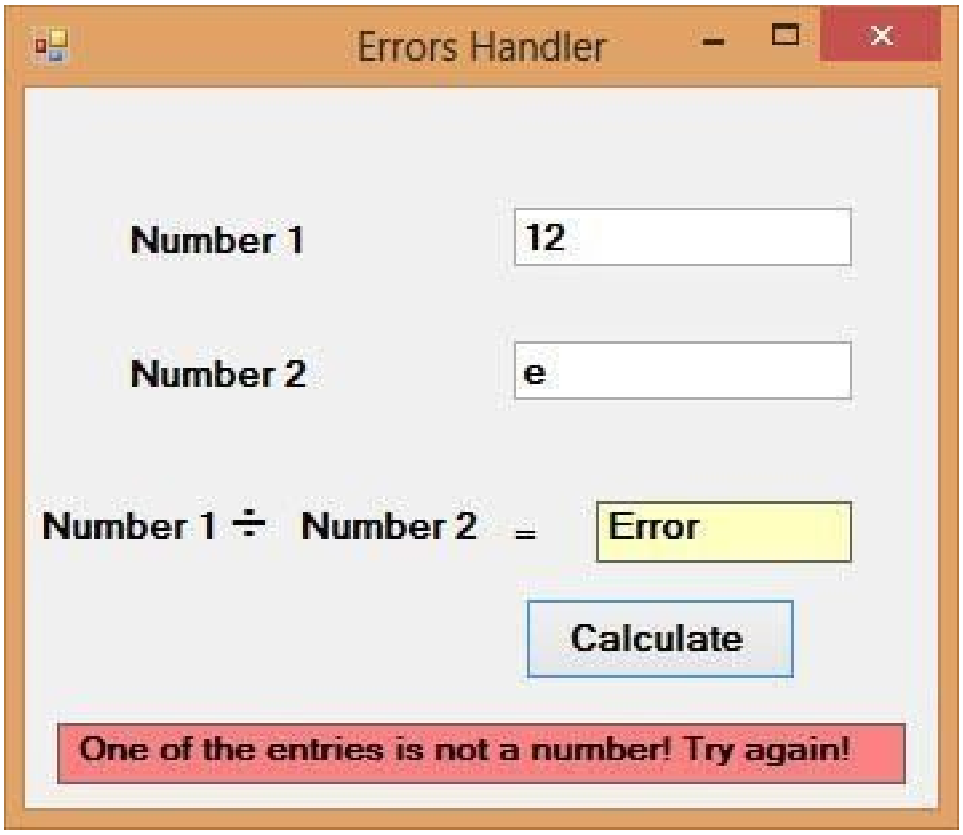
```
Try
statements
Catch exception_variable As Exception
```

```
statements to deal with exceptions
End Try
```

Example 18.2 Data Type Mismatch Error

```
Private Sub BtnCal_Click(sender As Object, e As EventArgs) Handles
BtnCal.Click
Lbl_ErrMsg.Visible = False
Dim firstNum, secondNum, answer As Double
Try
firstNum = TxtNum1.Text
secondNum = TxtNum2.Text
answer = firstNum / secondNum
Lbl_Answer.Text = answer
Catch ex As Exception
Lbl_Answer.Text = "Error"
Lbl_ErrMsg.Visible = True
Lbl_ErrMsg.Text = " One of the entries is not a number! Try again!"
End Try
End Sub
```

The runtime interface is shown in Figure 18.2



One of the entries is not a number! Try again!

Figure 18.2

Summary

- In section 18.1, you have learned the concept of errors handling
- In section 18.2, you have learned how to handle errors using On Error Goto Syntax
- In section 18.3, you have learned how to handle errors using Try…Catch…End Try Structure

Chapter 19 Object Oriented Programming

- ❖ Understand the concepts of object-oriented programming

19.1 Concepts of Object-Oriented Programming

In order for a programming language to qualify as an object-oriented programming language, it must have three core technologies namely encapsulation, inheritance and polymorphism. These three terms are explained below:

19.1.1 Encapsulation

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.

19.1.2 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.

19.1.3 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.

19.2 Creating Class

Visual Basic 2017 allows users to write programs that break down into modules. These modules 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 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 Visual Basic 2017, 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
```

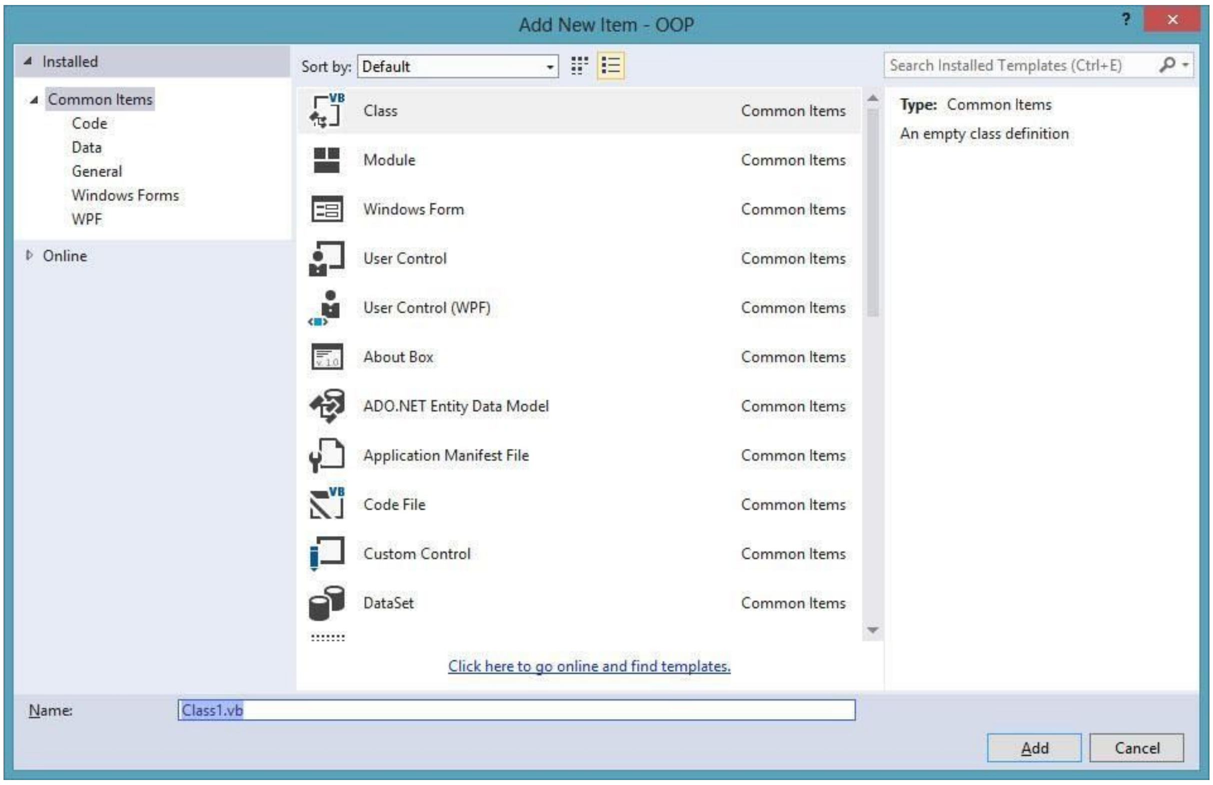
Another Example:

```
Public Class Car
'Data Members
Private Brand As String
Private Model As String
Private Year Made As String
Private Capacity As Integer
'Methods
Overridable Sub ShowInfo()
MessageBox.Show(Brand)
MessageBox.Show(Model)
MessageBox.Show(Year Made)
MessageBox.Show(Capacity)
End Sub
End Class
```

Example 19.1 BMI Calculator

Let us look at one example on how to create a class. The following example shows you how to create a class that can calculate your BMI (Body Mass Index).

To create class, start Visual Basic 2017 as usual and choose Windows Applications. In the Visual Basic 2017 IDE, click on Project on the menu bar and select Add Class, the Add New Item dialog appears, as shown in Figure 19.1



Type: Common Items
An empty class definition

Name: Class1.vb

Figure 19.1

The default class Class1.vb will appear as a new tab with a code window. Rename the class as MyClass.vb. Rename the form as MyFirstClass.vb.

Now, in the MyClass.vb window, create a new class MyClass1 and enter the following code

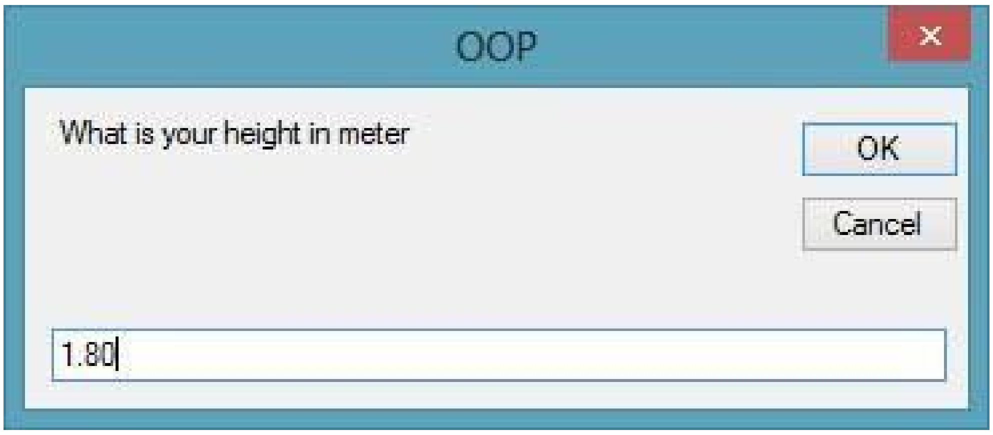
```
Public Class MyClass1
Public Function BMI(ByVal height As Single, ByVal weight As Single)
BMI = Format((weight) / (height ^ 2), "0.00")
End Function
End Class
```

Now you have created a class (an object) called MyClass1 with a method known as BMI.

To use the BMI class, insert a button into the form and click on the button to enter the following code:

```
Private Sub BtnBMI_Click(sender As Object, e As EventArgs) Handles
BtnBMI.Click
Dim MyObject As Object
Dim h, w As Single
MyObject = New MyClass1()
h = InputBox("What is your height in meter")
w = InputBox("What is your weight in kg")
MessageBox.Show(MyObject.BMI(h, w))
End Sub
```

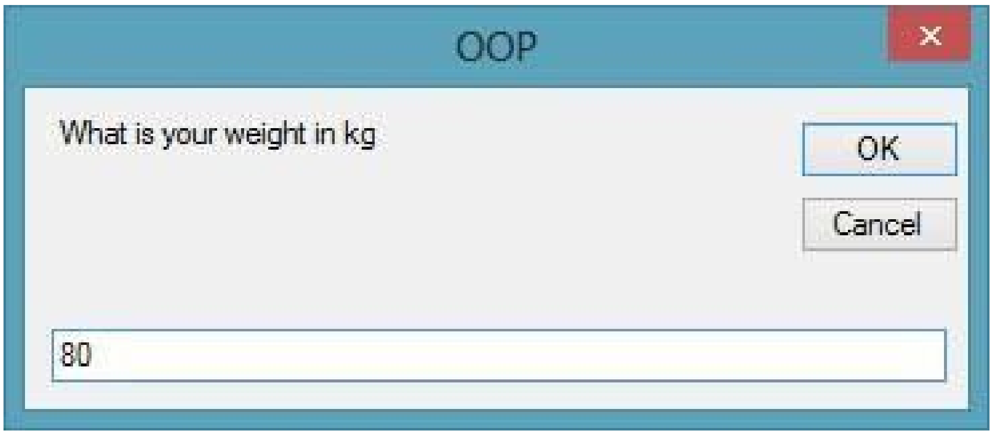
When you run this program and click the button, the user will be presented with two input boxes to enter his or her height and weight subsequently and the value of BMI will be shown in a pop-up message box, as shown in the figures below:


OOP

What is your height in meter

1.80

Figure 19.2


OOP

What is your weight in kg

80

Figure 19.3

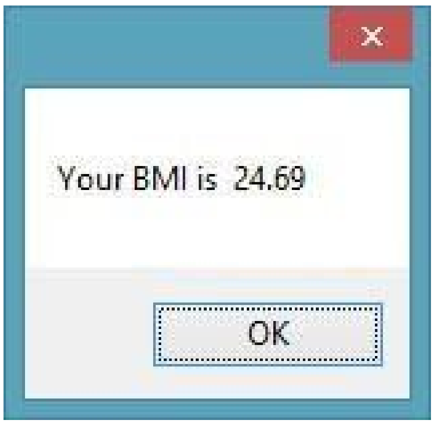

Your BMI is 24.69

Figure 19.4

Summary

- In section 19.1, you have learned the concepts of object-oriented programming
- In section 19.2, you have learned how to create a class

Chapter 20 Creating Graphics

- ❖ Understand the concepts of Graphics Creation
- ❖ Learn how to create the Graphics object
- ❖ Learn how to create the Pen object
- ❖ Learn how to draw a line
- ❖ Learn how to create a rectangle
- ❖ Learn how to customize lifestyle
- ❖ Learn how to draw an ellipse
- ❖ Learn how to draw a circle
- ❖ Learn how to draw text
- ❖ Learn how to draw a polygon
- ❖ Learn how to draw a pie
- ❖ Learn how to draw fill shapes with colors

20.1 Introduction

Creating graphics was easy in earlier versions of Visual Basic because they have built-in drawing tools. For example, In Visual Basic 6, the drawing tools are included in the toolbox where the programmer just need to drag the shape controls into the form to create rectangle, square, ellipse, circle and more. However, its simplicity has the shortcomings; you do not have many choices in creating customized graphics.

Since Visual Basic evolved into an object-oriented programming language under the VB.net framework, shape controls are no longer available. Now the programmer must write code to create various shapes and drawings. Even though the learning curve is steeper, the programmer can write powerful code to create all kinds of graphics. You can even design your own controls

Visual Basic 2017 offers various graphics capabilities that enable programmers to write code that can create all kinds of shapes and even fonts. In this chapter, you will learn how to write code to draw lines and shapes on the Visual Basic 2017 IDE.

20.2 Creating the Graphics Object

Before you can draw anything on a form, you need to create the Graphics object in Visual Basic 2017. A graphics object is created using the `CreateGraphics()` method. You can create a graphics object that draw to the form itself or a control.

To draw graphics on the default form, you can use the following statement:

```
Dim myGraphics As Graphics = me.CreateGraphics
```

To draw in a picture box, you can use the following statement:

```
Dim myGraphics As Graphics = PictureBox1.CreateGraphics
```

You can also use the text box as a drawing canvas, the statement is:

```
Dim myGraphics As Graphics = TextBox1.CreateGraphics
```

The Graphics object that is created does not draw anything on the screen until you call the methods of the Graphics object. In addition, you need to create the `Pen` object as the drawing tool. We shall examine the code that can create a pen in the following section.

20.3 Creating the Pen Object

A `Pen` can be created using the following code:

```
myPen = New Pen(Brushes.Color, LineWidth)
```

`myPen` is a `Pen` variable. You can use any variable name instead of `myPen`. The first argument of the pen object defines the color of the drawing line and the second argument defines the width of the drawing line. For example, the following code created a pen that can draw a dark magenta line and the width of the line is 10 pixels:

```
myPen = New Pen(Brushes.DarkMagenta, 10)
```

You can also create a `Pen` using the following statement:

```
Dim myPen As Pen

myPen = New Pen(Drawing.Color.Blue, 5)
```

Where the first argument defines the color and the second argument is the width of the drawing line. Having created the Graphics and the Pen objects, you are now ready to draw graphics on the screen.

20.4 Drawing a Line

The method to draw a straight line is `Drawline`. Let us illustrate how to draw a straight line with an example.

Example 20.1 Drawing a Straight Line

```
Private Sub BtnDraw_Click(sender As Object, e As EventArgs) Handles
BtnDraw.Click
Dim myGraphics As Graphics = Me.CreateGraphics
Dim myPen As Pen
myPen = New Pen(Brushes.DarkMagenta, 20)
myGraphics.DrawLine(myPen, 60, 180, 220, 50)
End Sub
```

The second line of the code creates the Graphics object and the third and fourth line create the `Pen` object. The fifth draw a line on the Form using the DrawLine method. The first argument use the Pen object created by you, the second argument and the third arguments define the coordinate the starting point of the line, the fourth and the last arguments define the ending coordinate of the line. The syntax of the `Drawline` method is

```
object.DrawLine(Pen, x1, y1, x2, y2)
```

In the example, the starting coordinate is (60,80) and the ending coordinate is (220,50). Figure 20.1 shows the line created by the program.

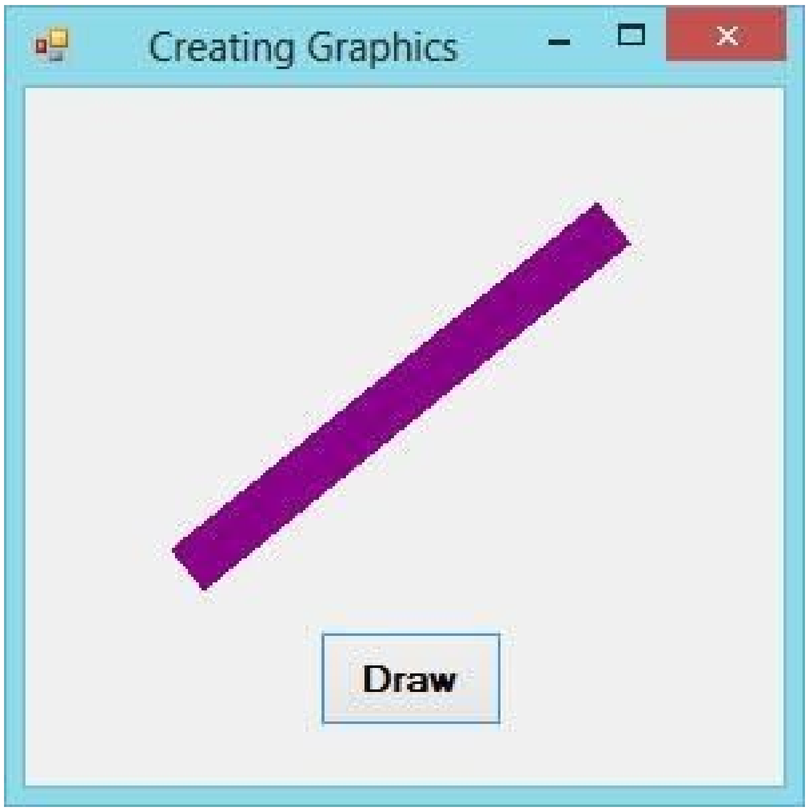


Figure 20.1

20.5 Drawing Lines that Connect Multiple Points

In section 20.4, we have learned to draw a straight line that connect two points. Now we shall learn how to draw straight lines that connect multiple points. The method is `Drawlines` and the syntax is

```
Graphics.DrawLines(Pen, Point())
```

Notice that the method to draw a straight line is `DrawLine` whereas the method to draw multiple lines is `Drawlines`. The points can be defined using the `Point()` array with the following syntax:

```
Point() = {point1, point2, point3, point4,............}
```

We need to declare the array using the `Dim` keyword, as follows:

```
Dim MyPoints As  Point() = {point1, point2, point3, point4,.....}
```

In addition, each point be declared using the `Dim` keyword, as follows:

```
Dim point1 as New Point (x1,y1)
Dim point2 as New Point (x2,y2)
Dim point3 as New Point (x3,y3)
```

Let us examine the following example.

Example 20.2 Drawing Lines that Connect Multiple Points

```
Private Sub BtnDrawLine_Click(sender As Object, e As EventArgs) Handles
BtnDrawLine.Click
Dim point1 As New Point(30, 30)
Dim point2 As New Point(70, 15)
Dim point3 As New Point(100, 5)
Dim point4 As New Point(200, 70)
Dim point5 As New Point(350, 90)
Dim point6 As New Point(300, 150)
Dim point7 As New Point(20, 200)
Dim myPoints As Point() = {point1, point2, point3, point4, point5,
point6, point7}
Dim myGraphics As Graphics = Me.CreateGraphics
Dim myPen As Pen
     myPen = New Pen(Brushes.OrangeRed, 2)
     myGraphics.DrawLines(myPen, myPoints)
End Sub
```

The output interface is as shown in Figure 20.2

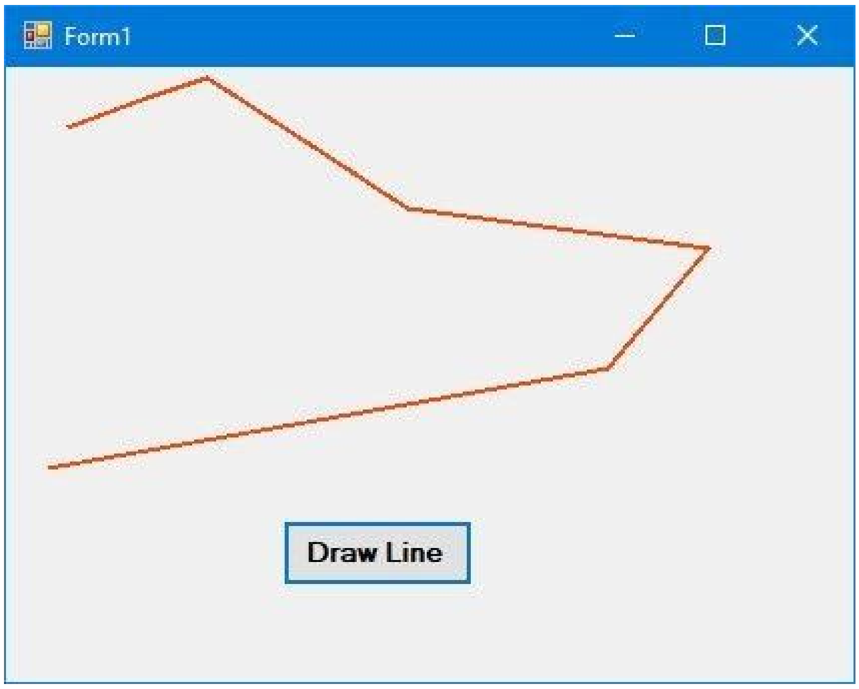


Figure 20.2

20.6 Drawing a curve that Connect Multiple Points

In this section, we shall learn how to draw a curve that connects multiple points. To draw a curve, we use the `DrawCurve()` method and the syntax is

```
Graphics.DrawCurve(Pen, Point())
```

The points can be defined using the `Point()` array with the following syntax:

```
Point() = {point1, point2, point3, point4,.............}
```

We need to declare the array using the Dim keyword, as follows:

```
Dim MyPoints as  Point() = {point1, point2, point3, point4,.....}
```

In addition, each point must be declared using the Dim keyword, as follows:

```
Dim point1 as New Point (x1,y1)
Dim point2 as New Point (x2,y2)
Dim point1 as New Point (x3,y3)
```

Example 20.3 Drawing a Curve that Connect Multiple Points

```
Private Sub BtnDrawCurve_Click(sender As Object, e As EventArgs) Handles
BtnDrawCurve.Click
```

```
Dim point1 As New Point(30, 30)
Dim point2 As New Point(70, 15)
Dim point3 As New Point(100, 5)
Dim point4 As New Point(200, 70)
Dim point5 As New Point(350, 90)
Dim point6 As New Point(300, 150)
Dim point7 As New Point(20, 200)
Dim myPoints As Point() = {point1, point2, point3, point4,  point5,
point6, point7}
Dim myGraphics As Graphics = Me.CreateGraphics
Dim myPen As Pen
        myPen = New Pen(Brushes.DarkMagenta, 2)
        myGraphics.DrawCurve(myPen, myPoints)
End Sub
```

The output interface is as shown in Figure 20.3

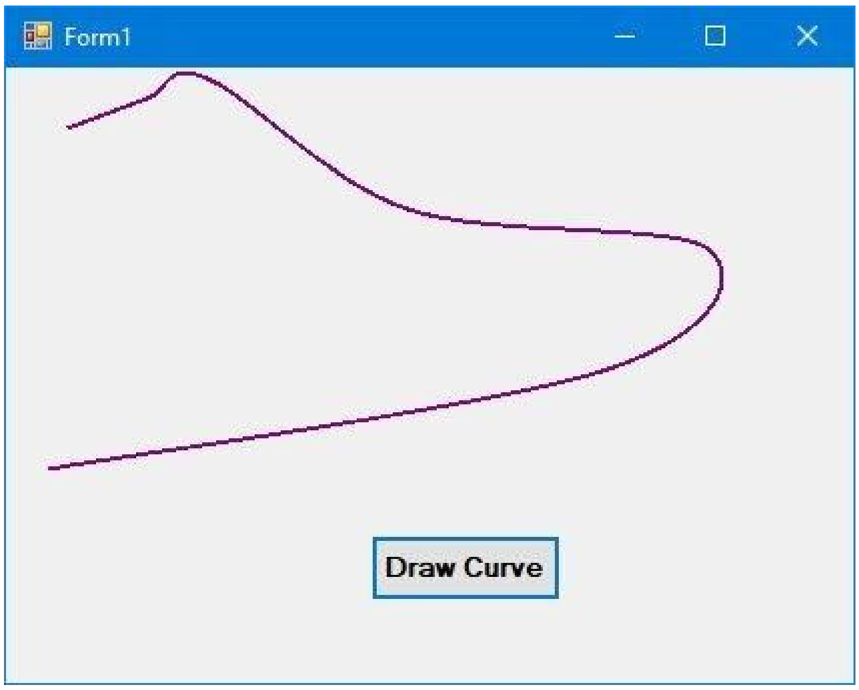


Figure 20.3

20.7 Drawing Quadratic Curve

In this section, we shall learn how to draw a quadratic curve. To do this, we need to adjust the coordinate system of the drawing surface. The default origin of the drawing surface of the object such as the form or the picture box is at the upper left corner. We can move its origin to another point using the **TranslateTransform** method of the Graphics class, the syntax is as follows:

```
MyGraphics.TranslateTransform(dx:=x1, dy:=y1)
```

where (x1,y1) is the new origin.

Example 20.4 Drawing a Quadratic Curve

In this example, we want to draw a quadratic graph for the following quadratic function:

$y=x^2-3x+1$

First, we insert a picture box as the drawing canvas. Set its size to 500, 300 which means its width=500 pixels and height=300 pixels. We wish to move the origin to the middle of x-axis and close to x-axis, so we set its origin using the following syntax:

```
myGraphics.TranslateTransform(dx:=250, dy:=295)
```

Next, we must declare a point array so that we can use the `For...Next` loop to generate points for the quadratic equation. After generating the points, we can use the `DrawCurve` method to draw the curve.

The equation x = i - 250 is to make sure the point start from -250 and end at 250. The equation y =300 - (x ^ 2 - 3 * x + 1) is to ensure the point start from the bottom instead of the top of the drawing canvas. We divide it by 200 to reduce the values of the y-coordinates so that it will not go out of bound.

The Code

```
Private p(500) As Point
Private Sub BtnDrawCurve_Click(sender As Object, e As EventArgs) Handles
BtnDrawCurve.Click
        Dim x, y, i As Double
        For i = 0 To 500
            x = i - 250
            y = (300 - (x ^ 2 - 3 * x + 1)) / 200
            p(i) = New Point(x, y)
        Next
        Dim myGraphics As Graphics = MyCanvas.CreateGraphics
        Dim myPen As Pen
        myGraphics.TranslateTransform(dx:=250, dy:=295)
        myPen = New Pen(Brushes.DarkMagenta, 2)
        myGraphics.DrawCurve(myPen, p)
        myGraphics.Dispose()
    End Sub
```

The output is as shown in Figure 20.4.

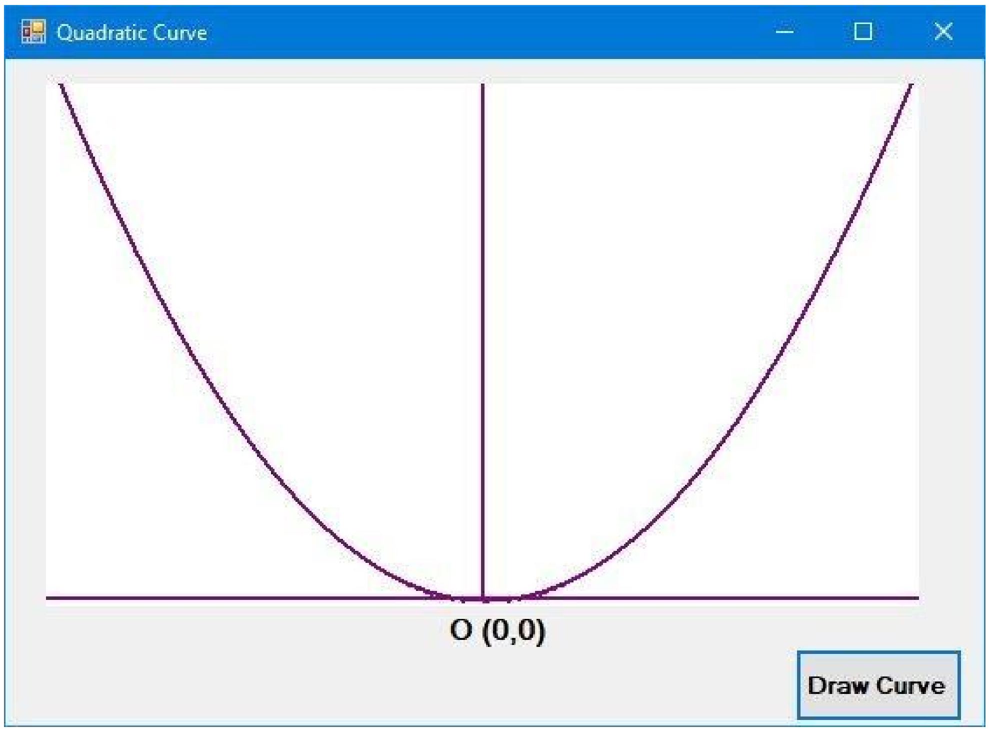


Figure 20.4

20.8 Drawing Sine Curve

The Sin function returns the sine value of an angle. We must convert the angle to radian as Visual Basic 2017 cannot deal with an angle in degree. The conversion is based on the following equation:

π radian= 180º

so 1º=π/180 radian

The issue is how to get the exact value of π? We can use π=3.14159 but it will not be accurate. To get the exact value of π, we use the arc sine function, i.e. is Asin.

Using the equation sin(π/2)=1, so Asin(1)=π/2, therefore, π=2Asin(1). Therefore, The syntax of the Sin function in Visual Basic 2017 is

```
Math.Sin(Angle in radian)
```

Example 20.5 Drawing a Sine Curve

In this example, we insert a picture box and fix its size as 600, 300 which means its width is 600 and its height is 300. Based as the same principle as in Example 20.3, we shift the origin to (300, 180) using the `TranslateTransform` method. The sine function in VB2017 is

```
y = -(Math.Sin((pi * x) / 180)) * 80
```

We use the negative to accommodate to the new coordinate system. We multiply it by 80 so that the value is big enough for the coordinate y to be visible on the drawing canvas.

The Code

```
Public Class Form1
Private p(600) As Point
Private Sub BtnDrawCurve_Click(sender As Object, e As EventArgs) Handles
BtnDrawCurve.Click
        Dim x, y, i As Double
        Dim pi As Single
        pi = 2 * Math.Asin(1)
        For i = 0 To 600
            x = i - 300
            y = -(Math.Sin((pi * x) / 180)) * 80
            p(i) = New Point(x, y)
        Next
        Dim myGraphics As Graphics = MyCanvas.CreateGraphics
        Dim myPen As Pen
        myGraphics.TranslateTransform(dx:=300, dy:=180)
        myPen = New Pen(Brushes.DarkMagenta, 2)
        myGraphics.DrawLine(myPen, 0, 100, 0, -200)
        myGraphics.DrawLine(myPen, -300, 0, 300, 0)
        myGraphics.DrawCurve(myPen, p)
        myGraphics.Dispose()

    End Sub
End Class
```

The output is shown in Figure 20.5

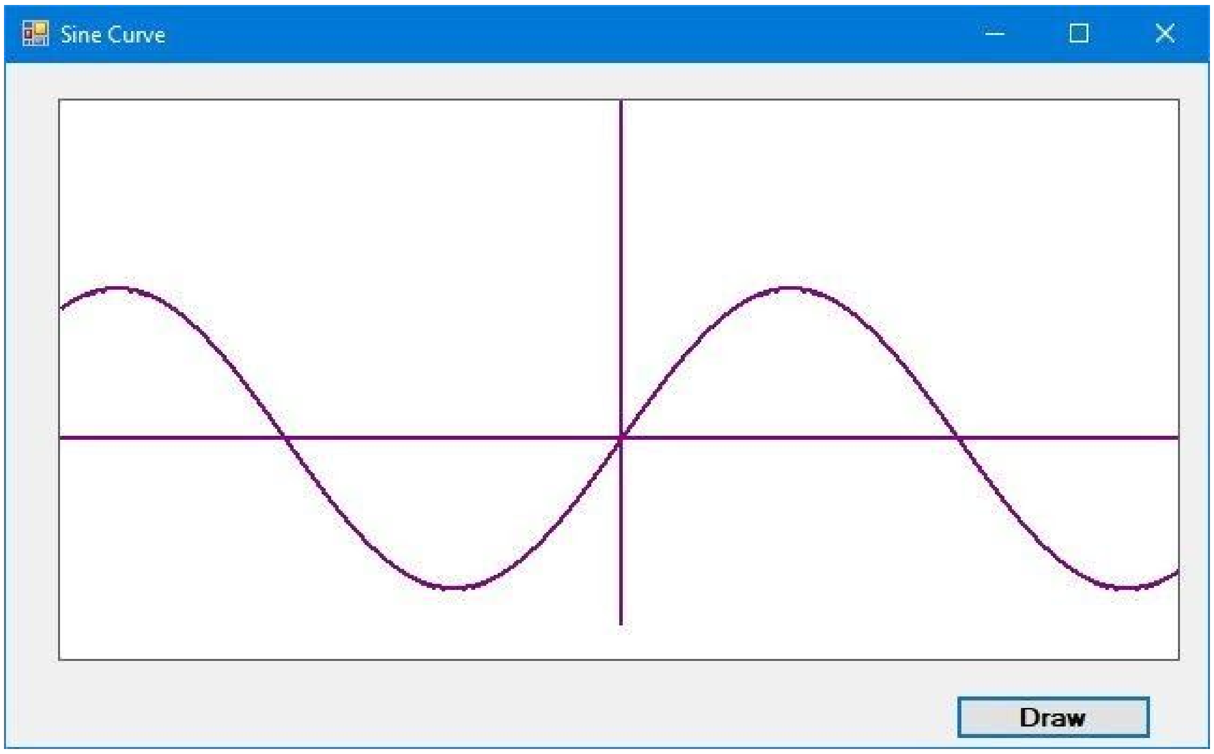


Figure 20.5

20.9 Drawing a Rectangle

To draw a rectangle on the default form in Visual Basic 2017, there are two ways:

(i)The first is to draw a rectangle directly using the **`DrawRectangle`** method by specifying its upper-left corner coordinates and it width and height. You also need to create a Graphics and a Pen object to handle the actual drawing. The method of the Graphics object to draw the rectangle is `DrawRectangle` .

The syntax is:

```
myGrapphics.DrawRectangle(myPen, X, Y, width, height)
```

Where **myGraphics** is the variable name of the **Graphics** object and myPen is the variable name of the Pen object created by you. You can use any valid and meaningful variable names. X, Y is the coordinate of the upper left corner of the rectangle while width and height are self-explanatory, i.e., the width and height of the rectangle.

The code is as follows:

```
Dim myPen As Pen
myPen = New Pen(Drawing.Color.Blue, 5)
Dim myGraphics As Graphics = Me.CreateGraphics
myGraphics.DrawRectangle(myPen, 0, 0, 100, 50)
```

(ii) The second way is to create a rectangle object first and then draw this triangle using the **DrawRectangle** method. The syntax is as shown below:

```
myGraphics.DrawRectangle(myPen,myRectangle)
```

where **myRectangle** is the rectangle object created by you, the user.

The code to create a rectangle object is as shown below:

```
Dim myRectangle As New Rectangle
myRect.X = 10
myRect.Y = 10
myRect.Width = 100
myRect.Height = 50
```

You can also create a rectangle object using a one-line code as follows:

```
Dim myRectangle As New Rectangle(X,Y,width, height)
```

The code to draw the above rectangle is

```
myGraphics.DrawRectangle(myPen, myRectangle)
```

20.10 Customizing Line Style of the Pen Object

The shape we draw so far are drawn with solid line, we can customize the line style of the Pen object so that we have dotted line, line consisting of dashes and more. For example, the syntax to draw with dotted line is shown below.

```
myPen.DashStyle=Drawing.Drawing2D.DashStyle.Dot
```

The last argument Dot specifies a line DashStyle value, a line that makes up of dots here. Other DashStyles values are `Dash`, `DashDot`, `DashDotDot` and `Solid`. The following code draws a rectangle with blue dotted line, as shown in Figure 20.6.

Example 20.6 Drawing a Rectangle with DashStyle

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.ClickDim myPen As Pen
myPen = New Pen(Drawing.Color.Blue, 5)
Dim myGraphics As Graphics = Me.CreateGraphics
myPen.DashStyle = Drawing.Drawing2D.DashStyle.Dot
myGraphics.DrawRectangle(myPen, 10, 10, 100, 50)
End Sub
```

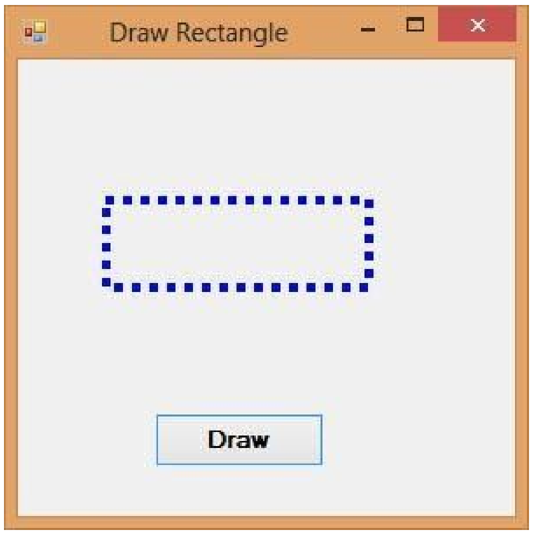


Figure 20.6

If you change the DashStyle value to DashDotDot, you will get the rectangle as shown in Figure 20.7

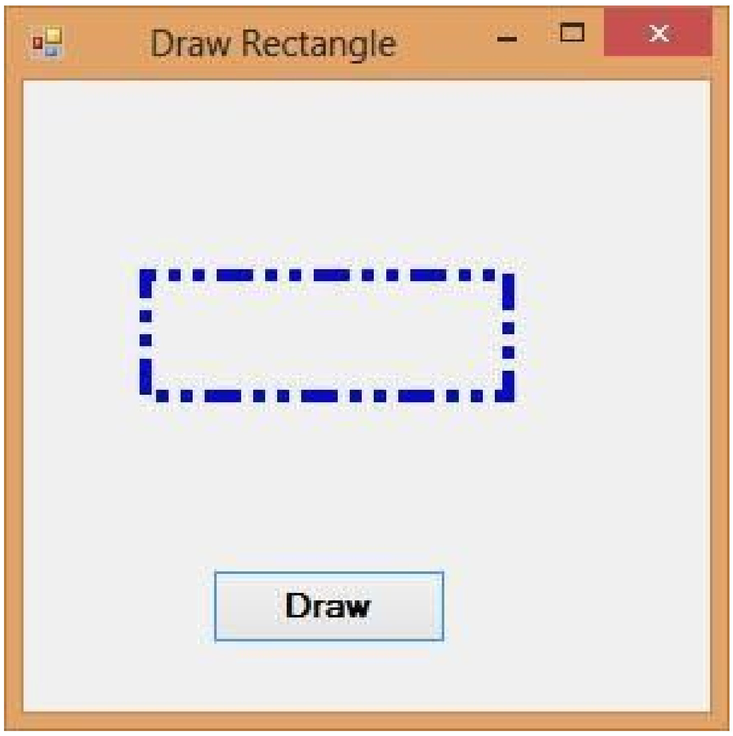


Figure 20.7

20.11 Drawing an Ellipse

Now we shall learn how to draw ellipse and circle. First, we need to understand the principle behind drawing an ellipse in Visual Basic 2017. The basic structure of most shapes is a rectangle, ellipse is no exception. Ellipse is an oval shape that is bounded by a rectangle, as shown in Figure 20.7

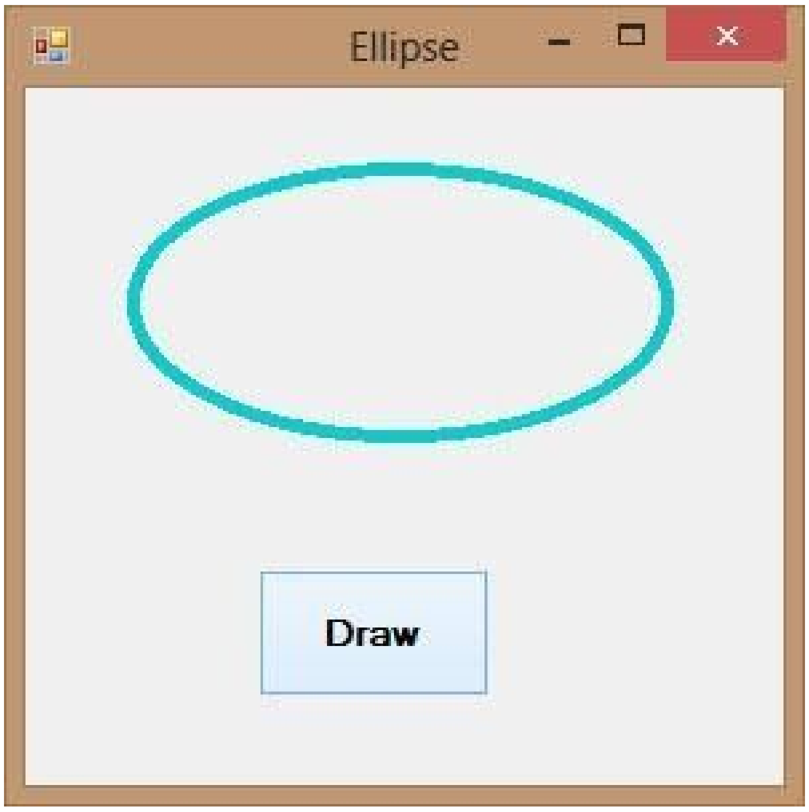


Figure 20.8

Therefore, we need to create a Rectangle object before we can draw an ellipse. This rectangle serves as a bounding rectangle for the ellipse. However, you still need to use the **`DrawEllipse`** method to finish the job. On the other hand, we can also draw an ellipse with the **`DrawEllipse`** method without first creating a rectangle. We shall show you both ways. In the first method, let's say you have created a rectangle object known as **`myRectangle`** and a pen object as `myPen`, then you can draw an ellipse using the following statement:

```
myGraphics.DrawEllipse(myPen, myRectangle)
```

* Assume you have also already created the Graphics object **`myGraphics`**.

Example 20.7 Drawing an Ellipse

```
Private Sub BtnDraw_Click(sender As Object, e As EventArgs) Handles
BtnDraw.Click
Dim myPen As Pen
myPen = New Pen(Drawing.Color.DarkTurquoise, 5)
Dim myGraphics As Graphics = Me.CreateGraphics
Dim myRectangle As New Rectangle
myRectangle.X = 40
myRectangle.Y = 30
myRectangle.Width = 200
myRectangle.Height = 100
myGraphics.DrawEllipse(myPen, myRectangle)
End Sub
```

The output image is shown in Figure 20.9

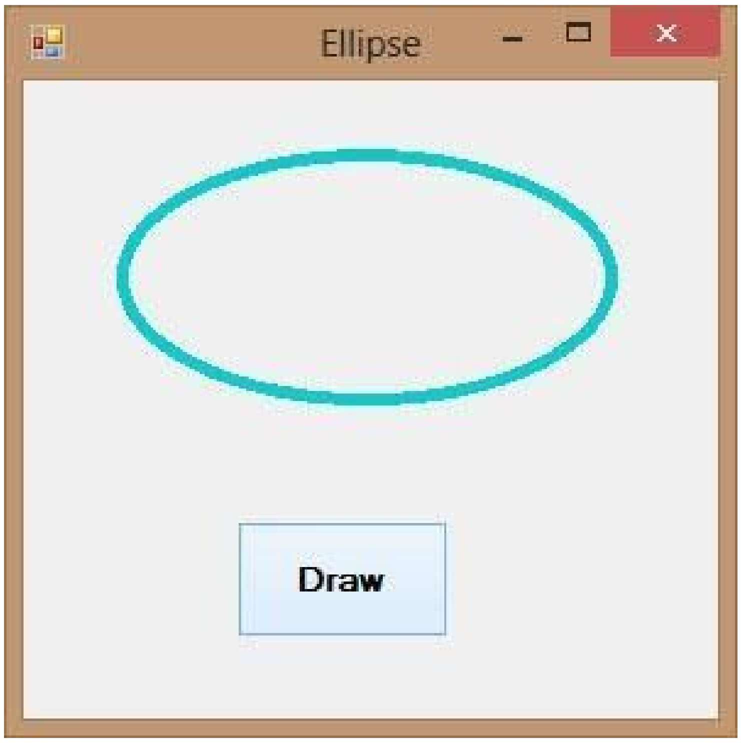


Figure 20.9

The second method is using the **`DrawEllipse`** method without creating a rectangle object. Off course you still have to create the Graphics and the Pen objects. The syntax is:

```
myGraphics.DrawEllipse(myPen, X,Y,Width, Height)
```

Where (X,Y) are the coordinates of the upper left corner of the bounding rectangle, width is the width of the ellipse and height is the height of the ellipse.

Example 20.8 Drawing an Ellipse

```
Private Sub BtnDraw_Click(sender As Object, e As EventArgs) Handles
BtnDraw.Click
Dim myPen As Pen
myPen = New Pen(Drawing.Color.DarkTurquoise, 5)
Dim myGraphics As Graphics = Me.CreateGraphics
myGraphics.DrawEllipse(myPen, 40, 30, 200, 100)
End Sub
```

20.12 Drawing a Circle

After you have learned how to draw an ellipse, drawing a circle becomes quite simple. We use the same methods used in the preceding section but modify the width and height so that they are of the same values.

The following examples draw the same circle.

Example 20.9 Draw a Circle

```
Dim myPen As Pen
myPen = New Pen(Drawing.Color.DarkTurquoise, 5)
Dim myGraphics As Graphics = Me.CreateGraphics
Dim myRectangle As New Rectangle
myRectangle.X = 90
myRectangle.Y = 30
myRectangle.Width = 100
myRectangle.Height = 100
myGraphics.DrawEllipse(myPen, myRectangle)
```

Note: You may use this line to define the coordinates and size of the circle.

```
myGraphics.DrawEllipse(myPen, 90, 30, 100, 100)
```

The output image is shown in Figure 20.10

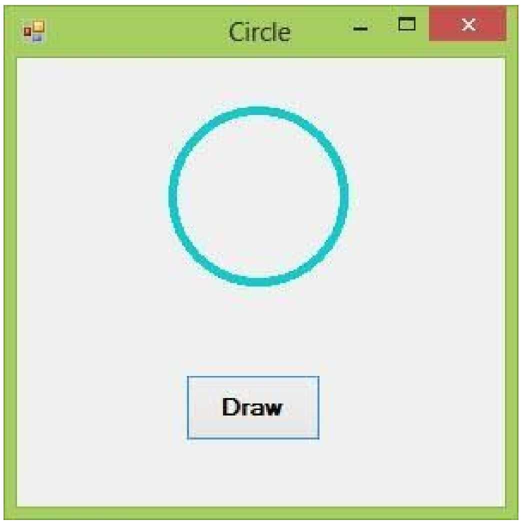


Figure 20.10

20.13 Drawing Text

In order to draw text on the screen, we can use the DrawString method. The format is as follows:

```
myGraphics.DrawString(myText, myFont, mybrush, X , Y)
```

Where **`myGraphics`** is the Graphics object, myText is the text you wish to display on the screen, myFont is the font object created by you, myBrush is the brush style created by you and X, Y are the coordinates of upper left corner of the Text.

You can create the Font object in visual basic 2017 using the following statement:

```
myFont = New System.Drawing.Font("Verdana", 20)
```

Where the first argument of the font is the font typeface, and the second argument is the font size. You can add a third argument as font style, either bold, italic, underline. Here are the examples:

```
myFont = New System.Drawing.Font("Verdana", 20, FontStyle.Bold)
myFont = New System.Drawing.Font("Verdana", 20, FontStyle.Underline)
myFont = New System.Drawing.Font("Verdana", 20, FontStyle.Italic)
myFont = New System.Drawing.Font("Verdana", 20, FontStyle.Regular)
```

To create your Brush object, you can use the following statement:

```
Dim myBrush As Brush
myBrush = New Drawing.SolidBrush(Color.BrushColor)
```

Besides the seven colors, some of the common Brush colors are AliceBlue, AquaMarine Beige, DarkMagenta, DrarkOliveGreen, SkyBlue and more. You don't have to remember the names of all the colors, the intelliSense will let you browse through the colors in a drop-down menu once you type the dot after the word Color.

Now we shall proceed to draw the font using the sample code in Example 20.9, as follows:

Example 20.10 Drawing Text

```
Private Sub BtnDraw_Click(sender As Object, e As EventArgs) Handles
BtnDraw.Click
Dim myGraphics As Graphics = Me.CreateGraphics
Dim myFont As Font
Dim myBrush As Brush
myBrush = New Drawing.SolidBrush(Color.DarkOrchid)
myFont = New System.Drawing.Font("Verdana", 20, FontStyle.Underline)
myGraphics.DrawString("Visual Basic 2017", myFont, myBrush, 10, 10)
End Sub
```

The runtime interface is as shown in Figure 20.11

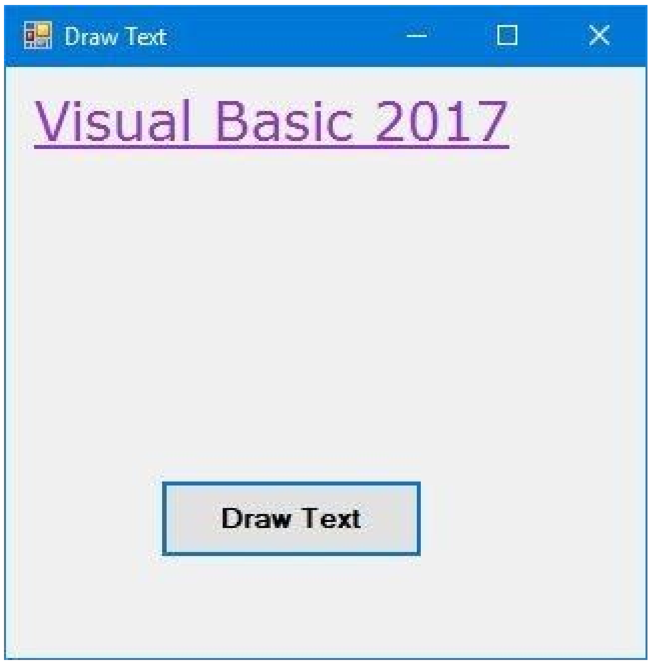


Figure 20.11

The preceding example can be modified if you do not want to create the Font and the Brush objects. You can use the font of an existing object such as the Form and the System Colors. Replace the last line in the preceding example with this line (you need to delete the lines that create the Brush and the Font objects as well)

```
myGraphics.DrawString("Visual Basic 2017", me.Font,
System.Drawing.Brushes.DarkOrchid, 10, 10)
```

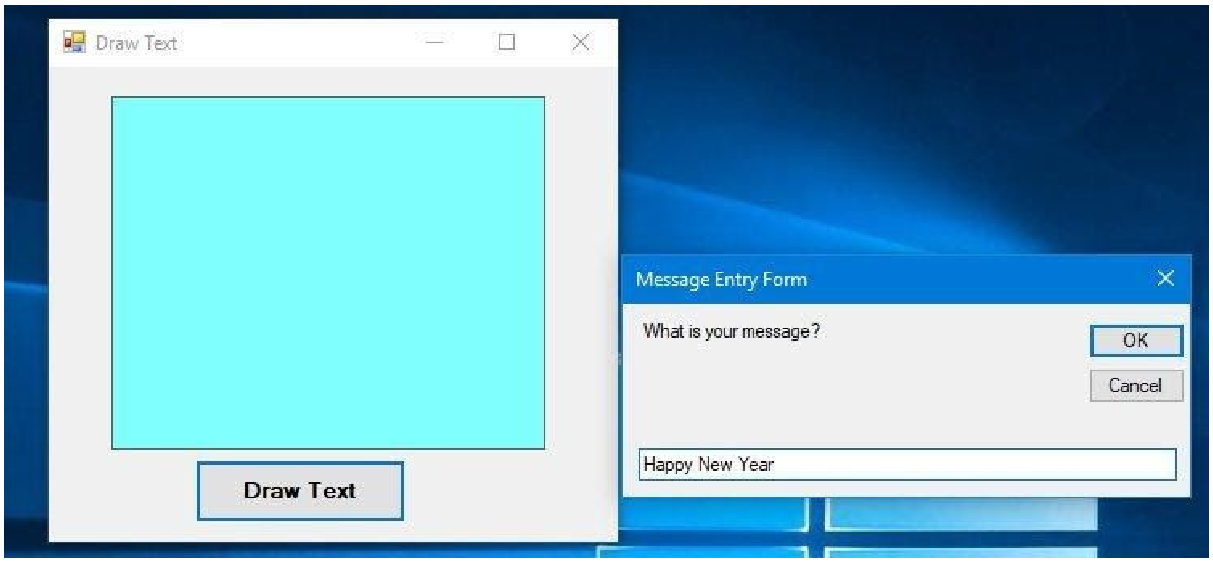
You can also add a input box to let the user enter his or her message then display the message on the screen. Besides that, you can insert a picture box and draw text on it instead of drawing on the form. In this example, insert a picture box and rename it as MyPicBox. Now, when you declare the Graphics object, you use the object MyPicBox instead of Me (the Form object)

The Code is as shown in Example 20.11

Example 20.11 Drawing Text Input by the User

```
Private Sub BtnDrawText_Click(sender As Object, e As EventArgs) Handles
BtnDrawText.Click
Dim myGraphics As Graphics = MyPicBox.CreateGraphics
Dim myFont As Font
Dim myBrush As Brush
Dim userMsg As String
userMsg = InputBox("What is your message?", "Message Entry Form",
"Enter your message here", 100, 200)
myBrush = New Drawing.SolidBrush(Color.DarkOrchid)
myFont = New System.Drawing.Font("Verdana", 20, FontStyle.Underline)
myGraphics.DrawString(userMsg, myFont, myBrush, 10, 10)
End Sub
```

The runtime interfaces are shown in Figure 20.12 and 20.13



What is your message?

Happy New Year

Figure 20.12

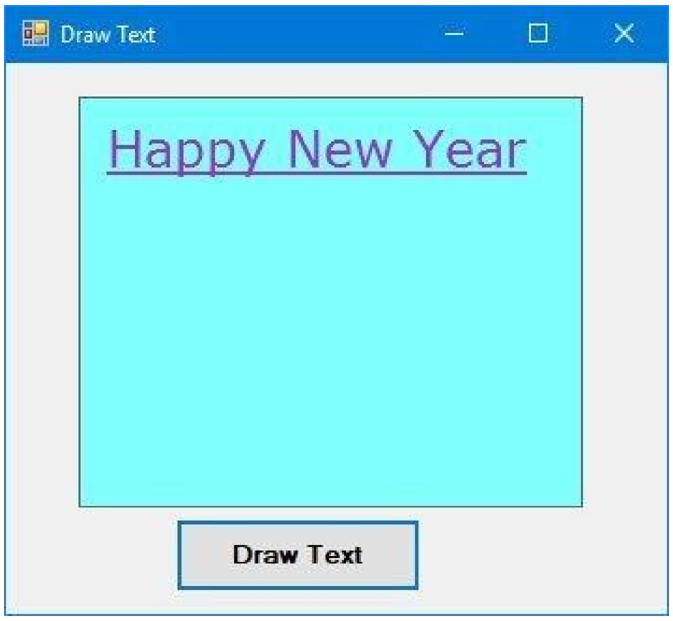


Figure 20.13

20.14 Drawing Polygons

Polygon is a closed plane figure bounded by three or more straight sides. To draw a polygon on the screen, we need to define the coordinates of all the points (also known as vertices) that joined up to form the polygon. The syntax to defines the points of a polygon with vertices A1,A2,A3,A4…….An is as follows.

```
Dim A1 As New Point(X1,Y1)
Dim A2 As New Point(X2,Y2)
Dim A3 As New Point(X3,Y3)
Dim A4 As New Point(X4,Y4)
.
.
Dim An as New Point(Xn,Yn)
```

After declaring the points, we need to define a point structure that group all the points together using the following syntax:

```
Dim myPoints As Point() = {A1, A2, A3,....,An}
```

Finally, create the graphics object and use the `DrawPolygon` method to draw the polygon using the following syntax:

```
Dim myGraphics As Graphics = Me.CreateGraphics
myGraphics.DrawPolygon(myPen, myPoints)
```

where `myPen` is the Pen object created using the following syntax:

```
myPen = New Pen(Drawing.Color.Blue, 5)
```

A triangle is a polygon with three vertices. Let us examine Example 20.10

Example 20.12 Drawing a Triangle

```
Private Sub BtnDraw_Click(sender As Object, e As EventArgs) Handles
BtnDraw.Click
Dim myPen As Pen
Dim A As New Point(10, 10)
Dim B As New Point(100, 50)
Dim C As New Point(60, 150)
Dim myPoints As Point() = {A, B, C}
myPen = New Pen(Drawing.Color.Blue, 5)
Dim myGraphics As Graphics = Me.CreateGraphics
myGraphics.DrawPolygon(myPen, myPoints)
End Sub
```

Running the program produces the image as shown in Figure 20.14

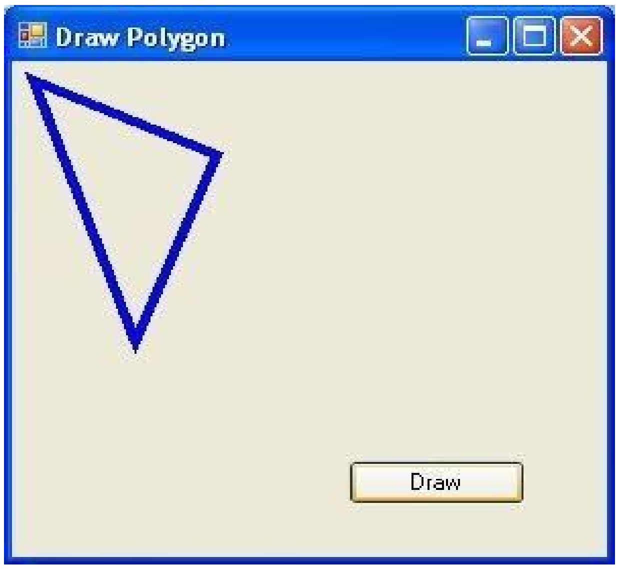


Figure 20.14

Example 20.13 Drawing a Quadrilateral

A quadrilateral is a polygon consists of four sides, so you need to define four vertices. The Code is as follows.

```
Private Sub BtnDraw_Click(sender As Object, e As EventArgs) Handles
BtnDraw.Click
Dim myPen As Pen
Dim A As New Point(10, 10)
      Dim B As New Point(100, 50)
      Dim C As New Point(120, 150)
      Dim D As New Point(60, 200)
      Dim myPoints As Point() = {A, B, C, D}
      myPen = New Pen(Drawing.Color.Blue, 5)
      Dim myGraphics As Graphics = Me.CreateGraphics
      myGraphics.DrawPolygon(myPen, myPoints)
End Sub
```

The output image is as shown in Figure 20.15

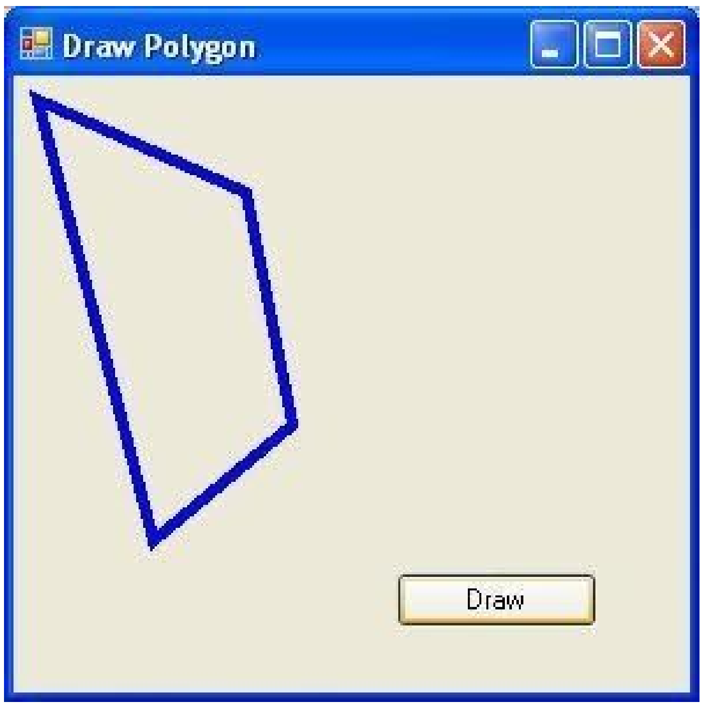


Figure 20.15

20.15 Drawing a Pie

In order to draw a pie, you can use the `DrawPie` method of the graphics object. As usual, you need to create the Graphics and the Pen objects. The syntax for drawing a pie is:

```
myGraphics.DrawPie(myPen, X, Y, width,height, StartAngle, SweepAngle)
```

Where X and Y are the coordinates the bounding rectangle, other arguments are self-explanatory. Both `StartAngle` and **`SweepAngle`** are measured in degree. **`SweepAngle`** can take possible or negative values. If the value is positive, it sweeps through clockwise direction while negative means it sweeps through anticlockwise direction.

Example 20.14 Drawing a pie that sweeps clockwise through 60 degree.

```
Private Sub BtnDraw_Click(sender As Object, e As EventArgs) Handles
BtnDraw.Click
Dim myPen As Pen myPen = New Pen(Drawing.Color.Blue, 5)
     Dim myGraphics As Graphics = Me.CreateGraphics
     myGraphics.DrawPie(myPen, 50,50, 150,150,0,60)
End Sub
```

The output image is as shown in Figure 20.16

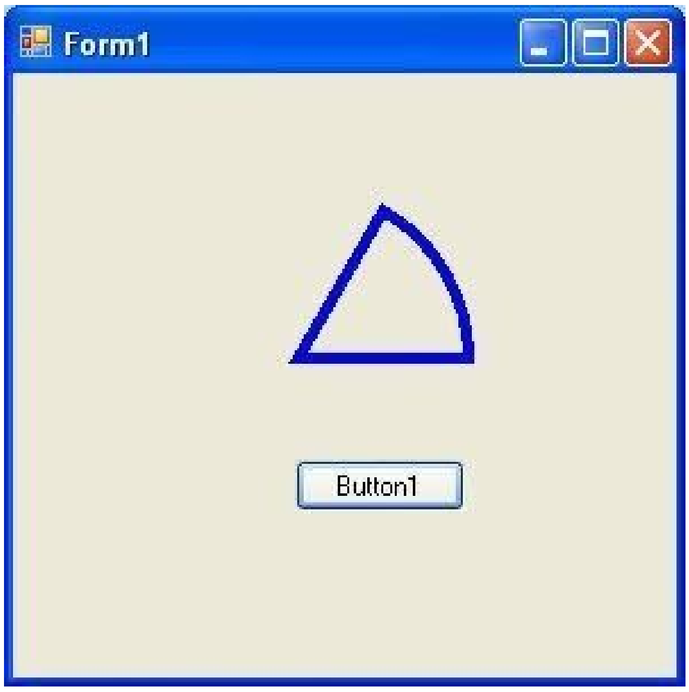


Figure 20.16

20.16 Filling Shapes with Color

There several methods that we can use to fill shapes . They are **`FillRectangle,`** **`FillEllipse`** **,** **`FillPolygon`** and **`FillPie.`**

In order to fill the above shapes with color, we need to create the Brush object using the following syntax:

```
myBrush = New SolidBrush(Color.myColor)
```

myColor can be any color such as red, blue, yellow and more. You do not have to worry about the names of the colors because the intellisense will display the colors and enter the period after the Color keyword.

Example 20.15 Drawing and Filling a Rectangle with Color

The syntax to fill a rectangle with the color defined by the brush object is:

```
myGraphics.FillRectangle (myBrush, 0, 0, 150, 150)
```

The Code

```
Private Sub BtnDraw_Click(sender As Object, e As EventArgs) Handles
BtnDraw.ClickDim myPen As Pen
Dim myBrush As Brush
Dim myGraphics As Graphics = Me.CreateGraphics
myPen = New Pen(Drawing.Color.Blue, 5)
myBrush = New SolidBrush(Color.Coral)
myGraphics.DrawRectangle(myPen, 65, 50, 150, 150)
myGraphics.FillRectangle(myBrush, 65, 50, 150, 150)
End Sub
```

The output is shown in Figure 20.17

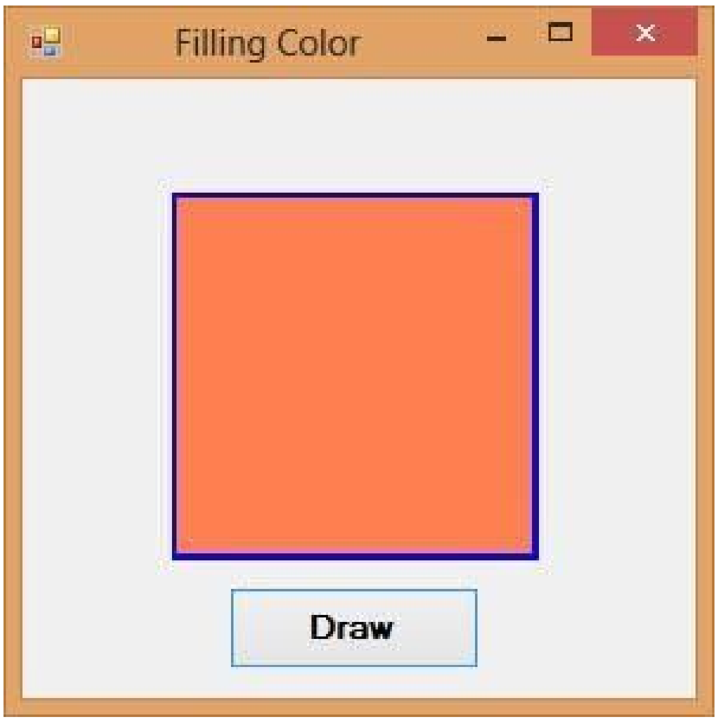


Figure 20.17

*Note that if you omit the line `myGraphics.DrawRectangle(myPen, 65, 50, 150, 150)`, you will get a solid rectangle without outline, as shown in in Figure 20.18

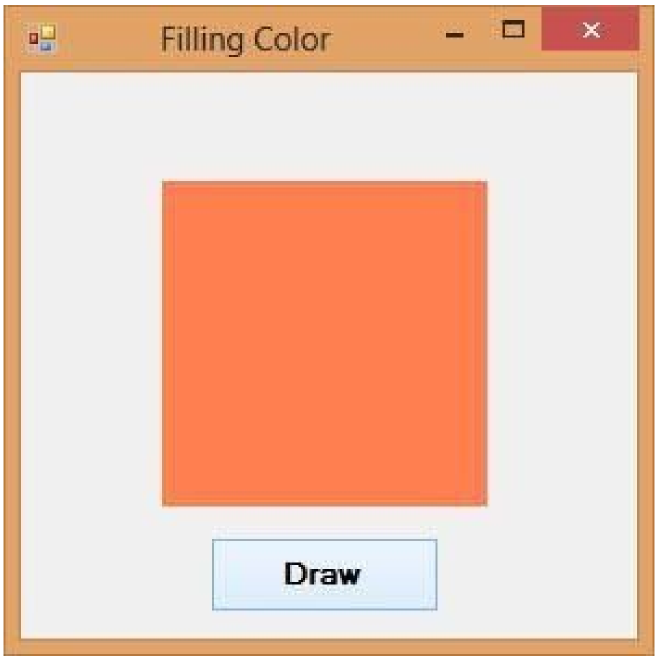


Figure 20.18

Example 20.16 Drawing and Filling an Ellipse with Color

The syntax to fill an ellipse with the color defined by the brush object is:

```
myGraphics.FillEllipse (myBrush, 0, 0, 150, 150)
```

The Code

```
Private Sub BtnDraw_Click(sender As Object, e As EventArgs) Handles
BtnDraw.Click
Dim myPen As Pen
Dim myBrush As Brush
Dim myGraphics As Graphics = Me.CreateGraphics
myPen = New Pen(Drawing.Color.Blue, 5)
myBrush = New SolidBrush(Color.Coral)
myGraphics.DrawEllipse(myPen, 50, 50, 180, 100)
myGraphics.FillEllipse(myBrush, 50, 50, 180, 100)
End Sub
```

The output interface is as shown in Figure 20.19

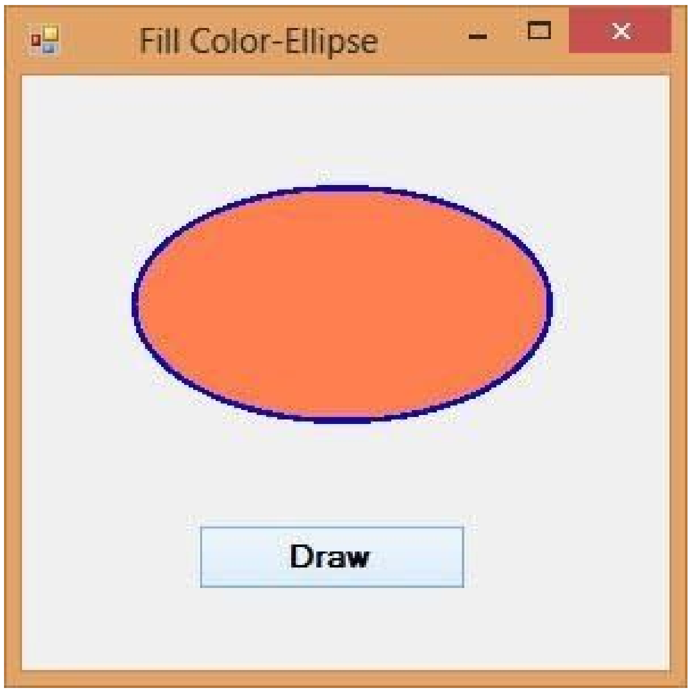


Figure 20.19

*If you omit the line **`myGraphics.DrawEllipse(myPen, 50, 50, 180, 100)`**, you will get a solid ellipse without outline.

Example 20.17 Drawing and Filling a Polygon with Color

The syntax to fill a polygon with the color defined by the brush object is:

```
myGraphics.FillPolygon(myBrush, myPoints)
```

The Code

```
Private Sub BtnDraw_Click(sender As Object, e As EventArgs) Handles
BtnDraw.Click
Dim myPen As Pen
Dim myBrush As Brush
Dim A As New Point(70, 10)
Dim B As New Point(170, 50)
Dim C As New Point(200, 150)
Dim D As New Point(140, 200)
Dim myPoints As Point() = {A, B, C, D}
myPen = New Pen(Drawing.Color.Blue, 5)
```

```
myBrush = New SolidBrush(Color.Coral)
Dim myGraphics As Graphics = Me.CreateGraphics
myGraphics.DrawPolygon(myPen, myPoints)
myGraphics.FillPolygon(myBrush, myPoints)
End Sub
```

The output interface is as shown in Figure 20.20

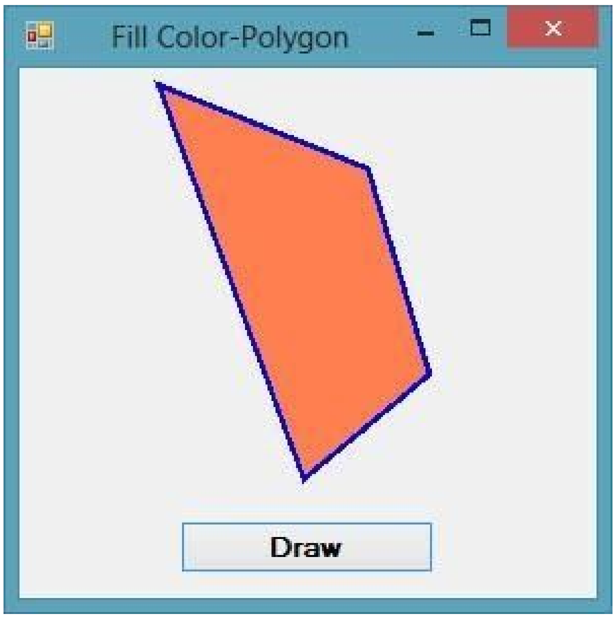


Figure 20.20

* If you omit `myGraphics the line DrawPolygon(myPen, myPoints),` you will get a polygon without outline

Example 20.18 Drawing and Filling a Pie

The syntax to fill a pie with the color defined by the brush object is:

```
myGraphics.FillPie(myBrush, X, Y, width, height, StartAngle, SweepAngle)
```

The Code

```
Private Sub BtnDraw_Click(sender As Object, e As EventArgs) Handles
BtnDraw.Click
Dim myPen As Pen
Dim myBrush As Brush
Dim myGraphics As Graphics = Me.CreateGraphics
myPen = New Pen(Drawing.Color.Blue, 5)
```

```
myBrush = New SolidBrush(Color.Coral)
myGraphics.DrawPie(myPen, 30, 40, 150, 150, 0, 60)
myGraphics.FillPie(myBrush, 30, 40, 150, 150, 0, 60)
End Sub
```

The output is as shown in Figure 20.21

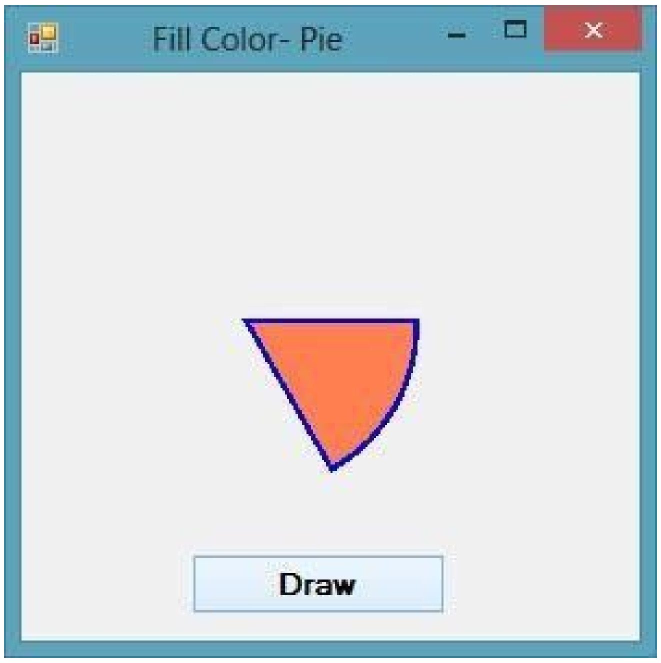


Figure 20.21

Summary

- In section 20.1, you have understood the concept of graphics creation
- In section 20.2, you have learned how to create the Graphics object
- In section 20.3, you have learned how to create the Pen object
- In section 20.4, you have learned how to draw a line
- In section 20.5, you have learned how to draw a rectangle
- In section 20.6, you have learned how to customize line style
- In section 20.7, you have learned how to draw an ellipse
- In section 20.8, you have learned how to draw a circle
- In section 20.9, you have learned how to draw a polygon
- In section 20.10, you have learned how to draw a pie
- In section 20.12, you have learned how to fill shapes with colors

Chapter 21 Using Timer

- ❖ Learn how to create a digital clock
- ❖ Learn how to create a stopwatch
- ❖ Learn how to create a digital dice

Timer is a useful control in Visual Basic 2017. It can be used to program events that are time related. For example, you need the timer to create a clock, a stopwatch, a dice, animation and more. Timer is a hidden control at runtime, like the engine of an automobile. We shall illustrate the usage of timer through a few examples.

Example 21.1 Creating a Digital Clock

To create the clock, first start a new project in Visual Basic 2017 and select a new Windows Application. You can give the project any name you wish, but we will name it `MyClock`. Change the text of the Form1 to `MyClock` in the properties window. Now add the Timer control to the form by double-clicking it in the ToolBox. Next, insert a label control into the form. Change the Font size of the label to any size you wish and set the Font alignment to be middle center.

We shall set the `Interval` property of the Timer control to 1000, which is equal to a one second interval (1 unit is 1 millisecond). Remember to set the `MaximizeBox` property of Form1 to false so that the user cannot enlarge the clock. You also need to ensure that the `Enabled` property of the Timer control is set to `True` so that the clock starts running as soon as it is loaded.

Now, you are ready for the coding. You would be surprised that what you need to create a clock is only a one-line code, that is:

```
Label1.Text = TimeOfDay
```

*`TimeOfDay()` is a Visual Basic 2017 function that returns the current time today based on your computer system time.

Click on the Timer control and enter the code as shown below:

```
Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
LblClock.Text = TimeOfDay
End Sub
```

The digital clock is as shown in Figure 21.1

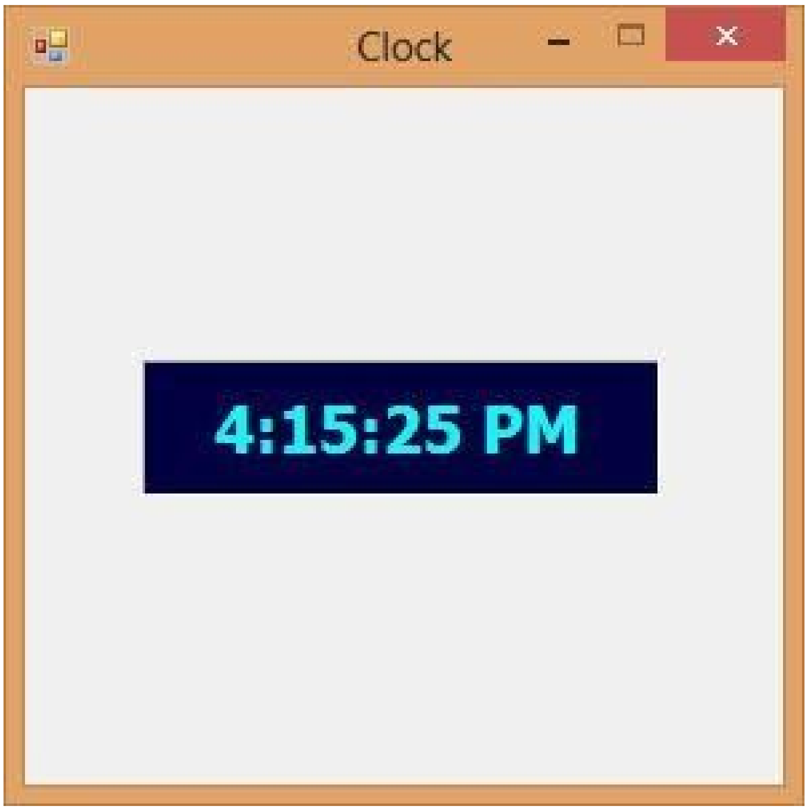


Figure 21.1

Example 21.2 Creating a Stopwatch

We can create a stopwatch using the Timer control. Start a new project and name it stopwatch. Change the Form1 caption to `Stopwatch`. Insert the Timer control into the form and set its interval to 1000 which is equal to one second. Besides that, set the timer Enabled property to False so that it will not start ticking when the program is started. Insert three buttons and change their names to `BtnStart`, `BtnStop` and `BtnReset` respectively. Change their text to “Start”, “Stop” and “Reset” accordingly. Now, enter the code as follows:

```
Private Sub BtnStart_Click(sender As Object, e As EventArgs) Handles
BtnStart.Click
Timer1.Enabled = True
End Sub
Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles
Timer1.Tick
LblPanel.Text = Val(LblPanel.Text) + 1
End Sub
Private Sub BtnStop_Click(sender As Object, e As EventArgs) Handles
BtnStop.Click
Timer1.Enabled = False
End Sub
Private Sub BtnReset_Click(sender As Object, e As EventArgs) Handles
BtnReset.Click
LblPanel.Text = 0
End Sub
```

The Interface of the Stopwatch is as shown in Figure 21.2

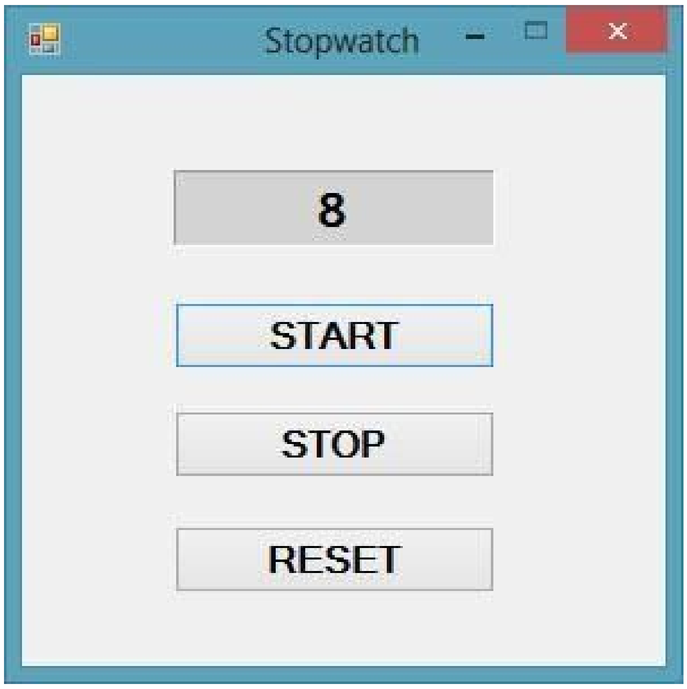


Figure 21.2

Example 21.3 Creating a Digital Dice

To create a dice, you need to generate random numbers using the `Rnd` function. The `Rnd` function generates numbers between 0 and 1. However, you must use the `Int` function to obtain random integers. The following statement generates random integers from 1 to 6 is as follows:

```
n = Int(1 + Rnd() * 6)
```

In the code, we introduce the variable m to control the length of time of the rolling process. If m is more than 1000, then the rolling process will stop by setting the timer enabled property to False. Set the timer interval to 10 so that the number changes every 0.01 second.

The Code

```
Dim n, m As Integer
Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles
Timer1.Tick
m = m + 10
```

```
If m < 1000 Then
n = Int(1 + Rnd() * 6)
LblDice.Text = n
Else
Timer1.Enabled = False
m = 0
End If
End Sub
Private Sub BtnRoll_Click(sender As Object, e As EventArgs) Handles
BtnRoll.Click
Timer1.Enabled = True
End Sub
```

Running the program produces a dice with fast changing numbers which stops at a specific number. The interface is as shown in Figure 21.3

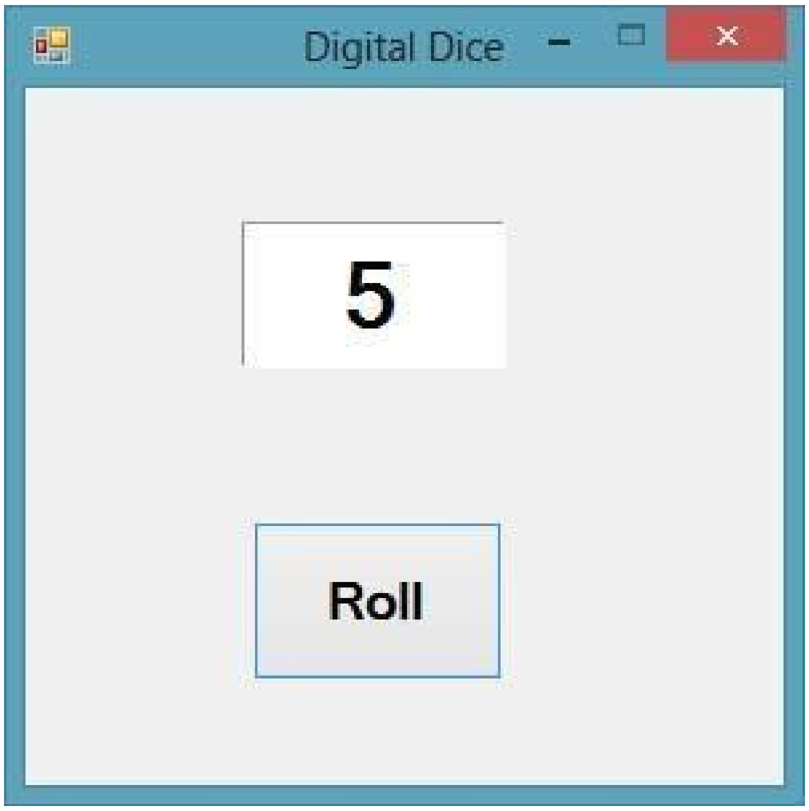


Figure 21.3

Summary

- In section 21.1, you have learned how to create a digital clock

- In section 21.2, you have learned how to create a digital stopwatch
- In section 21.3, you have learned how to create a digital dice

Chapter 22 Creating Animation

- ❖ Learn how to create motion
- ❖ Learn how to create a graphical dice
- ❖ Learn how to create a slot machine

Visual Basic 2017 can be used to create animation. In the preceding chapter, we have learned how to create animation using timer. In fact, the programs we have created in the previous chapter such as the stopwatch and the digital dice are animated programs. In this chapter, we shall show you more advanced animated programs.

We can create a continuously moving object using timer. The motion can be from left to right or from top to bottom motion or diagonal.

Example 22.1 Creating Moving Object

To create a moving object, insert a picture box into the form. In the picture box properties window, select the image property and click to import an image file from your storage devices such as your hard drive. Next, insert a Timer control into the form and set its interval property to 100, which is equivalent to 0.1 second. Finally, add two buttons to the form, name one of them as `AnimateBtn` and the other one as `StopBtn`, and change to caption to Animate and Stop respectively.

We make use of the Left property of the picture box to create the motion. PictureBox.Left means the distance of the PictureBox from the left border of the Form.

```
Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
If PictureBox1.Left < Me.Width Then
PictureBox1.Left = PictureBox1.Left + 10
Else
PictureBox1.Left = 0
End If
End Sub
```

In the code above, `Me.Width` represents the width of the Form. If the distance of the PictureBox from the left is less than the width of the Form, a value of 10 is added to the distance of the PictureBox from the left border each time the Timer tick, or every 0.1 second in this example. When the distance of the PictureBox from the left border is equal to the width of the form, the distance from the left border is set to 0, which move the PictureBox object to the left border and then move left again, thus creates an oscillating motion from left to right. We need to insert a button to stop the motion. The code is:

```
Timer1.Enabled = False
```

To animate the PictureBox object, we insert a button and enter the following code:

```
Timer1.Enabled = True
```

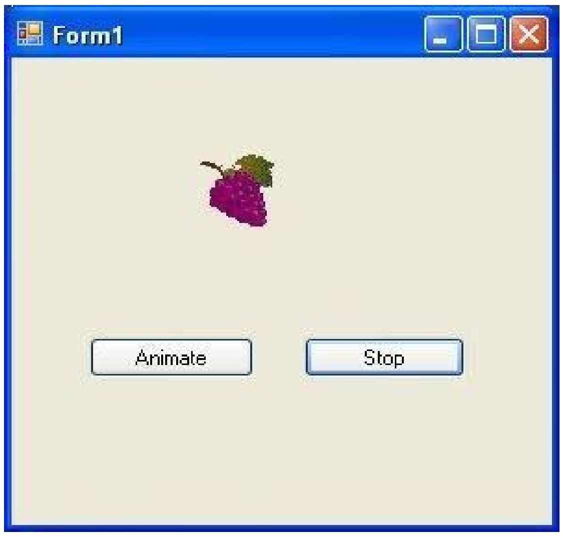


Figure 22.1 The runtime interface

Example 22.2 Creating an Animated Dice

In this program, we need to insert a timer and set its interval to 100, which means the drawings will refresh every 0.1 second. Next, insert a picture box which is used as the surface of a dice. Finally, add a button and change its text to Roll. Under the Timer sub procedure, we create the Graphics object and the Pen object following the procedures we have learned in preceding chapters. Next, we use a **`Do loop`** and the `Select Case` structure to cycle through all six surfaces of the dice. To create six random cases, we use the syntax **`n = Int(6 * Rnd()) + 1.`** We can stop the loop by introducing a variable t and the loop until condition. The condition we set here is t >1000, you can use any figure you wish.

The code

```
Private Sub BtnRoll_Click(sender As Object, e As EventArgs) Handles
BtnRoll.Click
Timer1.Enabled = True
End Sub
Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles
Timer1.Tick
Dim t As Integer
t = 0
Do
MyPicBox.Refresh()
Dim n As Integer
Dim myPen As Pen
myPen = New Pen(Drawing.Color.DarkTurquoise, 10)
Dim myGraphics As Graphics = MyPicBox.CreateGraphics
n = Int(6 * Rnd()) + 1
Select Case n
Case 1
myGraphics.DrawEllipse(myPen, 80, 80, 10, 10)
Case 2
myGraphics.DrawEllipse(myPen, 40, 40, 10, 10)
myGraphics.DrawEllipse(myPen, 120, 120, 10, 10)
Case 3
myGraphics.DrawEllipse(myPen, 40, 40, 10, 10)
myGraphics.DrawEllipse(myPen, 80, 80, 10, 10)
myGraphics.DrawEllipse(myPen, 120, 120, 10, 10)
```

```
Case 4
myGraphics.DrawEllipse(myPen, 40, 40, 10, 10)
myGraphics.DrawEllipse(myPen, 120, 40, 10, 10)
myGraphics.DrawEllipse(myPen, 40, 120, 10, 10)
myGraphics.DrawEllipse(myPen, 120, 120, 10, 10)
Case 5
myGraphics.DrawEllipse(myPen, 40, 40, 10, 10)
myGraphics.DrawEllipse(myPen, 120, 40, 10, 10)
myGraphics.DrawEllipse(myPen, 80, 80, 10, 10)
myGraphics.DrawEllipse(myPen, 40, 120, 10, 10)
myGraphics.DrawEllipse(myPen, 120, 120, 10, 10)
Case 6
myGraphics.DrawEllipse(myPen, 40, 40, 10, 10)
myGraphics.DrawEllipse(myPen, 120, 40, 10, 10)
myGraphics.DrawEllipse(myPen, 40, 80, 10, 10)
myGraphics.DrawEllipse(myPen, 120, 80, 10, 10)
myGraphics.DrawEllipse(myPen, 40, 120, 10, 10)
myGraphics.DrawEllipse(myPen, 120, 120, 10, 10)
End Select
t = t + 1
Loop Until t > 1000
Timer1.Enabled = False
End Sub
```

The runtime interface is as shown in Figure 22.2

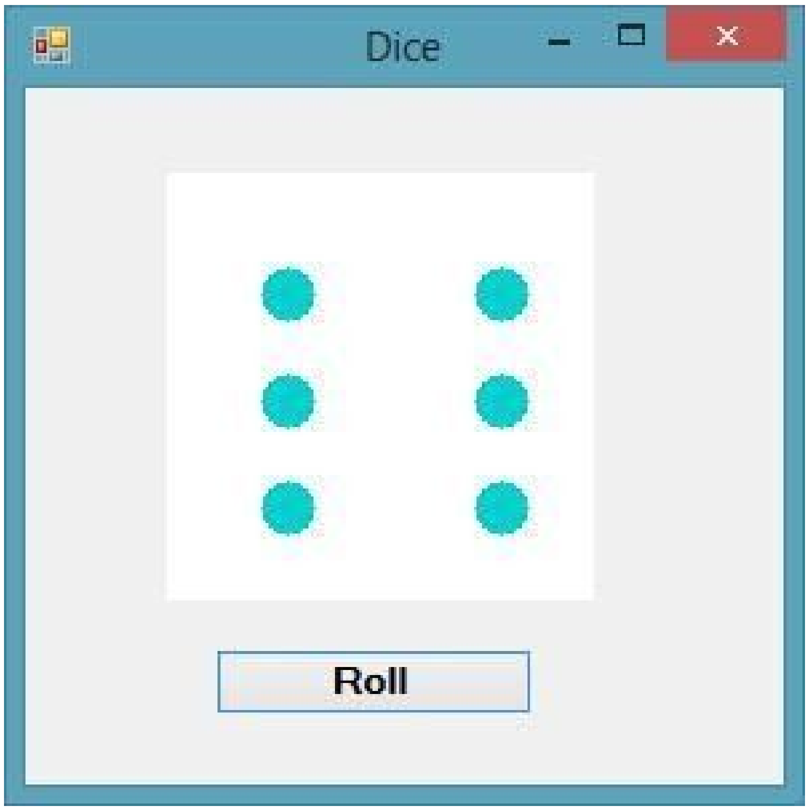


Figure 22.2

Example 22.3 Creating a Slot Machine

You can also create a slot machine using timer. In this program, we add three picture boxes, a timer, a button and a label. Set the timer interval to 10, which means the images will refresh every 0.01 second. In the code, we shall introduce four variables m,a, b and c, where m is used to stop the timer and a,b,c are used to generate random images using the syntax `Int(1 + Rnd() * 3)`. To load the images, we use the following syntax:

```
PictureBox.Image = Image.FromFile(Path of the image file)
```

We employ the `If...Then` structure to control the timer and the `Select Case...End Select` structure to generate the random images. The label is used to display the message of the outcomes.

The Code

```
Dim m, a, b, c As Integer
Private Sub BtnSpin_Click(sender As Object, e As EventArgs) Handles
BtnSpin.Click
Timer1.Enabled = True
End Sub
Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles
Timer1.Tick
m = m + 10
If m < 1000 Then
a = Int(1 + Rnd() * 3)
b = Int(1 + Rnd() * 3)
c = Int(1 + Rnd() * 3)
Select Case a
Case 1
PictureBox1.Image = Image.FromFile("C:\Image\apple.gif")
Case 2
PictureBox1.Image = Image.FromFile("C:\Image\grape.gif")
Case 3
PictureBox1.Image = Image.FromFile("C:\Image\strawberry.gif")
End Select
Select Case b
Case 1
PictureBox2.Image = Image.FromFile("C:\Image\apple.gif")
Case 2
PictureBox2.Image = Image.FromFile("C:\Image\grape.gif")
Case 3
PictureBox2.Image = Image.FromFile("C:\Image\strawberry.gif")
```

```
End Select
Select Case c
Case 1
PictureBox3.Image = Image.FromFile("C:\Image\apple.gif")
Case 2
PictureBox3.Image = Image.FromFile("C:\Image\grape.gif")
Case 3
PictureBox3.Image = Image.FromFile("C:\Image\strawberry.gif")
End Select
Else
Timer1.Enabled = False
m = 0
If a = b And b = c Then
LblMsg.Text = "Jackpot! You won $1,000,000"
Else
LblMsg.Text = "No luck, try again"
End If
End If
End Sub
```

The runtime interface is shown in Figure 32.3

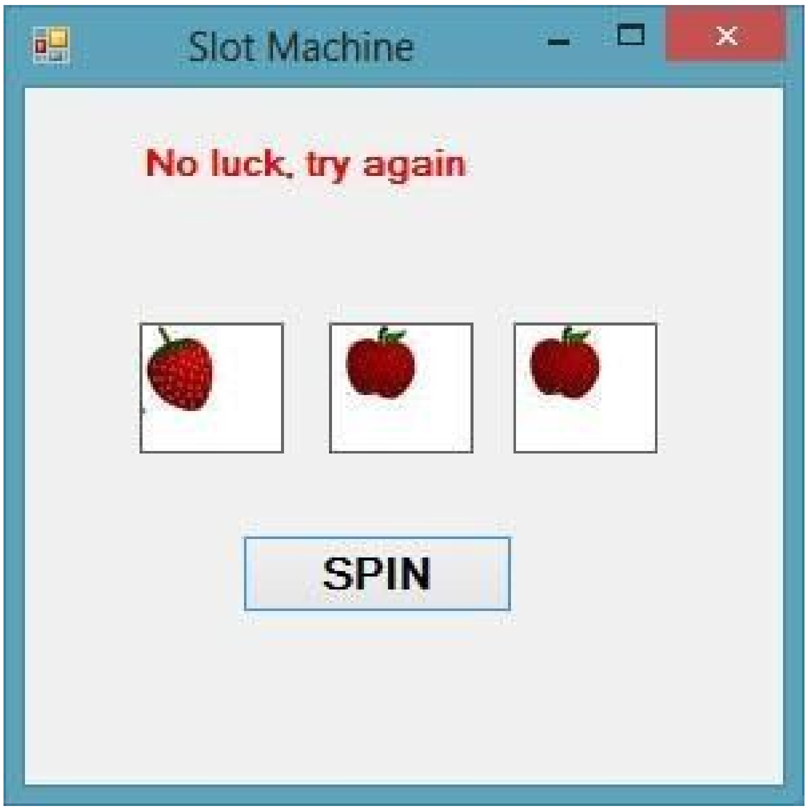


Figure 32.3

Summary

- In section 22.1, you have learned how to create motion
- In section 22.2, you have learned how to create a graphical dice
- In section 22.3, you have learned how to create a slot machine

Chapter 23 Working with Databases

- ❖ Understand the concepts of database
- ❖ Learn how to create a database
- ❖ Learn how to create connection to a database using ADO.Net
- ❖ Learn how to populate data
- ❖ Learn how to write code for browsing records in a database
- ❖ Learn how to write code for editing, saving ,adding and deleting records

23.1 Introduction to Database

In our daily life, we deal with many types of information or data such as names, addresses, money, date, stock quotes, statistics and more. If you are in business or working as a professional, you must handle even more data. For example, a doctor needs to keep track of patients' personal and medical information such as names, addresses, phone numbers as well as blood pressure readings, blood sugar readings, surgical history, medicines prescribed in the past and more. On the other hand, businesses usually must manage large amount of data pertaining to products and customers. All these data need to be organized into a database for the ease of data management.

In the past, people usually deal with data manually like using cards and folders. However, in present day fast pace global environment and Information age, it is no longer feasible to manage data manually. Most data are now managed using computer-based database management systems. Computer-based Database management systems can handle data much faster , more accurate and more efficient than human beings do. With the advent of the network and the Internet technologies, data can now be managed locally and remotely. Companies usually invest heavily in database management systems in order to run the organizations efficiently and effectively. Database management systems are usually used in running payroll system, inventory system, accounting system, payment system, order handling system, customer relationship management system (CRM) and more. Some of the commercial database management systems (DBMS) are Oracle, Microsoft SQL server and Microsoft Access

23.2 Creating a Database Application

A database management system typically deals with storing, modifying, and extracting information from a database. It can also add, edit and delete records from the database. However, a DBMS can be difficult to handle by ordinary people or businessmen who have no technological backgrounds. Fortunately, we can create user friendly database applications to handle the jobs with the DBMS running in the background. One of the best programs that can create such database application is none other than Visual Basic 2017.

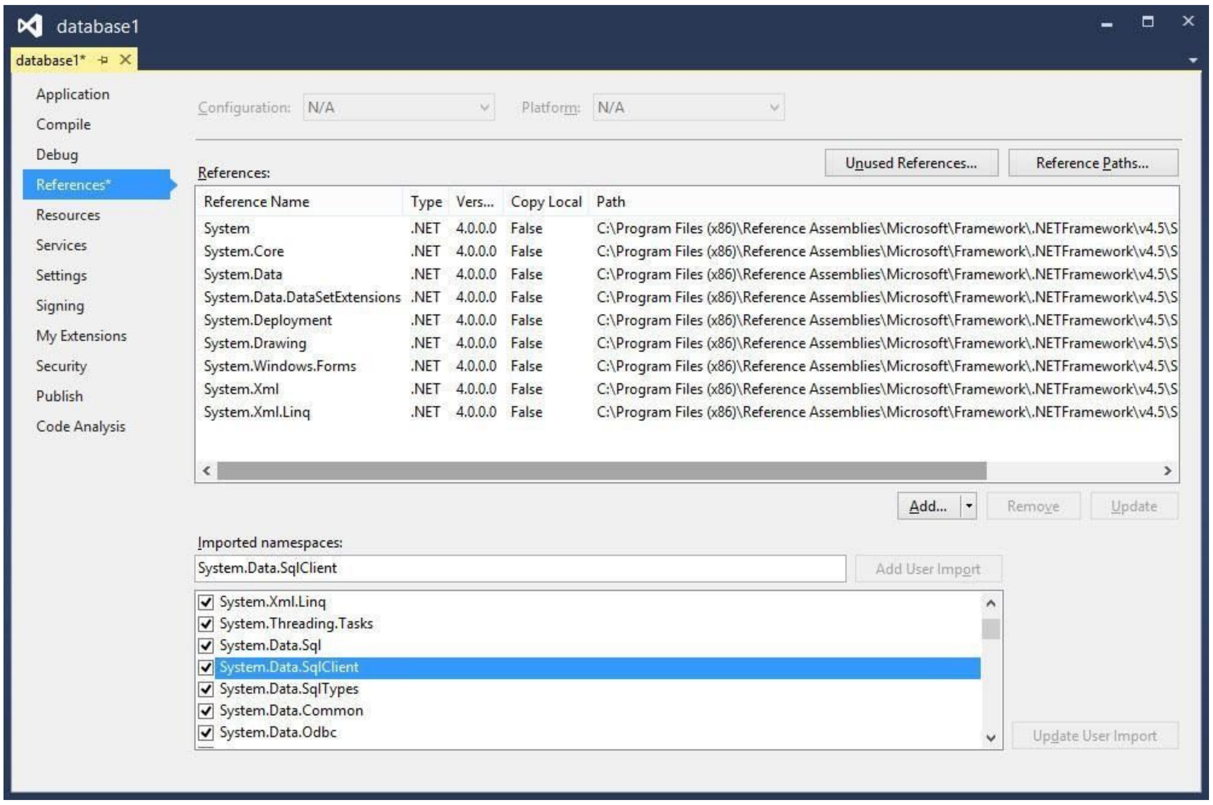
Visual Basic 2017 uses ADO.NET to handle databases. ADO.NET is Microsoft's latest database technology which can works with many other advanced database management systems such as Microsoft SQL server. In this chapter, we will develop codes that make use of SQL Server 2017; therefore, you need to have Microsoft SQL Server 2017 installed in your PC, you can download Microsoft SQL Server 2017 Express for free from

https://www.microsoft.com/en-us/sql-server/sql-server-downloads

To begin building the database project in Visual Basic 2017, launch Visual Basic 2017. You can name your project as Database Project 1 or whatever name you wish to call it. Next, change the default form's Text property to Contacts as we will be building a database of contact list. There are a few objects in ADO.NET that are required to build the database. There are:

- SqlConnection- to connect to a data source in SQL Server
- DataTable -to store data for navigation and manipulation
- DataAdapter- to populate a DataReader

The objects belong to the System.Data and the System.Xml namespace. Therefore, we need to reference them in the beginning before we can work with them. To reference the ADO.NET object, choose project from the menu then select Database Project 1 properties to display the project properties. Next click the References tab to show the active references for the project, as shown in Figure 23.1



database1

database1*

- Application
- Compile
- Debug
- References*
- Resources
- Services
- Settings
- Signing
- My Extensions
- Security
- Publish
- Code Analysis

Configuration: N/A Platform: N/A

References:

Reference Name	Type	Vers...	Copy Local	Path
System	.NET	4.0.0.0	False	C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\v4.5\S
System.Core	.NET	4.0.0.0	False	C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\v4.5\S
System.Data	.NET	4.0.0.0	False	C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\v4.5\S
System.Data.DataSetExtensions	.NET	4.0.0.0	False	C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\v4.5\S
System.Deployment	.NET	4.0.0.0	False	C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\v4.5\S
System.Drawing	.NET	4.0.0.0	False	C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\v4.5\S
System.Windows.Forms	.NET	4.0.0.0	False	C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\v4.5\S
System.Xml	.NET	4.0.0.0	False	C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\v4.5\S
System.Xml.Linq	.NET	4.0.0.0	False	C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\v4.5\S

Imported namespaces:

System.Data.SqlClient

- ✓ System.Xml.Linq
- ✓ System.Threading.Tasks
- ✓ System.Data.Sql
- ✓ System.Data.SqlClient
- ✓ System.Data.SqlTypes
- ✓ System.Data.Common
- ✓ System.Data.Odbc

Figure 23.1

Under imported namespaces, make sure system.data, System.Data.Sqlclient are selected, otherwise check them. Having done that, you need to click the Save All button on the toolbar and then return to the Visual Basic 2017 IDE.

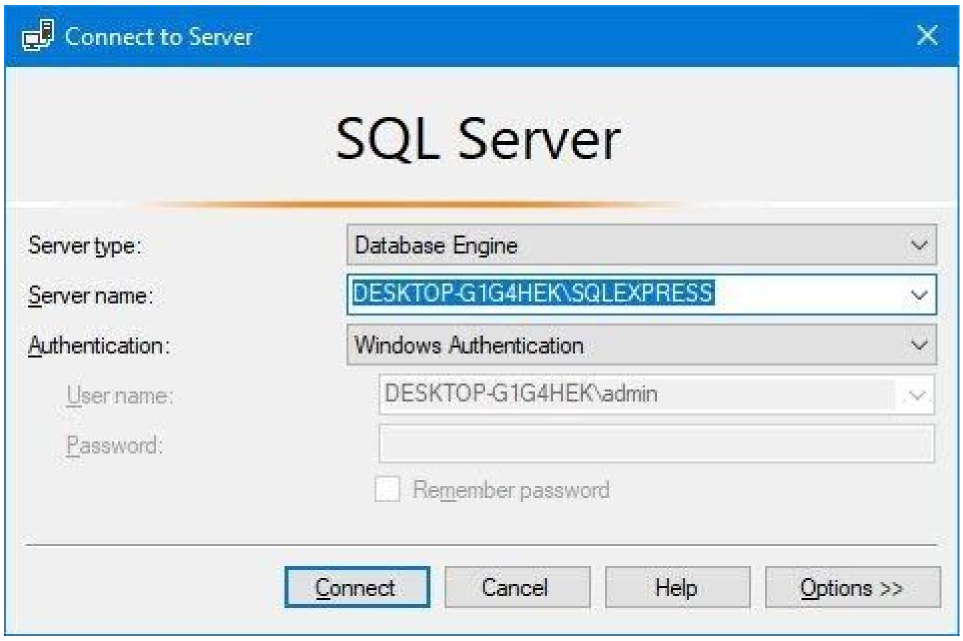
23.3 Creating Connection to a Database using ADO.NET

In Visual Basic 2017, we need to create connection to a database before we can access its data. Before we begin, let us create a new database. Since we are using SQL Server 2017 as the database engine, we will use **SQL Server Management Studio** to create a database file with the mdf extension. You can download **SQL Server Management Studio (SSMS)** for free.

from the link below:

https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms

We suggest you download the latest SSMS, SQL Server Management Studio 17.4. Upon launching SSMS, the initial dialog will ask you to connect it to the SQL server, as shown in Figure 23.2.



Server type: Database Engine
Server name: DESKTOP-G1G4HEK\SQLEXPRESS
Authentication: Windows Authentication
User name: DESKTOP-G1G4HEK\admin
Password:
Remember password

Figure 23.2

After clicking the ‘Connect’ button, SSMS will be connected to SQL server, as shown in Figure 23.3

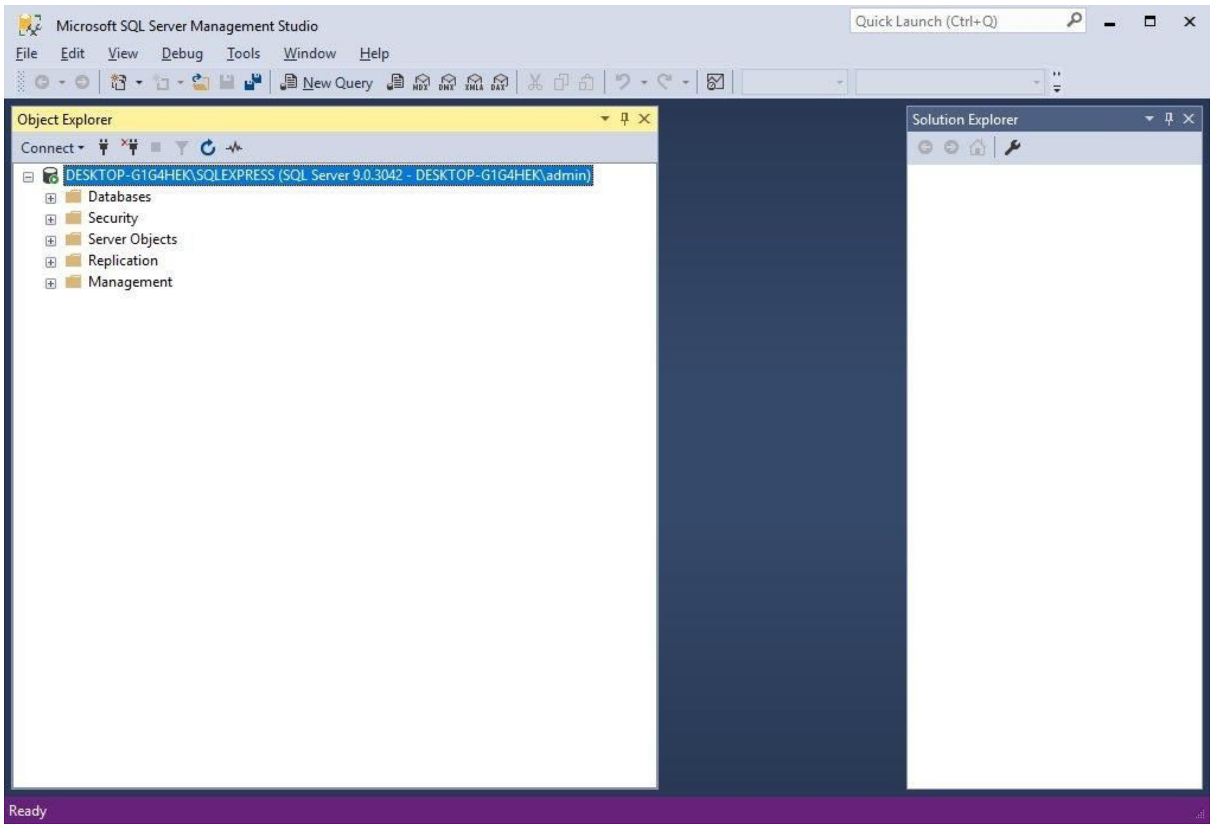


Figure 23.3

You can also disconnect from the SQL server anytime you wish. Next, we will create a database file. To create a new database, right-click on databases and select New Database, as shown in Figure 23.4

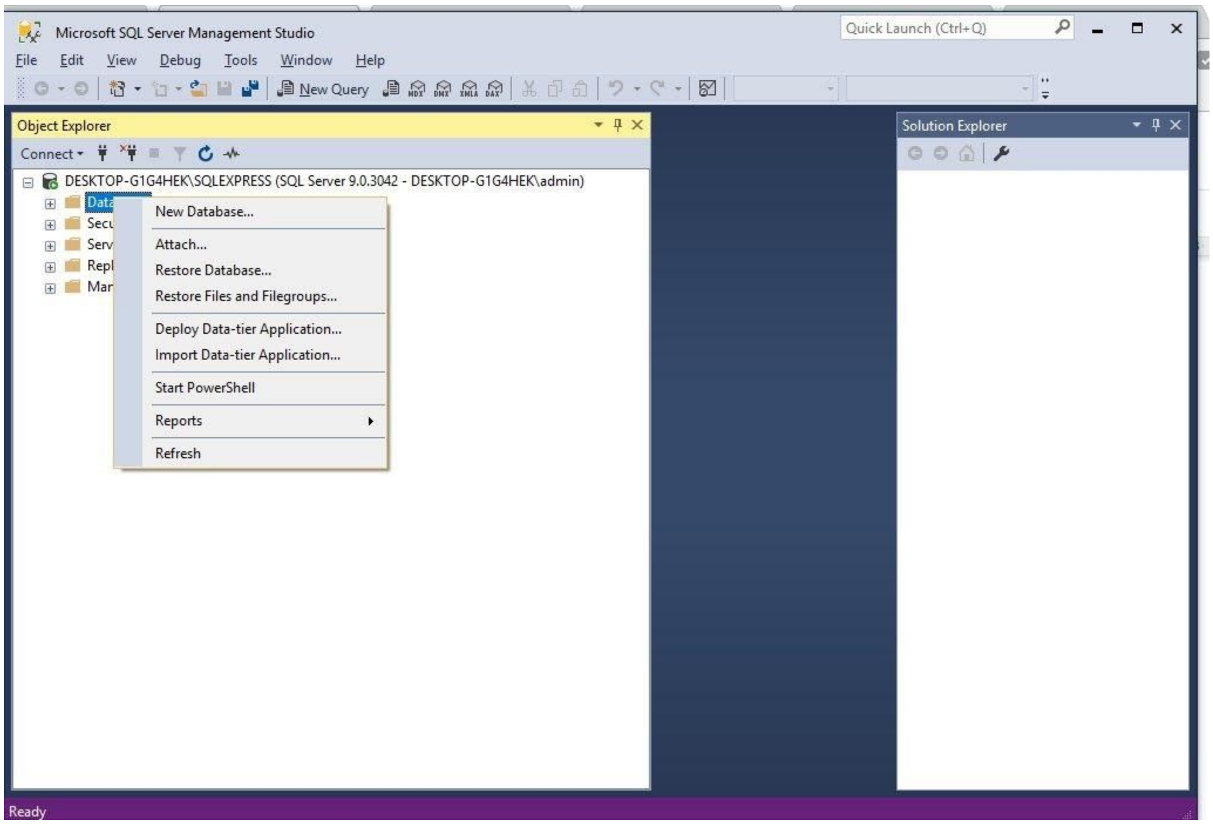
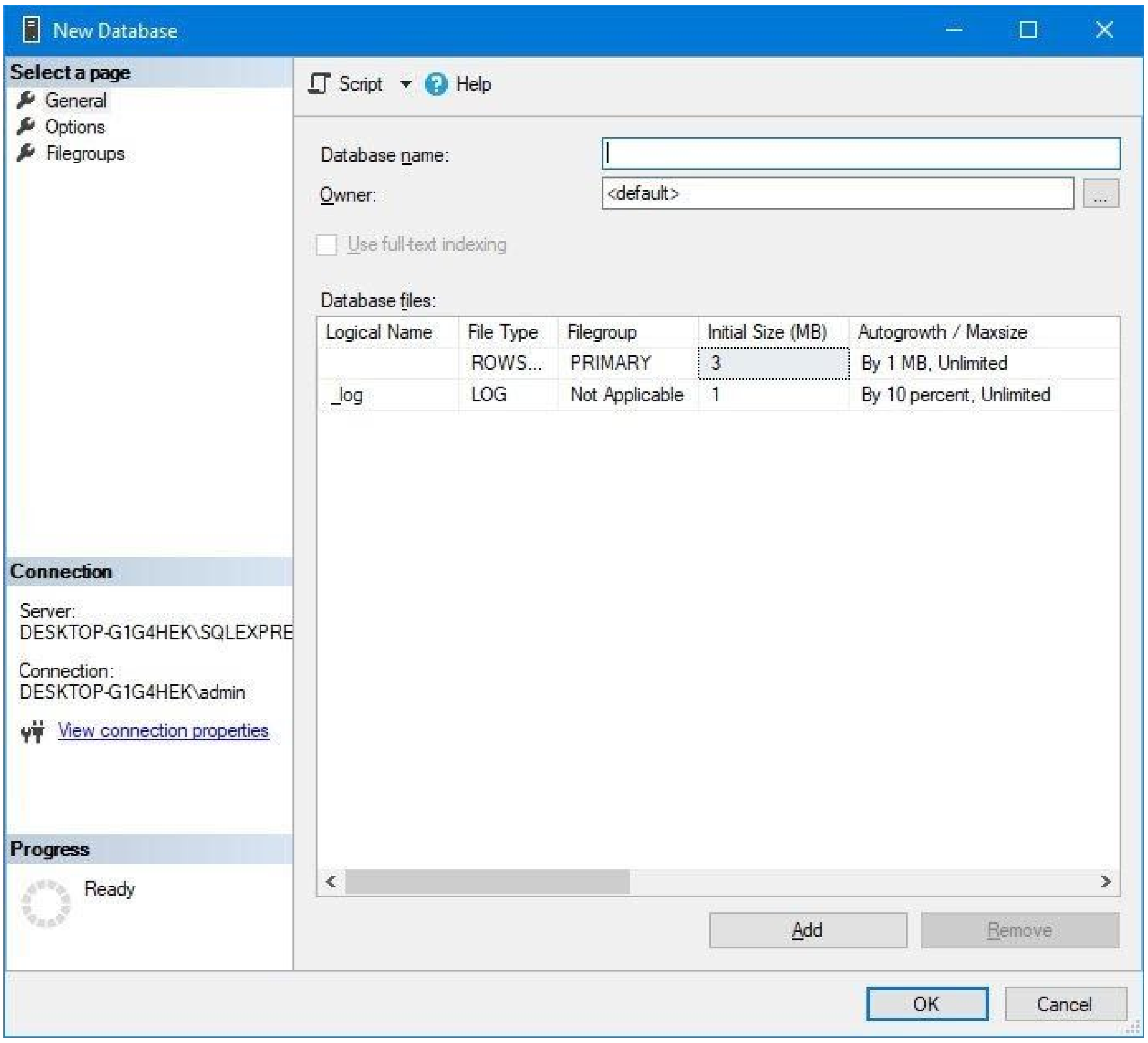

Microsoft SQL Server Management Studio
Quick Launch (Ctrl+Q)
File Edit View Debug Tools Window Help
New Query
Object Explorer
Connect
DESKTOP-G1G4HEK\SQLEXPRESS (SQL Server 9.0.3042 - DESKTOP-G1G4HEK\admin)
Data
Secu
Serv
Repl
Mar
New Database...
Attach...
Restore Database...
Restore Files and Filegroups...
Deploy Data-tier Application...
Import Data-tier Application...
Start PowerShell
Reports
Refresh
Solution Explorer
Ready

Figure 23.4

Clicking on New Database will bring out the New Database window, where you can create a new database by entering the database name as well as initial number of rows, as shown in Figure 23.5. In our example, we will create a database name known as customer, the database file is customer.mdf. After creating the database file, it will appear in SSMS, as shown in Figure 23.6



Database name:

Owner: <default>

Use full-text indexing

Database files:

Logical Name	File Type	Filegroup	Initial Size (MB)	Autogrowth / Maxsize
	ROWS...	PRIMARY	3	By 1 MB, Unlimited
_log	LOG	Not Applicable	1	By 10 percent, Unlimited

Connection

Server:
DESKTOP-G1G4HEK\SQLEXPRE

Connection:
DESKTOP-G1G4HEK\admin

Progress

Ready

Figure 23.5

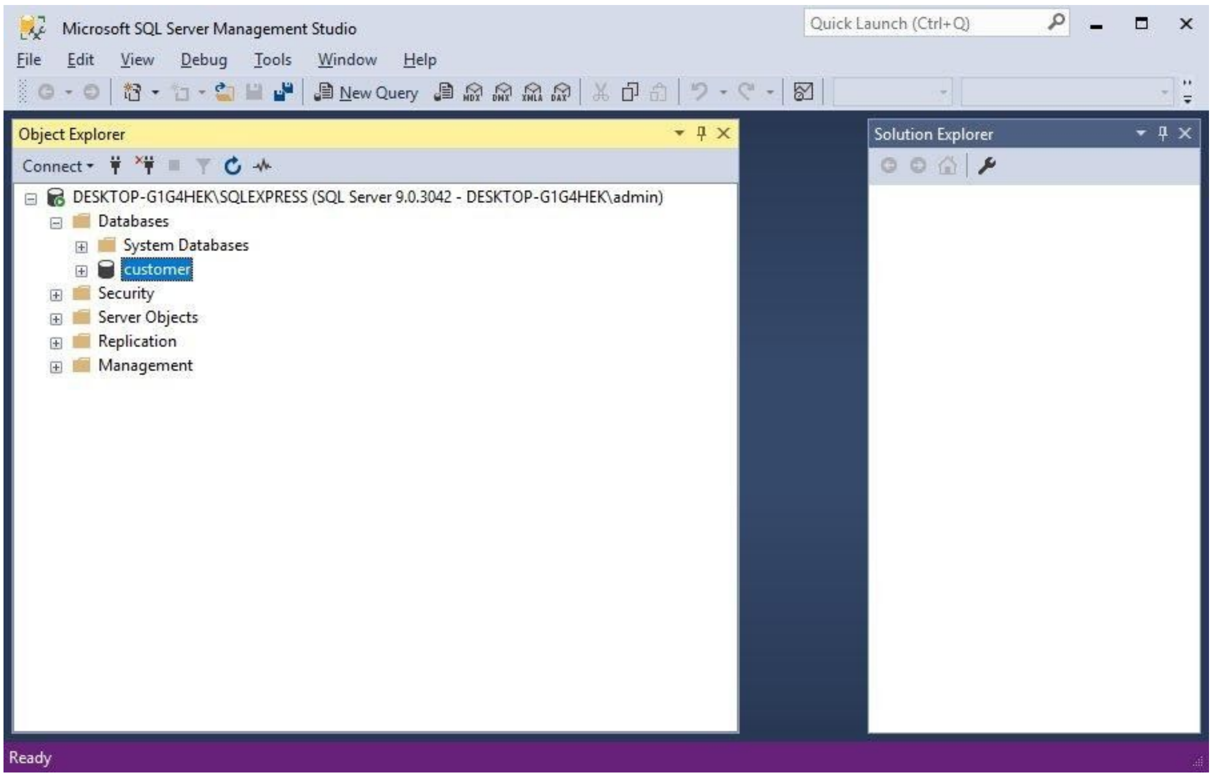
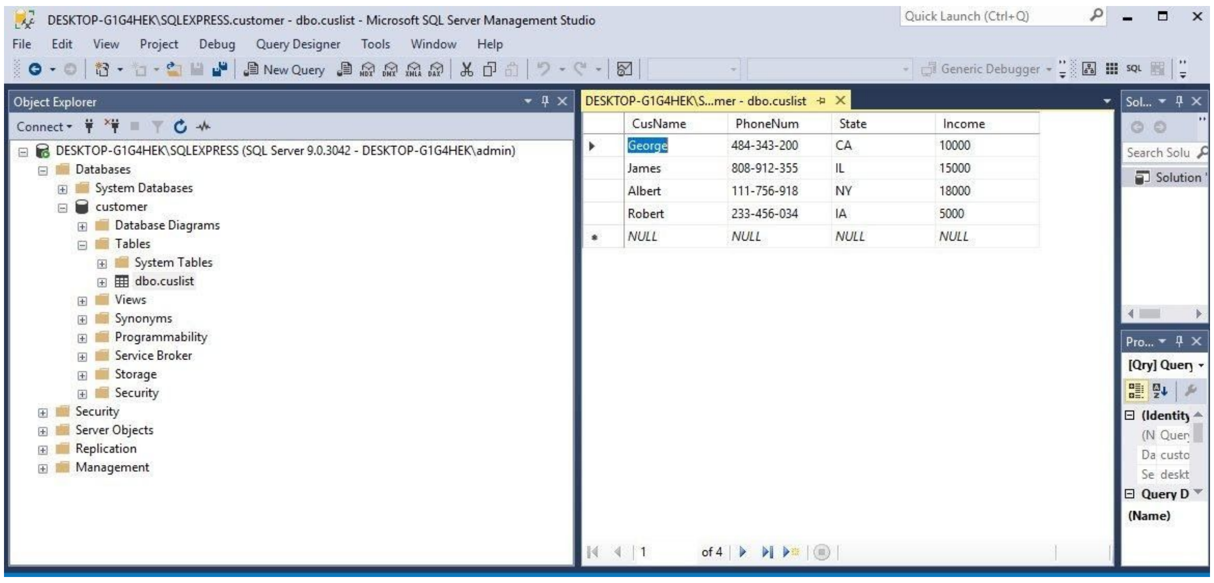


Figure 23.6

After creating the database, we need create table called cuslist. To create the table, expand the customer database and right click on table to bring up the table design window that allows you to key in the fields under column name and you can specify their data type such as string, numeric , money and so on. In our example, we create four fields, `CusName`, `PhoneNum`, `State` and `Income`. Save the table as cuslist.

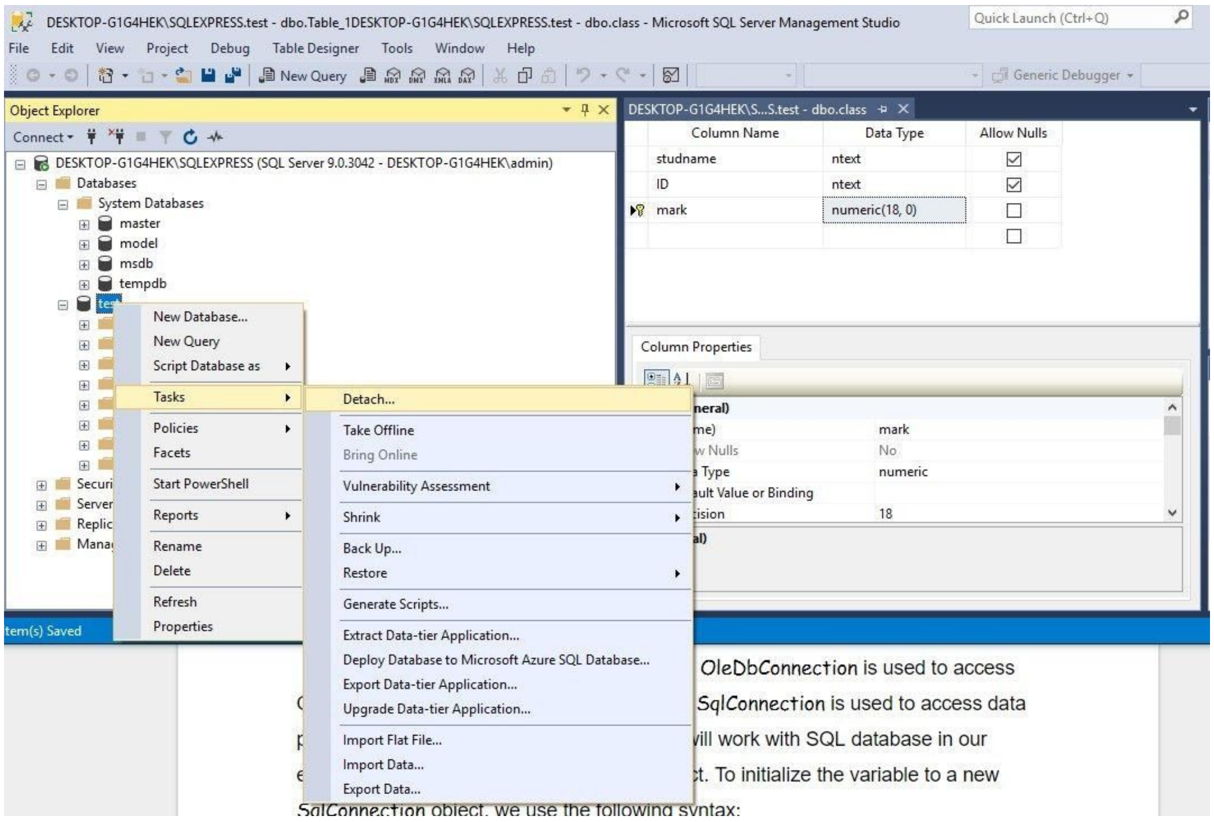
To enter data in the table, right click on the table name to bring a list of options, select Edit Top 200 Rows from the options to bring up the table for you to enter the data, as shown in Figure 23.8.


DESKTOP-G1G4HEK\SQLEXPRESS.customer - dbo.cuslist - Microsoft SQL Server Management Studio
File Edit View Project Debug Query Designer Tools Window Help
Object Explorer
DESKTOP-G1G4HEK\SQLEXPRESS (SQL Server 9.0.3042 - DESKTOP-G1G4HEK\admin)
Databases
System Databases
customer
Database Diagrams
Tables
System Tables
dbo.cuslist
Views
Synonyms
Programmability
Service Broker
Storage
Security
Security
Server Objects
Replication
Management
DESKTOP-G1G4HEK\S...mer - dbo.cuslist

CusName	PhoneNum	State	Income
George	484-343-200	CA	10000
James	808-912-355	IL	15000
Albert	111-756-918	NY	18000
Robert	233-456-034	IA	5000
NULL	NULL	NULL	NULL

Figure 23.8

Now we are ready to write the code to access the database we have created. You may need to detach the file from SQL Server Management Studio and copy the file to another location before you can access it from Visual Basic 2017. To detach a database file, right click on the file and select Task, then click o Detach, as shown in Figure 23.9.


DESKTOP-G1G4HEK\SQLEXPRESS.test - dbo.Table_1DESKTOP-G1G4HEK\SQLEXPRESS.test - dbo.class - Microsoft SQL Server Management Studio
Object Explorer
DESKTOP-G1G4HEK\SQLEXPRESS (SQL Server 9.0.3042 - DESKTOP-G1G4HEK\admin)
Databases
System Databases
master
model
msdb
tempdb
New Database...
New Query
Script Database as
Tasks
Policies
Facets
Start PowerShell
Reports
Rename
Delete
Refresh
Properties
Detach...
Take Offline
Bring Online
Vulnerability Assessment
Shrink
Back Up...
Restore
Generate Scripts...
Extract Data-tier Application...
Deploy Database to Microsoft Azure SQL Database...
Export Data-tier Application...
Upgrade Data-tier Application...
Import Flat File...
Import Data...
Export Data...

Column Name	Data Type	Allow Nulls
studname	ntext	☑
ID	ntext	☑
mark	numeric(18, 0)	☐
		☐

Figure 23.9

The ADO.NET object offers several connection objects such as **`OleDbConnection`**, **`SqlConnection`** and more. **`OleDbConnection`** is used to access OLEDB data such as Microsoft Access whilst **`SqlConnection`** is used to access data provided by Microsoft SQL server. Since we will work with SQL database in our example, we will use the **`SqlConnection`** object. To initialize the variable to a new **`SqlConnection`** object, we use the following syntax:

```
Private MyCn As New SqlConnection
```

Having created the instance of the **`SqlConnecton`** object, the next step is to establish a connection to the data source using the SQL **`ConnectionString`** property. The syntax is:

```
MyCn.ConnectionString ="Data Source=.\SQLEXPRESS;
AttachDbFilename=C:\Users\admin.DESKTOP-G1G4HEK\Documents\My
```

```
Websites\vbtutor\vb2017\customer.mdf; Integrated Security=True;
Connection Timeout=30;" & "User Instance=True"
```

* You need to change the reference to the SQL server (TOSHIBA-PC\SQL2017) as well as the path to database file Test.mdf to point to the database file in your own PC.

After establishing connection to the database, you can open the database using the following syntax:

```
MyCn.Open()
```

23.4 Populating Data in ADO.NET

Establishing connection to a database in Visual Basic 2017 using **`SqlConnection`** alone will not present anything tangible things to the user to manipulate the data until we add more relevant objects and write relevant codes to the project.

The next step is to create an instance of the **`SqlDataAdpater`** in our code so that we can populate the DataTable with data from the data source. Besides, you also need to create an instance of the **`DataTable`**. Other than that, you should also create an instance of the **`SqlCommandBuilder`** which is used to manipulate data such as updating and deleting data in the data table and send the changes back to the data source. The statements are:

```
Private MyDatAdp As New SqlDataAdapter
Private MyCmdBld As New SqlCommandBuilder
Private MyDataTbl As New DataTable
```

Besides that, we need to declare a variable to keep track of the user's current row within the data table. The statement is

```
Private MyRowPosition As Integer = 0
```

Having created the above of objects, you need to include the following statements in the Sub Form_Load event to start filling the DataTable with data from the data source. The statements are as follows:

```
MyDatAdp = New SqlDataAdapter("Select* from Contacts", MyCn)
MyCmdBld = New SqlCommandBuilder(MyDatAdp)
MyDatAdp.Fill(MyDataTbl)
```

After filling up the DataTable , we need to write code to access the data. To access data in the DataTable means that we need to access the rows in the table. We can achieve this by using the DataRow object. For example, we can write the following to access the first row of the table and present the data via two text boxes with the name txtName and txtState respectively:

```
Dim MyDataRow As DataRow = MyDataTbl.Rows(0)
Dim strName As String
Dim strState As String
strName = MyDataRow("ContactName")
strState = MyDataRow("State")
txtName.Text = strName.ToString
txtState.Text = strState.ToStringMe.showRecords()
```

* The two fields being referenced here are ContactName and State. Note Index 0 means first row.

showRecords() is a sub procedure created to show data in the text boxes. The code is as follows:

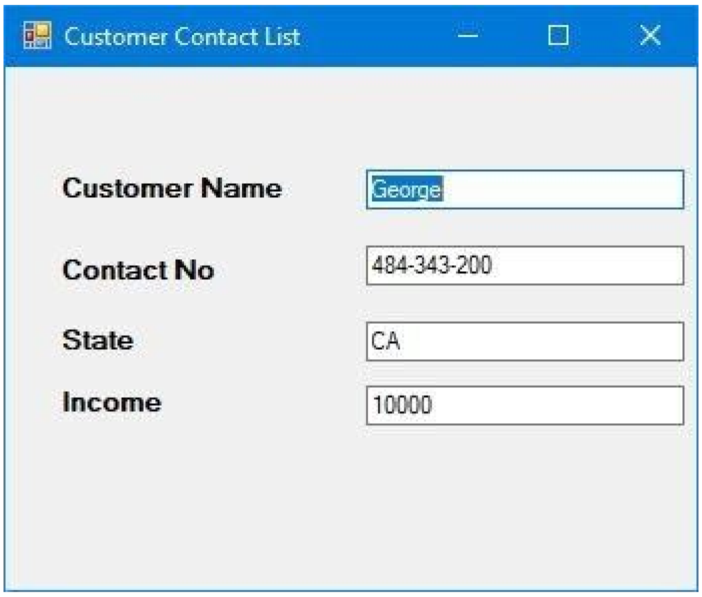
```
Private Sub showRecords()
If MyDataTbl.Rows.Count = 0 Then
txtName.Text = ""
txtState.Text = ""
Exit Sub
End If
txtName.Text = MyDataTbl.Rows(MyRowPosition)("ContactName").ToString
txtState.Text = MyDataTbl.Rows(MyRowPosition)("State").ToString
End Sub
```

The full Code is shown in Example 23.1

Example 23.1 Creating a Database

```
Private MyDatAdp As New SqlDataAdapter
Private MyCmdBld As New SqlCommandBuilder
Private MyDataTbl As New DataTable
Private MyCn As New SqlConnection
Private MyRowPosition As Integer = 0
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
MyCn.ConnectionString ="Data Source=.\SQLEXPRESS;
AttachDbFilename=C:\Users\Documents\vb2017\customer.mdf; Integrated
Security=True; Connection Timeout=30;" & "User Instance=True"
MyCn.Open()
MyDatAdp = New SqlDataAdapter("Select* from Contacts", MyCn)
MyCmdBld = New SqlCommandBuilder(MyDatAdp)
MyDatAdp.Fill(MyDataTbl)
Dim MyDataRow As DataRow = MyDataTbl.Rows(0)
Dim strName As String
        Dim strState As String
        Dim strPhone As String
        Dim strIncome As Double
        strName = MyDataRow("CusName")
        strPhone = MyDataRow("PhoneNum")
        strState = MyDataRow("State")
        strIncome = MyDataRow("Income")
        TxtCusName.Text = strName.ToString()
        TxtContact.Text = strPhone.ToString()
        TxtState.Text = strState.ToString()
        TxtIncome.Text = strIncome.ToString()
        Me.showRecords()

End Sub

  If MyDataTbl.Rows.Count = 0 Then
            TxtCusName.Text = ""
            TxtContact.Text = ""
            TxtState.Text = ""
            TxtIncome.Text = ""
            Exit Sub
        End If
TxtCusName.Text = MyDataTbl.Rows(MyRowPosition)("CusName").ToString()
        TxtContact.Text =
MyDataTbl.Rows(MyRowPosition)("PhoneNum").ToString()
        TxtState.Text = MyDataTbl.Rows(MyRowPosition)("State").ToString()
        TxtIncome.Text =
MyDataTbl.Rows(MyRowPosition)("Income").ToString()

End Sub
```

The output interface is shown in Figure 23.10



Customer Name	George
Contact No	484-343-200
State	CA
Income	10000

Figure 23.10

23.5 Browsing Records

In previous section, we have learned how to display the first record using the showRecords sub procedure. In this chapter, we will create command buttons and write relevant codes to allow the user to browse the records forward and backward as well as fast forward to the last record and back to the first record. The first button we need to create is for the user to browse the first record. We can use button's text **<<** to indicate to the user that it is the button to move to the first record and button's text **>>** to move to the last record. Besides we can use button's text **<** for moving to previous record and button's text > for moving to next record.

The code for moving to the first record is

```
MyRowPosition = 0

Me.showRecords()
```

The code for moving to previous record is

```
If MyRowPosition > 0 Then
MyRowPosition = MyRowPosition - 1
Me.showRecords()
End If
```

The code for moving to next record is

```
If MyRowPosition < (MyDataTbl.Rows.Count - 1) Then
MyRowPosition = MyRowPosition + 1
Me.showRecords()
End If
```

The code for moving to last record is

```
If MyDataTbl.Rows.Count > 0 Then

MyRowPosition = MyDataTbl.Rows.Count - 1

Me.showRecords()

End If
```

23.6 Editing, Saving, Adding and Deleting Records

You can edit any record by navigating to the record and change the data values. However, you need to save the data after editing them. You need to use the update method of the SqlDataAdapter to save the data. The code is:

```
If MyDataTbl.Rows.Count <> 0 Then
MyDataTbl.Rows(MyRowPosition)("ContactName") = txtName.Text
MyDataTbl.Rows(MyRowPosition)("state") = txtState.Text
MyDatAdp.Update(MyDataTbl)
End If
```

You can also add new record or new row to the table using the following code:

```
Dim MyNewRow As DataRow = MyDataTbl.NewRow()
MyDataTbl.Rows.Add(MyNewRow)
MyRowPosition = MyDataTbl.Rows.Count - 1
Me.showRecords()
```

The code above will present a new record with blank fields for the user to enter the new data. After entering the data, he or she can then click the save button to save the data.

Lastly, the user might want to delete the data. Remember to add a primary key to one of the fields in the database otherwise it will not work. The code to delete the data is:

```
If MyDataTbl.Rows.Count <> 0 Then
MyDataTbl.Rows(MyRowPosition).Delete()
MyDatAdp.Update(MyDataTbl)
MyRowPosition = 0
Me.showRecords()
End If
```

The complete code is shown in Example 23.2

Example 23.2 Browsing Records

```
Public Class Form1
Private MyDatAdp As New SqlDataAdapter
Private MyCmdBld As New SqlCommandBuilder
Private MyDataTbl As New DataTable
Private MyCn As New SqlConnection
Private MyRowPosition As Integer = 0
Private Sub Form1_FormClosed(sender As Object, e As FormClosedEventArgs)
Handles Me.FormClosed
MyCn.Close()
MyCn.Dispose()
End Sub
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
MyCn.ConnectionString = "Data Source=.\SQLEXPRESS;
AttachDbFilename=C:\Users\Documents\vb2017\customer.mdf; Integrated
Security=True; Connection Timeout=30;" & "User Instance=True"

MyCn.Open()
MyDatAdp = New SqlDataAdapter("Select* from cuslist", MyCn)
        MyCmdBld = New SqlCommandBuilder(MyDatAdp)
        MyDatAdp.Fill(MyDataTbl)
        Dim MyDataRow As DataRow = MyDataTbl.Rows(0)
        Dim strName As String
        Dim strState As String
        Dim strPhone As String
        Dim strIncome As Double
        strName = MyDataRow("CusName")
        strPhone = MyDataRow("PhoneNum")
        strState = MyDataRow("State")
        strIncome = MyDataRow("Income")
        TxtCusName.Text = strName.ToString()
        TxtContact.Text = strPhone.ToString()
        TxtState.Text = strState.ToString()
        TxtIncome.Text = strIncome.ToString()
        Me.showRecords()
    End Sub


Private Sub showRecords()
If MyDataTbl.Rows.Count = 0 Then
            TxtCusName.Text = ""
            TxtContact.Text = ""
            TxtState.Text = ""
            TxtIncome.Text = ""
            Exit Sub
        End If
TxtCusName.Text = MyDataTbl.Rows(MyRowPosition)("CusName").ToString()
TxtContact.Text = MyDataTbl.Rows(MyRowPosition)("PhoneNum").ToString()
```

```
TxtState.Text = MyDataTbl.Rows(MyRowPosition)("State").ToString()
TxtIncome.Text = MyDataTbl.Rows(MyRowPosition)("Income").ToString()
End Sub

Private Sub BtnMoveFirst_Click(sender As Object, e As EventArgs) Handles
BtnMoveFirst.Click
MyRowPosition = 0
Me.showRecords()
End Sub

Private Sub BtnMovePrev_Click(sender As Object, e As EventArgs) Handles
BtnMovePrev.Click
If MyRowPosition > 0 Then
MyRowPosition = MyRowPosition - 1
Me.showRecords()
End If
End Sub
Private Sub BtnMoveNext_Click(sender As Object, e As EventArgs) Handles
BtnMoveNext.Click
If MyRowPosition < (MyDataTbl.Rows.Count - 1) Then
MyRowPosition = MyRowPosition + 1
Me.showRecords()
End If
End Sub

Private Sub BtnMoveLast_Click(sender As Object, e As EventArgs) Handles
BtnMoveLast.Click
If MyDataTbl.Rows.Count > 0 Then
MyRowPosition = MyDataTbl.Rows.Count - 1
Me.showRecords()
End If
End Sub


Private Sub BtnAdd_Click(sender As Object, e As EventArgs) Handles
BtnAdd.Click

Dim MyNewRow As DataRow = MyDataTbl.NewRow()
MyDataTbl.Rows.Add(MyNewRow)
MyRowPosition = MyDataTbl.Rows.Count - 1
Me.showRecords()
End Sub
```

```
Private Sub BtnDelete_Click(ByVal sender As Object, ByVal e As EventArgs)
Handles BtnDelete.Click
If MyDataTbl.Rows.Count <> 0 Then
MyDataTbl.Rows(MyRowPosition).Delete()
MyRowPosition = 0
MyDatAdp.Update(MyDataTbl)
Me.showRecords()
End If
End Sub

Private Sub BtnSave_Click(ByVal sender As Object, ByVal e As EventArgs)
Handles BtnSave.Click
 If MyDataTbl.Rows.Count <> 0 Then
 MyDataTbl.Rows(MyRowPosition)("CusName") = TxtCusName.Text
 MyDataTbl.Rows(MyRowPosition)("PhoneNum") = TxtContact.Text
 MyDataTbl.Rows(MyRowPosition)("state") = TxtState.Text
 MyDataTbl.Rows(MyRowPosition)("Income") = TxtIncome.Text
 MyDatAdp.Update(MyDataTbl)
 End If
End Sub

End Class
```

The output interface is as shown in Figure 23.11

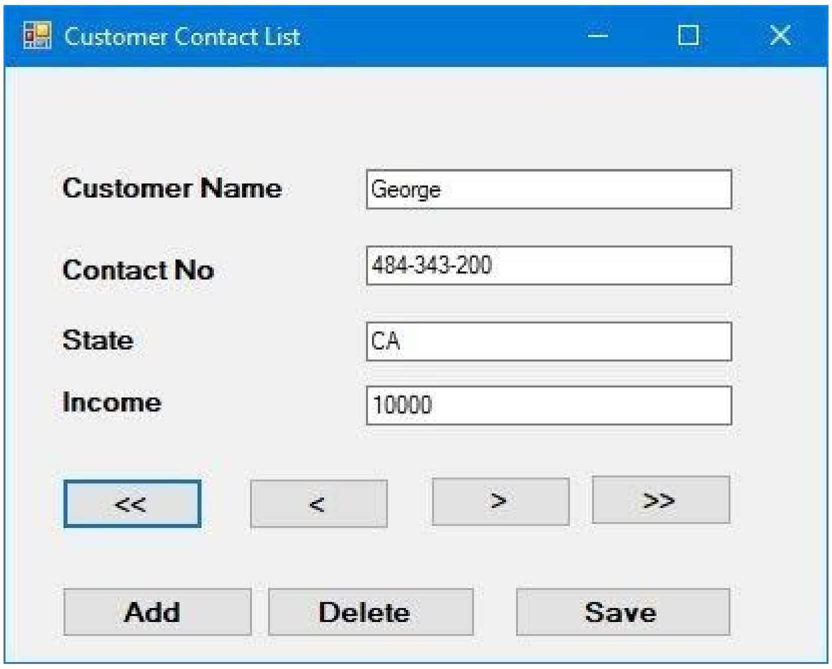


Figure 23.11

23.7 Accessing Database using DataGridView

Another method to access a database is to use the DataGridView control. DataGridView allows the user to browse the data in a database via a table that comprises rows and columns. To access the database, use the following statement:

```
Private Datastr As String = "Data Source=.\SQLEXPRESS;
AttachDbFilename=C:\Users\Documents\vb2017\customer.mdf; Integrated
Security=True; Connection Timeout=30;" & "User Instance=True"
```

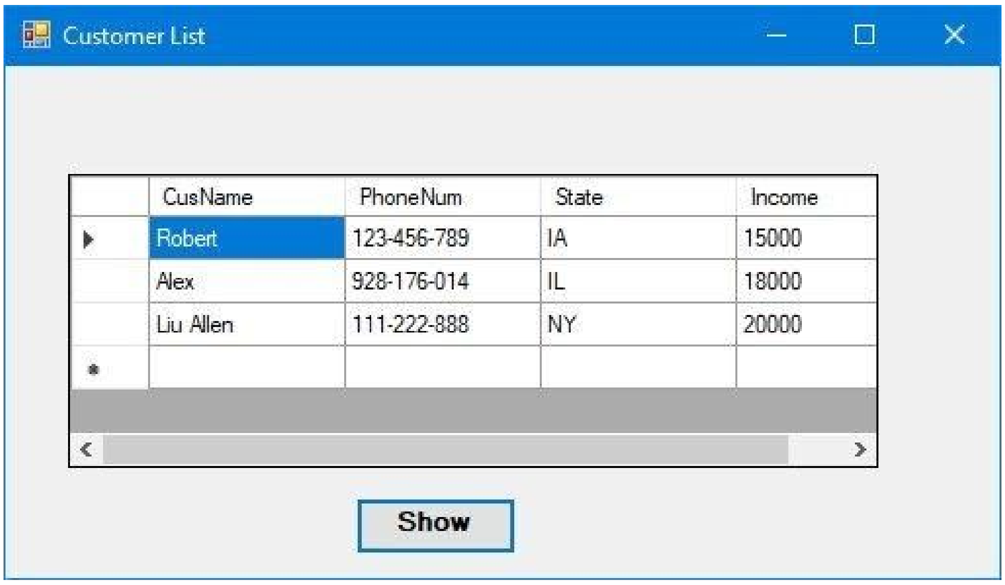
To use the DataGridView control, drag it from the toolbox and insert it into the form. Next, insert a button and rename it as BtnShow. Click the button and enter the following code, as shown in Example 23.3

Example 23.3 Browsing Data Using DataGridView

```
Public Class Form1
Private Datastr As String = "Data Source=.\SQLEXPRESS;
AttachDbFilename=C:\Users\Documents\vb2017\customer.mdf; Integrated
```

```
Security=True; Connection Timeout=30;" & "User Instance=True"
Private Sub BtnShow_Click(sender As Object, e As EventArgs) Handles
BtnShow.Click
 Dim Mycon As New SqlConnection(Datastr)
 Dim MySelect As String = "Select CusName, PhoneNum,State, Income from
cuslist"
Dim MyAdpt As New SqlDataAdapter(MySelect, Mycon)
Dim Myds As New DataSet()
MyAdpt.Fill(Myds, "cuslist")
DataGridView1.DataSource = Myds.Tables(0)
End Sub
End Class
```

The output is shown in Figure 23.12



CusName	PhoneNum	State	Income
Robert	123-456-789	IA	15000
Alex	928-176-014	IL	18000
Liu Allen	111-222-888	NY	20000

Figure 23.12

23.8 Performing Arithmetic Calculations in a Database

In Visual Basic 2017, we can retrieve data from a database and perform arithmetic calculations. Let us create a simple database with two columns, name and income and save it as family.mdf in SQL server. For income, we need to specify it as money or numeric data type to enable calculation. We use money in our example.

In our example, we wish to calculate the sum of income for all the data. To perform this calculation, we use SQL keywords, as follows:

```
Dim mySelectQuery As String = "Select SUM (Income) FROM mylist;"
```

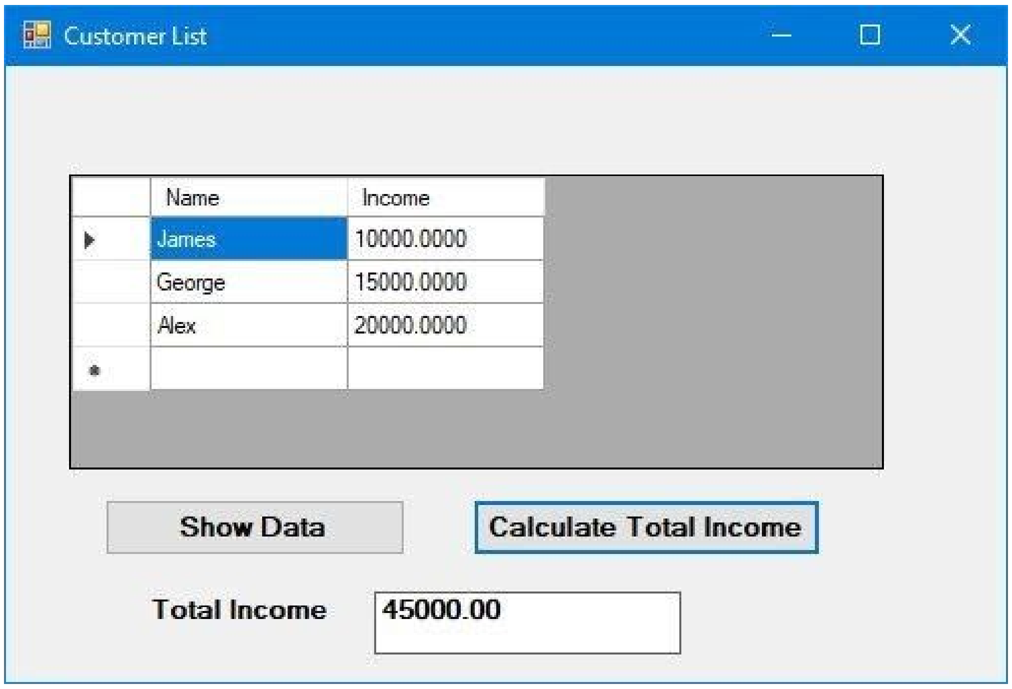
We shall show modify the code in Example 23.3, but we add a button so that we can show the data in a data grid and calculate the total income, as shown in Example 23.4.

Example 23.4 Performing Arithmetic Calculation

```
Public Class Form1
Private Datastr As String = "Data Source=.\SQLEXPRESS;
AttachDbFilename=C:\Users\Documents\vb2017\family.mdf; Integrated
Security=True; Connection Timeout=30;" & "User Instance=True"
   Private searchterm As String
   Private Sub BtnShow_Click(sender As Object, e As EventArgs) Handles
BtnShow.Click
Dim Mycon As New SqlConnection(Datastr)
Dim MySelect As String = "Select Name, Income from mylist;"
Dim Adpt As New SqlDataAdapter(MySelect, Mycon)
Dim Myds As New DataSet()
    Adpt.Fill(Myds, "mylist")
    DataGridView1.DataSource = Myds.Tables(0)
End Sub



'Sub procedure to calculate the sum of income
Public Sub totalIncome(ByVal myConnString As String, ByVal searchValue As
String)
Dim MyDatApt As New SqlDataAdapter
Dim Myds As New DataSet
Dim mySelectQuery As String = "Select SUM (Income) FROM mylist;"
Dim myConn As New SqlConnection(myConnString)
Dim myCommand As New SqlCommand(mySelectQuery, myConn)
myConn.Open()
MyDatApt.SelectCommand = myCommand
LblTotal.Text = Convert.ToString(Math.Round(myCommand.ExecuteScalar, 2))
myConn.Dispose()     'To dispose the object to release more memory
End Sub
Private Sub BtnCalInc_Click(sender As Object, e As EventArgs) Handles
```

```
BtnCalInc.Click
totalIncome(Datastr, searchterm)
End Sub
End Class
```

The Output



	Name	Income
▶	James	10000.0000
	George	15000.0000
	Alex	20000.0000
*		

Total Income 45000.00

Figure 23.13

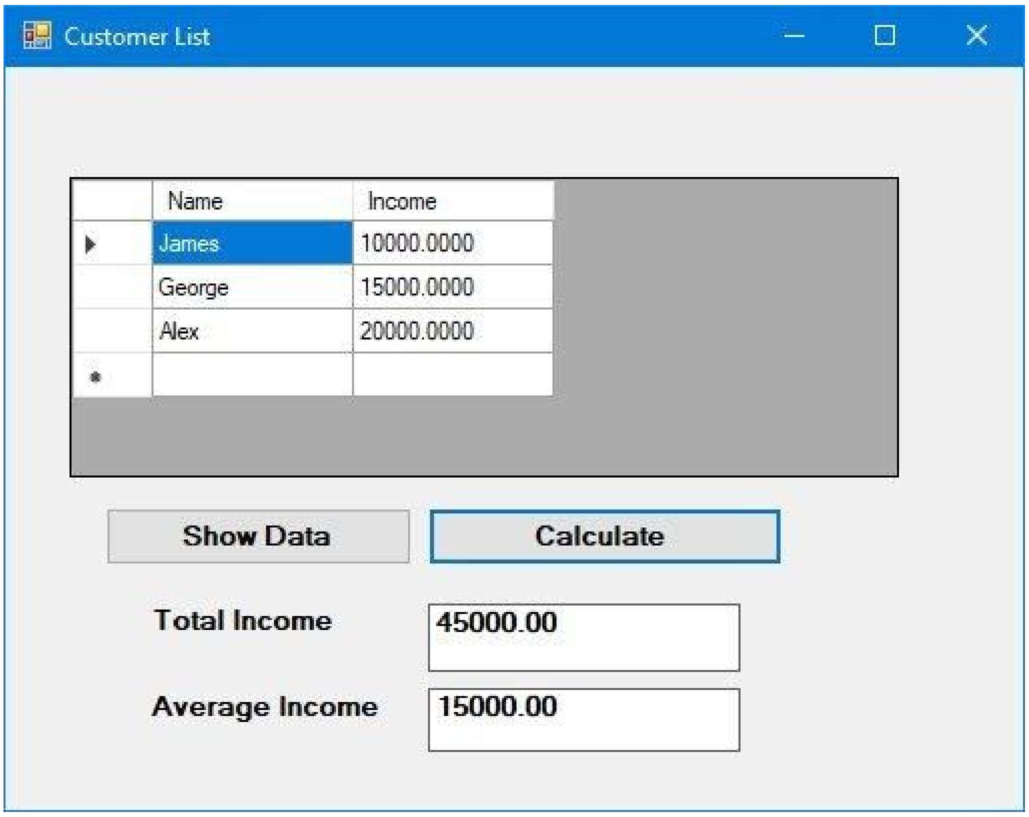
In the next example, we shall modify the code in Example 23.4 so that we can calculate the average income as well. In this example, we add a few more lines of code and a label to display the average income. The code is shown in Example 23.5

Example 23.5 Calculating Average

```
Private Datastr As String = "Data Source=.\SQLEXPRESS;
AttachDbFilename=C:\Users\Documents\vb2017\family.mdf; Integrated
Security=True; Connection Timeout=30;" & "User Instance=True"
    Private searchterm As String
 Private Sub BtnShow_Click(sender As Object, e As EventArgs) Handles
BtnShow.Click
```

```
        Dim Mycon As New SqlConnection(Datastr)
        Dim MySelect As String = "Select Name, Income from mylist"
        Dim Adpt As New SqlDataAdapter(MySelect, Mycon)
        Dim Myds As New DataSet()
        Adpt.Fill(Myds, "mylist")
        DataGridView1.DataSource = Myds.Tables(0)
    End Sub
 Public Sub Income(ByVal myConnString As String, ByVal searchValue As
String)
        Dim MyDatApt As New SqlDataAdapter
        Dim Myds As New DataSet
        Dim mySelectQuery As String = "Select SUM (Income) FROM mylist;"
        Dim mySelectQuery2 As String = "Select AVG (Income) FROM mylist;"
        Dim myConn As New SqlConnection(myConnString)
        Dim myCommand As New SqlCommand(mySelectQuery, myConn)
        Dim myCommand2 As New SqlCommand(mySelectQuery2, myConn)
        myConn.Open()
        MyDatApt.SelectCommand = myCommand
        MyDatApt.SelectCommand = myCommand2
LblTotal.Text = Convert.ToString(Math.Round(myCommand.ExecuteScalar, 2))
LblAvg.Text = Convert.ToString(Math.Round(myCommand2.ExecuteScalar, 2))
        myConn.Dispose()
    End Sub
Private Sub BtnCalInc_Click(sender As Object, e As EventArgs) Handles
BtnCalInc.Click
        Income(Datastr, searchterm)
End Sub
```

The output is shown in Figure 23.14



	Name	Income
▸	James	10000.0000
	George	15000.0000
	Alex	20000.0000
*		

Total Income 45000.00

Average Income 15000.00

Figure 23.14

Besides the SUM and AVG Sql Functions, there is also a COUNT function. The syntax is

```
SELECT COUNT(column_name) FROM table_name WHERE condition;
```

We shall show how it works in Example 23.6

Example 23.6 Using SQL Count Function

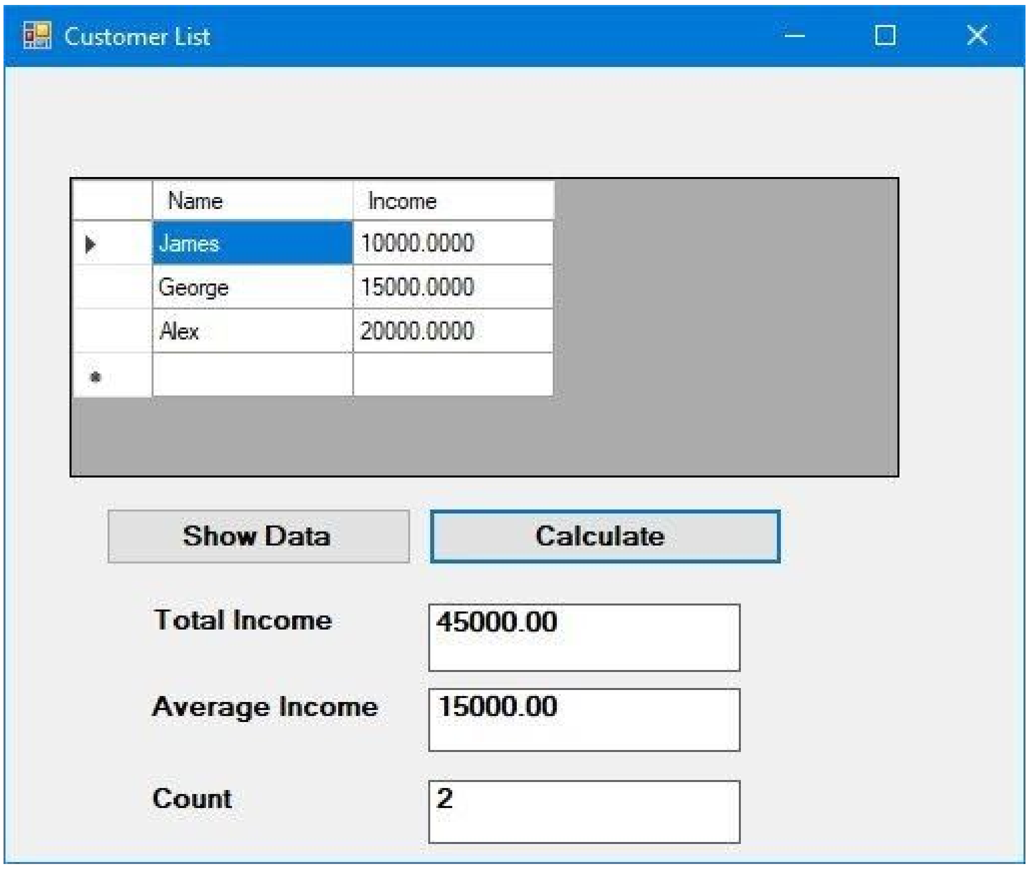
In this example, we add another label to show the count value. We impose a condition where count is executed for income <20000 with the following statement:

```
Dim mySelectQuery3 As String = "Select COUNT (Income) FROM mylist WHERE
Income<20000;"
```

Modify this part of the code in Example 23.5, as follows:

```
Public Sub Income(ByVal myConnString As String, ByVal searchValue As
String)
Dim MyDatApt As New SqlDataAdapter
Dim Myds As New DataSet
Dim mySelectQuery As String = "Select SUM (Income) FROM mylist;"
Dim mySelectQuery2 As String = "Select AVG (Income) FROM mylist;"
Dim mySelectQuery3 As String = "Select COUNT (Income) FROM mylist WHERE
Income<20000;"
        Dim myConn As New SqlConnection(myConnString)
        Dim myCommand As New SqlCommand(mySelectQuery, myConn)
        Dim myCommand2 As New SqlCommand(mySelectQuery2, myConn)
        Dim myCommand3 As New SqlCommand(mySelectQuery3, myConn)
        myConn.Open()
        MyDatApt.SelectCommand = myCommand
        MyDatApt.SelectCommand = myCommand2
LblTotal.Text=Convert.ToString(Math.Round(myCommand.ExecuteScalar, 2))
LblAvg.Text =Convert.ToString(Math.Round(myCommand2.ExecuteScalar, 2))
LblCount.Text =Convert.ToString(Math.Round(myCommand3.ExecuteScalar, 2))
        myConn.Dispose()
    End Sub
```

The output is shown in Figure 23.15



	Name	Income
▶	James	10000.0000
	George	15000.0000
	Alex	20000.0000
*		

Total Income	45000.00
Average Income	15000.00
Count	2

Figure 23.15

Summary

- In section 23.1, you have understood the concepts of database
- In section 23.2, you have learned how to create a database
- In section 23.3, you have learned how to create connection to a database
- In section 23.4, you have learned how to populate a database
- In section 23.5, you have learned how to write code for browsing records
- In section 23.5, you have learned how to write code for editing, saving, adding and deleting records in a database

Chapter 24 Reading and Writing Text Files

- ❖ Learn how to create program to read text files
- ❖ Learn how to create program to Write text files

24.1 Introduction

To be able to open a file and read the data from storage unit of a computer, such as a hard drive as well as able to save the data into the storage unit are important functions of a computer program. In fact, the ability to store, retrieve and modify data makes a computer a powerful tool in database management.

In this Chapter, we will learn how to create a text file that can store date. Using text file is an easy way to manage data, although it is not as sophisticated as full-fledged database management software such as SQL Server, Microsoft Access and Oracle. Visual Basic 2017 allows the user to create a text file, save the text file as well as to read the text file. It is relatively easy to write code for the above purposes in VB2017.

Reading and writing to a text file in VB2017 required the use of the StreamReader class and the StreamWriter class respectively. StreamReader is a tool that enables the streaming of data by moving it from one location to another so that the user can read it. For example, it allows the user to read a text file that is stored in a hard drive. On the other hand, the StreamWriter class is a tool that can write data input by the user to a storage device such as the hard drive.

24.2 Reading a Text File

In order to read a file from the hard disk or any storage device, we need to use the StreamReader class. To achieve that, first we need to include the following statement in the program code:

```
Imports System.IO
```

This line must precede the whole program code as it is higher in hierarchy than the StreamReader Class. In Fact, this is the concept of object-oriented programming where StreamReader is part of the namespace System.IO. You have to put it on top of the whole program (i.e. above the Public Class Form 1 statement). The word import means we import the namesapce System.IO into the program. Once we have done that, we can declare a variable of the streamReader data type with the following statement:

```
Dim FileReader As StreamReader
```

If we do not include the Imports System.IO, we must use the statement

```
Dim FileReader As IO.StreamReader
```

each time we want to use the StreamReader class.

Now, start a new project and name it in whatever name you wish, we named it TxtEditor here. Now, insert the `OpenFileDialog` control into the form because we will use it to read the file from the storage device. We also need to declare a new OpenFileDialog object before we can use it, you can name it as **`OpenFileDialog1`**. The **`OpenFileDialog`** control will return a DialogResult value that can determine whether the user clicks the OK button or Cancel button. We will also insert a command button and change its displayed text to 'Open'. The user can use it to open and read a specific text file. The following statement will accomplish the task above.

```
Dim results As DialogResult
          Dim OpenFileDialog1 As New OpenFileDialog()
results = OpenFileDialog1.ShowDialog
If results = DialogResult.OK Then
'Code to be executed if OK button was clicked
Else
'Code to be executed if Cancel button was clicked
End If
```

Next, we insert a textbox ,name it TxtEditor and set its Multiline property to true. It is used for displaying the text from a text file. We also insert a button and name it BtnOpen. In order to read the text file, we need to create a new instant of the streamReader and connect it to a text file with the following statement:

```
FileReader = New StreamReader(OpenFileDialog1.FileName)
```

In addition, we need to use the ReadToEnd method to read the entire text of a text file and display it in the text box. The syntax is:

```
TxtEditor.Text = FileReader.ReadToEnd( )
```

Lastly, we need to close the file by using the Close() method. The entire code is shown in the box below:

The Code

```
Imports System.IO
Public Class Form1
Private Sub BtnOpen_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnOpen.Click
Dim FileReader As StreamReader
Dim results As DialogResult
Dim OpenFileDialog1 As New OpenFileDialog()
results = OpenFileDialog1.ShowDialog
If results = DialogResult.OK Then
FileReader = New StreamReader(OpenFileDialog1.FileName)
TxtEditor.Text = FileReader.ReadToEnd()
FileReader.Close()
End If
End Sub
```

The Design Interface is shown in Figure 24.1

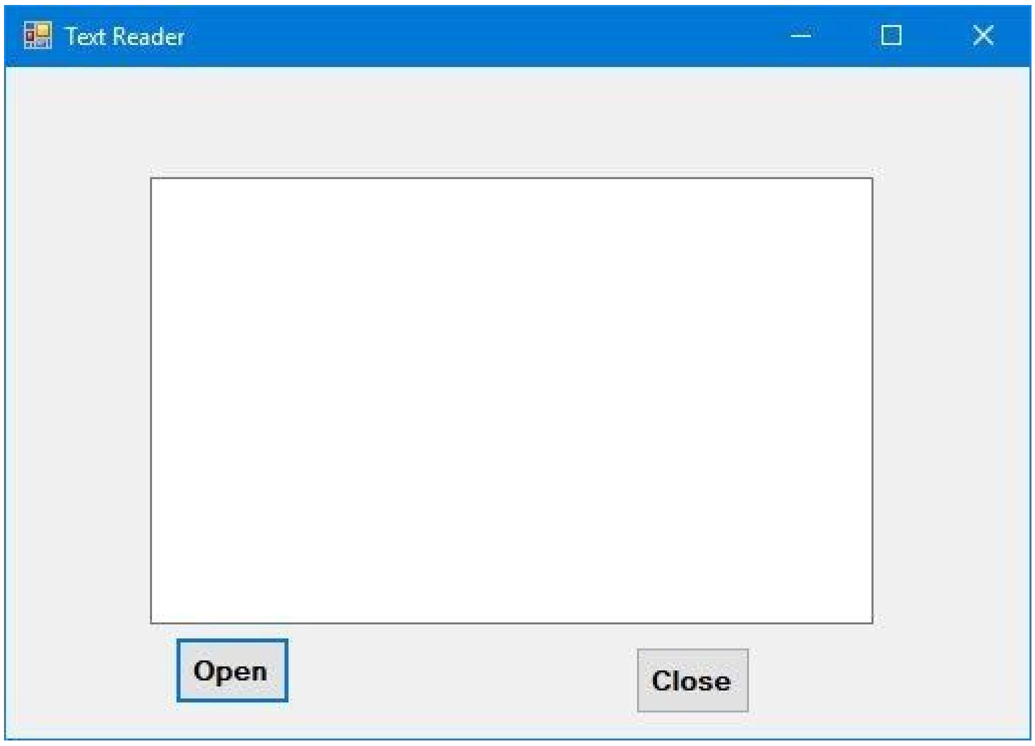
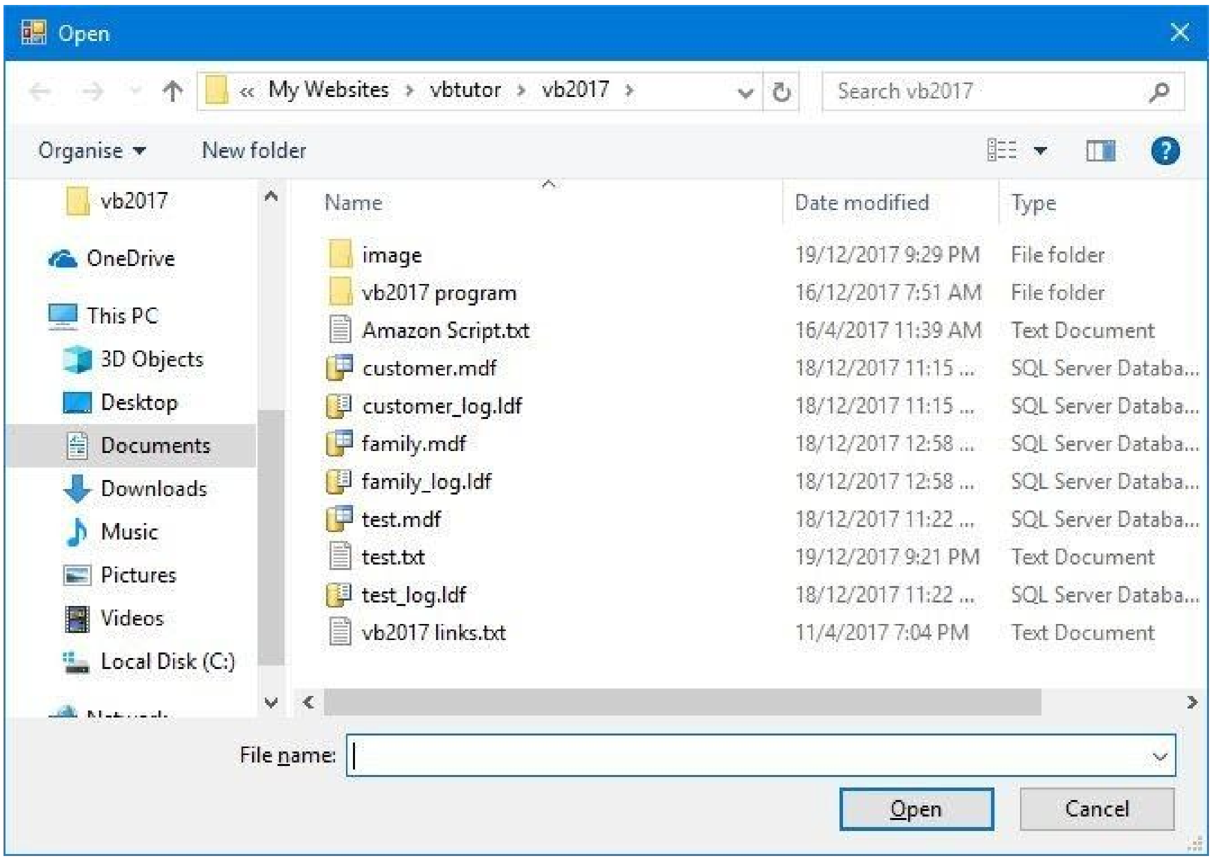


Figure 24.1

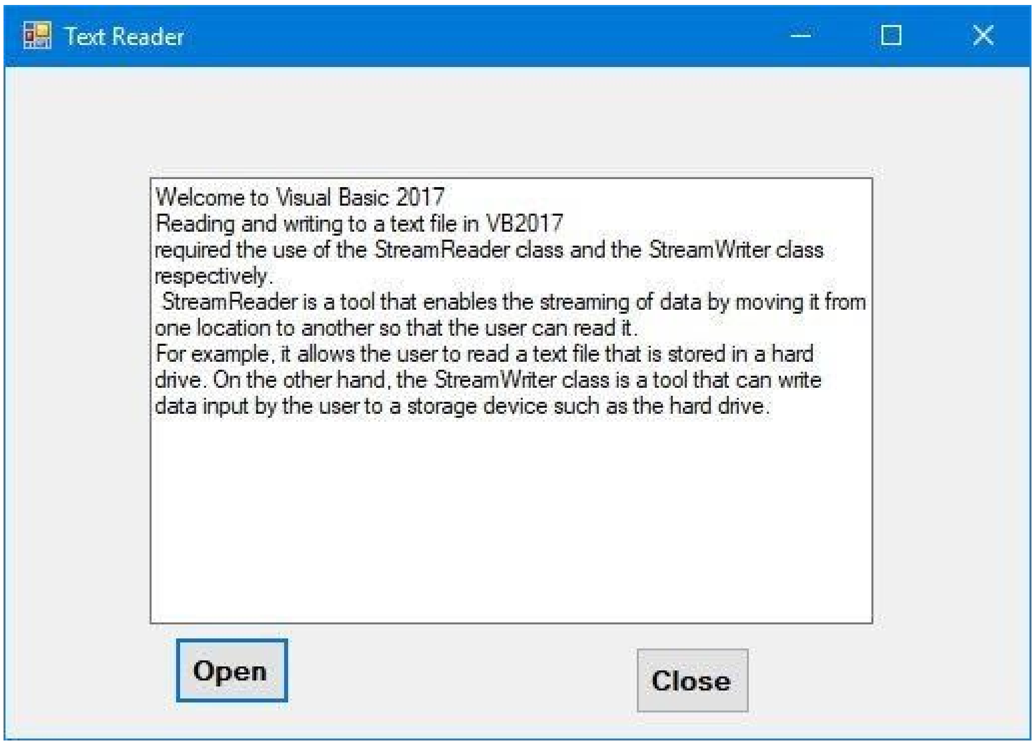
The Open Dialog box is shown in Figure 24.2



Name	Date modified	Type
image	19/12/2017 9:29 PM	File folder
vb2017 program	16/12/2017 7:51 AM	File folder
Amazon Script.txt	16/4/2017 11:39 AM	Text Document
customer.mdf	18/12/2017 11:15 ...	SQL Server Databa...
customer_log.ldf	18/12/2017 11:15 ...	SQL Server Databa...
family.mdf	18/12/2017 12:58 ...	SQL Server Databa...
family_log.ldf	18/12/2017 12:58 ...	SQL Server Databa...
test.mdf	18/12/2017 11:22 ...	SQL Server Databa...
test.txt	19/12/2017 9:21 PM	Text Document
test_log.ldf	18/12/2017 11:22 ...	SQL Server Databa...
vb2017 links.txt	11/4/2017 7:04 PM	Text Document

Figure 24.2

The Output Interface is shown in Figure 24.3



```
Welcome to Visual Basic 2017
Reading and writing to a text file in VB2017
required the use of the StreamReader class and the StreamWriter class
respectively.
 StreamReader is a tool that enables the streaming of data by moving it from
one location to another so that the user can read it.
For example, it allows the user to read a text file that is stored in a hard
drive. On the other hand, the StreamWriter class is a tool that can write
data input by the user to a storage device such as the hard drive.
```

Figure 24.3

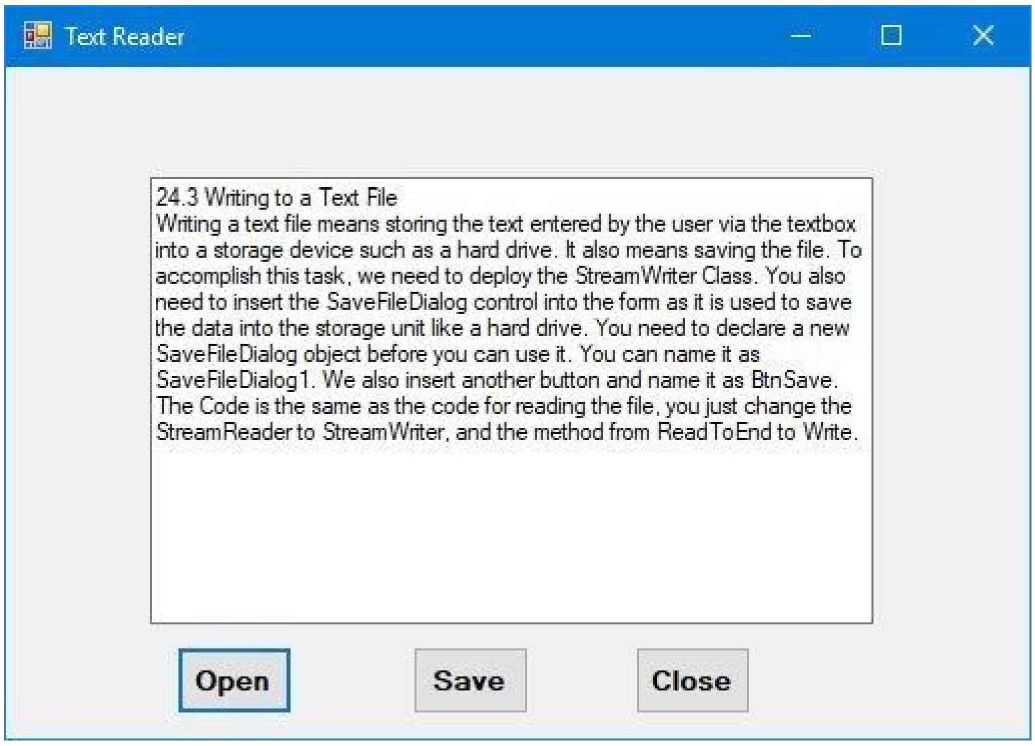
24.3 Writing to a Text File

Writing a text file means storing the text entered by the user via the textbox into a storage device such as a hard drive. It also means saving the file. To accomplish this task, we need to deploy the StreamWriter Class. You also need to insert the SaveFileDialog control into the form as it is used to save the data into the storage unit like a hard drive. You need to declare a new SaveFileDialog object before you can use it. You can name it as SaveFileDialog1. We also insert another button and name it as BtnSave. The Code is the same as the code for reading the file, you just change the StreamReader to StreamWriter, and the method from ReadToEnd to Write.

The code

```
Imports System.IO
Public Class Form1
Private Sub BtnSave_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)
Dim FileWriter As StreamWriter
Dim results As DialogResult
Dim saveFileDialog1 As New SaveFileDialog()
results = SaveFileDialog1.ShowDialog
If results = DialogResult.OK Then
FileWriter = New StreamWriter(SaveFileDialog1.FileName, False)
FileWriter.Write(TxtEditor.Text)
FileWriter.Close()
End If
End Sub
```

The Interface is shown in Figure 24.4



24.3 Writing to a Text File
Writing a text file means storing the text entered by the user via the textbox into a storage device such as a hard drive. It also means saving the file. To accomplish this task, we need to deploy the StreamWriter Class. You also need to insert the SaveFileDialog control into the form as it is used to save the data into the storage unit like a hard drive. You need to declare a new SaveFileDialog object before you can use it. You can name it as SaveFileDialog1. We also insert another button and name it as BtnSave. The Code is the same as the code for reading the file, you just change the StreamReader to StreamWriter, and the method from ReadToEnd to Write.

Figure 24.4

When you click the save button, the program will prompt you to key in a filename and the text will be saved as a text file. You can close the text editor and reopen the text file saved by you.

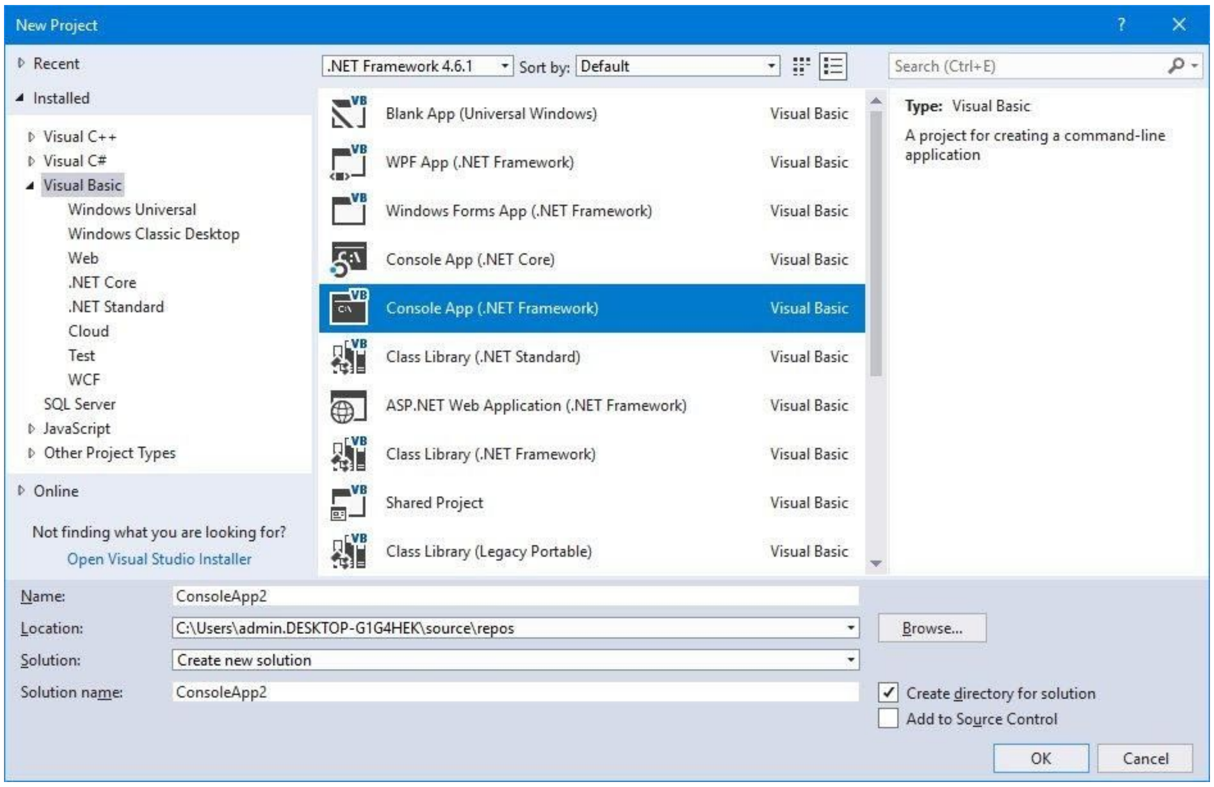
Summary

- In Section 24.1, you have understood the concepts of reading and writing a text file
- In section 24.2, you have learned how to write code to read a text file
- In Section 24.3, you have learned how to write code to write a text file.

Chapter 25 Building Console Applications

- ❖ Learn how to create a console app that display a message
- ❖ Learn how to create a console text file writer app
- ❖ Learn how to create a console text file reader app
- ❖ Learn how to create a console app using If...Then...Else

In Visual Basic 2017, you can build console applications besides the Windows Form Applications. To start creating a console application, start Visual Basic 2017 and choose Console App(.NET Framework) in the new project window, as shown in Figure 25.1 below:


New Project
Recent
Installed
Visual C++
Visual C#
Visual Basic
Windows Universal
Windows Classic Desktop
Web
.NET Core
.NET Standard
Cloud
Test
WCF
SQL Server
JavaScript
Other Project Types
Online
Not finding what you are looking for?
Open Visual Studio Installer
.NET Framework 4.6.1
Sort by: Default
Search (Ctrl+E)
Blank App (Universal Windows) Visual Basic
WPF App (.NET Framework) Visual Basic
Windows Forms App (.NET Framework) Visual Basic
Console App (.NET Core) Visual Basic
Console App (.NET Framework) Visual Basic
Class Library (.NET Standard) Visual Basic
ASP.NET Web Application (.NET Framework) Visual Basic
Class Library (.NET Framework) Visual Basic
Shared Project Visual Basic
Class Library (Legacy Portable) Visual Basic
Browse...
OK
Cancel

Type: Visual Basic

A project for creating a command-line application

Name: ConsoleApp2

Location: C:\Users\admin.DESKTOP-G1G4HEK\source\repos

Solution: Create new solution

Solution name: ConsoleApp2

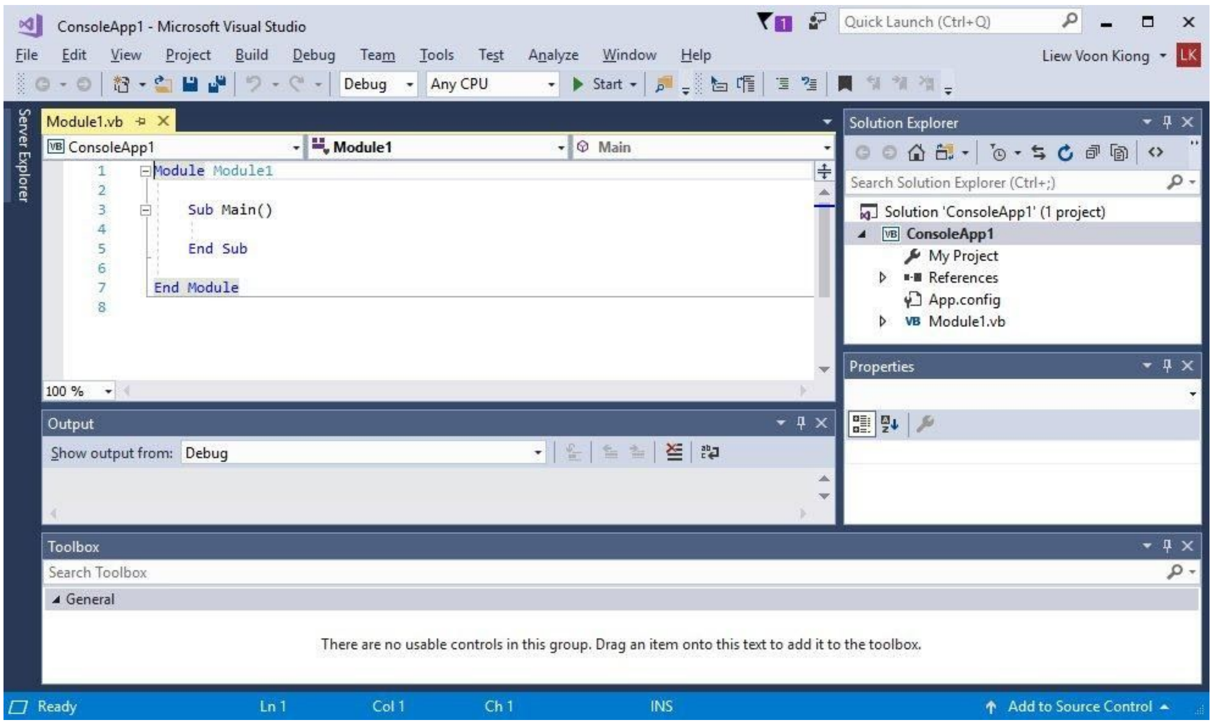
☑ Create directory for solution

☐ Add to Source Control

Figure 25.1

Retain the name as ConsoleApp or change it to the name of your choice.

Now, click on Console Application to bring up the code window, as shown in Figure 25.2 below:

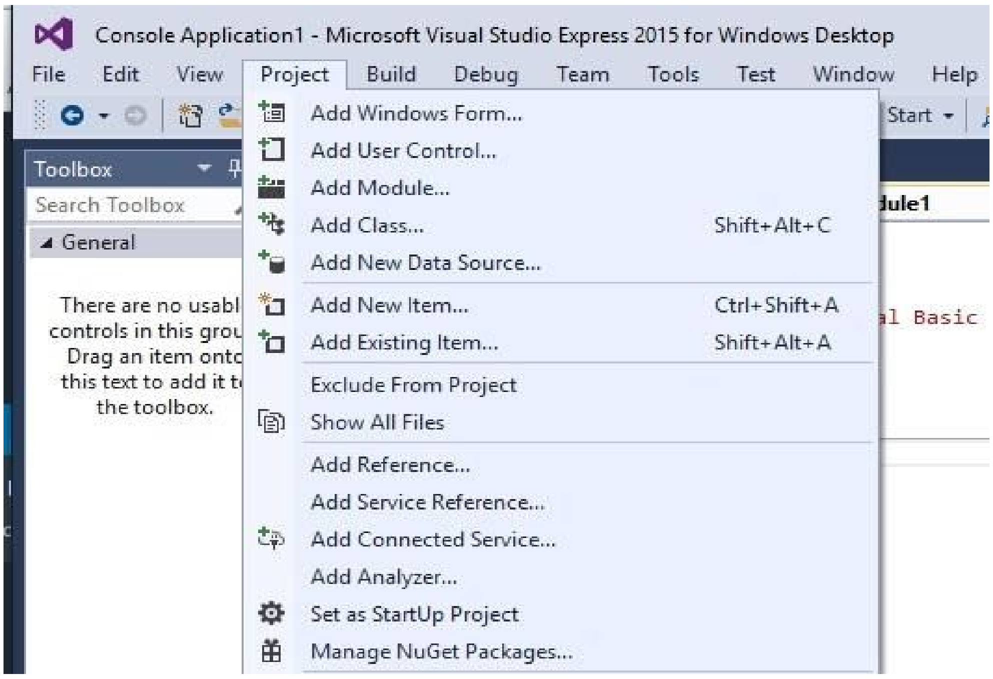


```
Module Module1

    Sub Main()

    End Sub

End Module
```

There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.

Figure 25.2

The console code window comprises modules, where the main module is module 1. You an add other modules by clicking on Project on the menu bar and click Add Module, as shown in Figure 25.3 below:



There are no usabl controls in this grou Drag an item onto this text to add it t the toolbox.

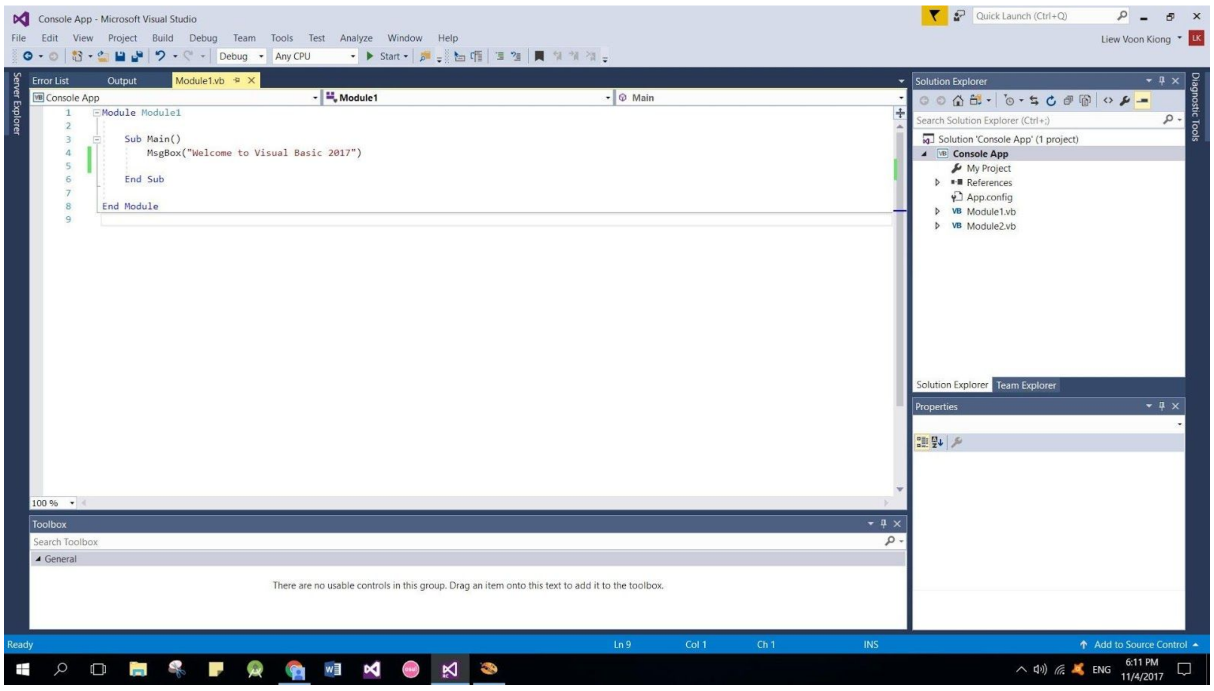
Figure 25.3

To start writing code for the console application, type your code in between Sub Main() and End Sub, as shown below:

```
Sub Main ( )
 Your code
End Sub
```

Example 25.1: Displaying a Message

The following program will display a message ” Welcome to Visual Basic 2017 Console Programming”. The function to display a message box is MsgBox(). Enter the code as follows:

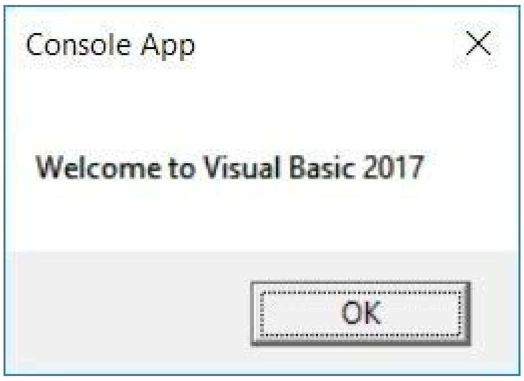


```
Module Module1

    Sub Main()
        MsgBox("Welcome to Visual Basic 2017")

    End Sub

End Module
```

Console App - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug
Any CPU
Start
Error List
Output
Module1.vb
Console App
Module1
Main
Solution Explorer
Solution 'Console App' (1 project)
Console App
My Project
References
App.config
Module1.vb
Module2.vb
Team Explorer
Properties
Toolbox
General
There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.
Ready
Ln 9
Col 1
Ch 1
INS
Add to Source Control

Figure 25.4 The Code Window

Run the program and the output is as shown in Figure 25.5 below:


Console App

Welcome to Visual Basic 2017

Figure 25.5

Example 25.2 Creating a Text File Writer in Console

We can write a Console app to create a text file in your hard drive or other storage devices. In order to write a text file to a hard drive or any storage devices, we need to use the StreamWriter class, as we have done the same thing in Chapter 24. The statement to write a text file with a file name `mytext.txt` to your hard drive is as follows:

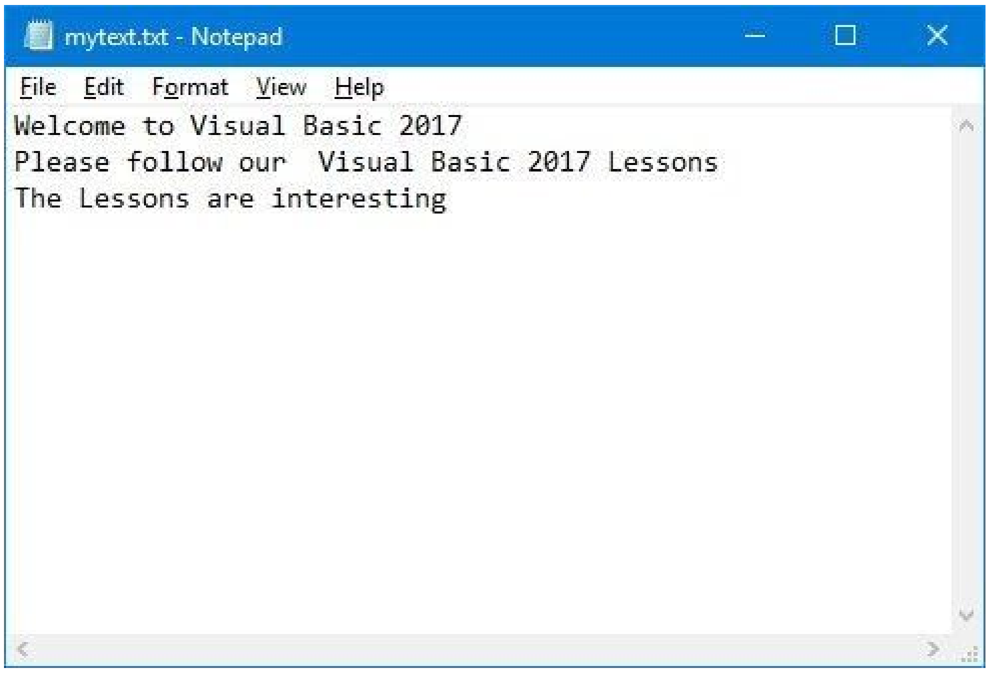
```
Dim objWriter As New
System.IO.StreamWriter("C:\Users\Documents\vb2017\mytext.txt")
```

It will create a text file in the specified location. In addition, we use the `WriteLine()` method to write text to the text file.

The Code

```
Sub Main()
Dim objWriter As New
System.IO.StreamWriter("C:\Users\Documents\vb2017\mytext.txt")
objWriter.WriteLine("Welcome to Visual Basic 2017")
objWriter.WriteLine("Please follow our  Visual Basic 2017 Chapters")
objWriter.WriteLine("The Chapters are interesting")
objWriter.Close()
objWriter.Dispose()
End Sub
```

The text file as seen in the Notepad is what as shown in Figure 25.6



```
Welcome to Visual Basic 2017
Please follow our  Visual Basic 2017 Lessons
The Lessons are interesting
```

Figure 25.6

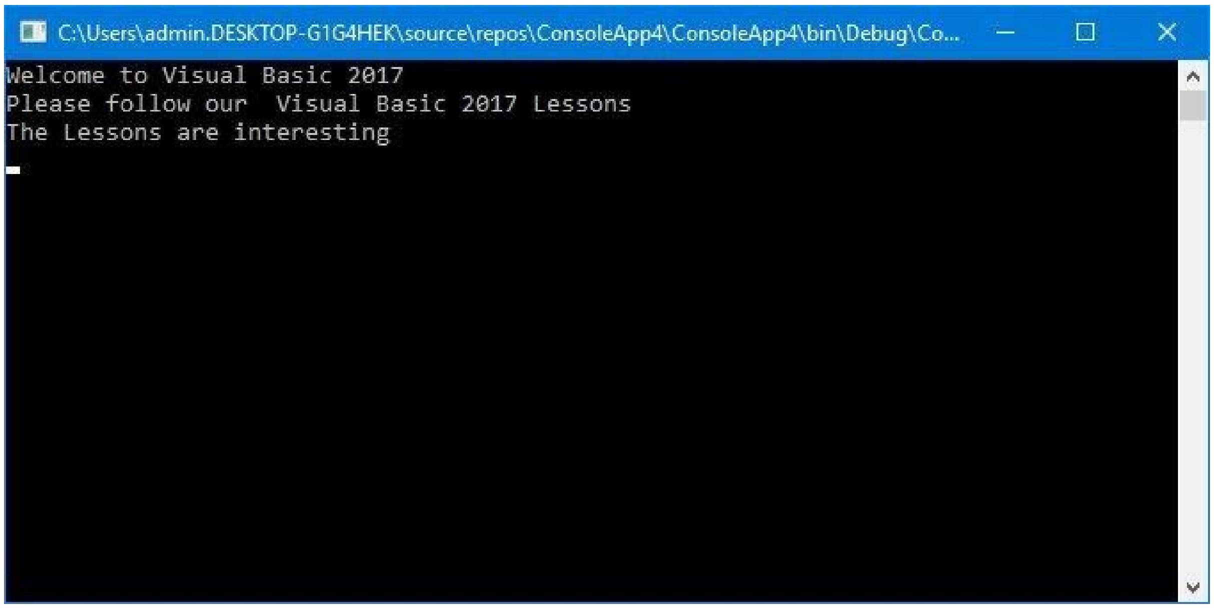
Example 25.3 Creating a Text File Reader in Console

In order to read a text file from your hard drive or any storage devices, we need to use the `StreamReader` class, as we have done the same thing in Chapter 24.

The Code

```
Sub Main()
Dim objReader As New
System.IO.StreamReader("C:\Users\Documents\vb2017\mytext.txt")
        Dim strLine As String
        'Reading  the first line
        strLine = objReader.ReadLine
        'Read until  the end of the file.
        Do While Not strLine Is Nothing
            'Write the line to the Console window.
            Console.WriteLine(strLine)
            'Read the next line.
            strLine = objReader.ReadLine

        Loop

End Sub
```

The output



```
C:\Users\admin.DESKTOP-G1G4HEK\source\repos\ConsoleApp4\ConsoleApp4\bin\Debug\Co...
Welcome to Visual Basic 2017
Please follow our  Visual Basic 2017 Lessons
The Lessons are interesting
```

Figure 25.7

Example 25.4 Creating a Console App using If...Then....Else

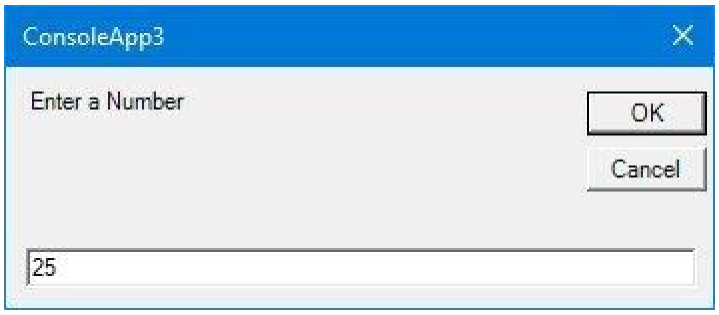
We can use `If....Then....Else` to create an Arithmetic program in the Console App.

The Code

```
Sub Main()
Dim x, y, z, total As Single
Dim firstnum As String
Dim secondnum As String
Dim sum As String
firstnum = InputBox("Enter a Number")
secondnum = InputBox("Enter a number")
sum = InputBox("The answer is")
total = Val(firstnum) + Val(secondnum)
If total = Val(sum) Then
MsgBox("Correct")
Else
MsgBox("Wrong")
```

```
End If
End Sub
```

When you run the application, you will be asked to enter a first number in an input box, as shown in Figure 25.8

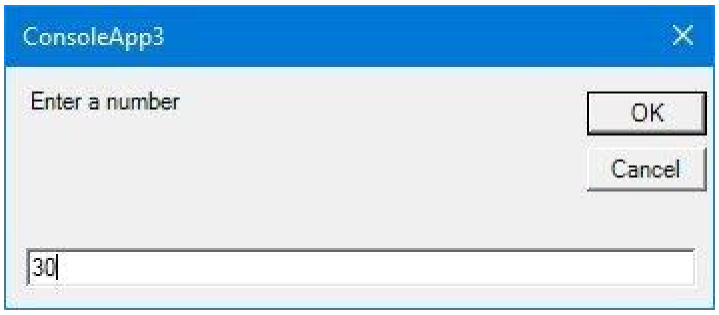


Enter a Number

25

Figure 25.8

After entering a number and click OK, you will be asked to enter the second number in a second input box, as shown in Figure 25.9

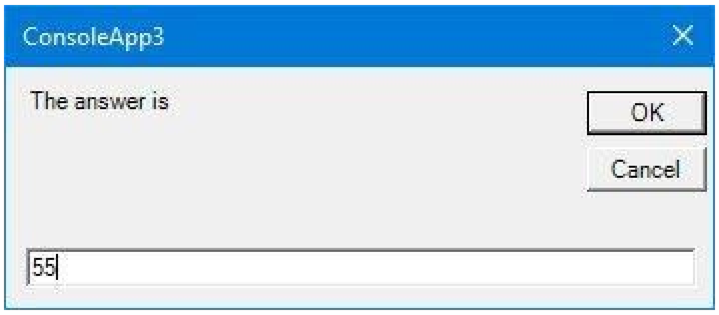


Enter a number

30

Figure 25.9

In the third input box that follows, you are asked the enter the answer, as in Figure 25.10.

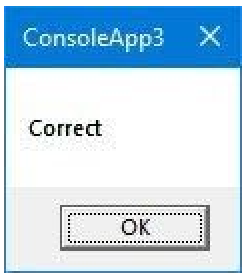


The answer is

55

Figure 25.10

Finally, after clicking the OK button, you know whether the answer you entered is correct from the message box that appears, as shown in Figure 25.11.


ConsoleApp3

Correct

Figure 25.11

Summary

- In Section 25.1, you have created a Console App that display a Message
- In section 25.2, you have built a Console App that creates a text file
- In Section 25.3, you have built a Console App that read a text file.
- In Section 25.4 You have built a Console App that performs arithmetic calculations involving the usage of If...Then...Else

Chapter 26 Creating Menu Bar and Toolbar

- ❖ Learn how to create the menu bar and add menu items
- ❖ Learn how to write code for the menu items
- ❖ Learn how to create the toolbar and add toolstrip buttons
- ❖ Learn how to write code for the toolstrip buttons

In previous chapters, we have learned how to create buttons that will trigger an action or a series of actions when the user clicks on them. However, most standard Windows applications come with very few clickable buttons, instead, they provide menu bar and toolbar so that the user can click on the items on the menu bar or the icons on the toolbar. Besides, the user can also use the shortcut keys to access the items using the keyboard. In this chapter, we will show you how to create the menu items on the menu bar and the icons on the toolbar

26.1 Creating Menu Items on the Menu Bar

In this section, we will show you how to create Menu Items on the Menu Bar. We will use the text editor we have created in chapter 24 to demonstrates how to create the menu bar. Besides, we will delete the buttons as we will develop the menu bar and toolbar for navigation. Now, open the text editor in the design mode. Next, insert the MenuStrip control into the form. You will notice that the MenuStrip control will not appear on the form, instead, it lies below the form and remains invisible at runtime, as shown in Figure 26.1.

You will also notice that a text box appears on the top of the form that displays the tip Type Here, this is where you can type in the top-level menu item. Here, we type &File in the text box, the ampersand sign is to display the menu item File with F underlined where the user can use hotkey Alt+F to access the item. After you type in the first top level item, the second text box will appear for you to enter the second item. You can enter as many top-level menu items as the form can accommodate. For our example, we create three top level menu items, File, Help and About. Next,

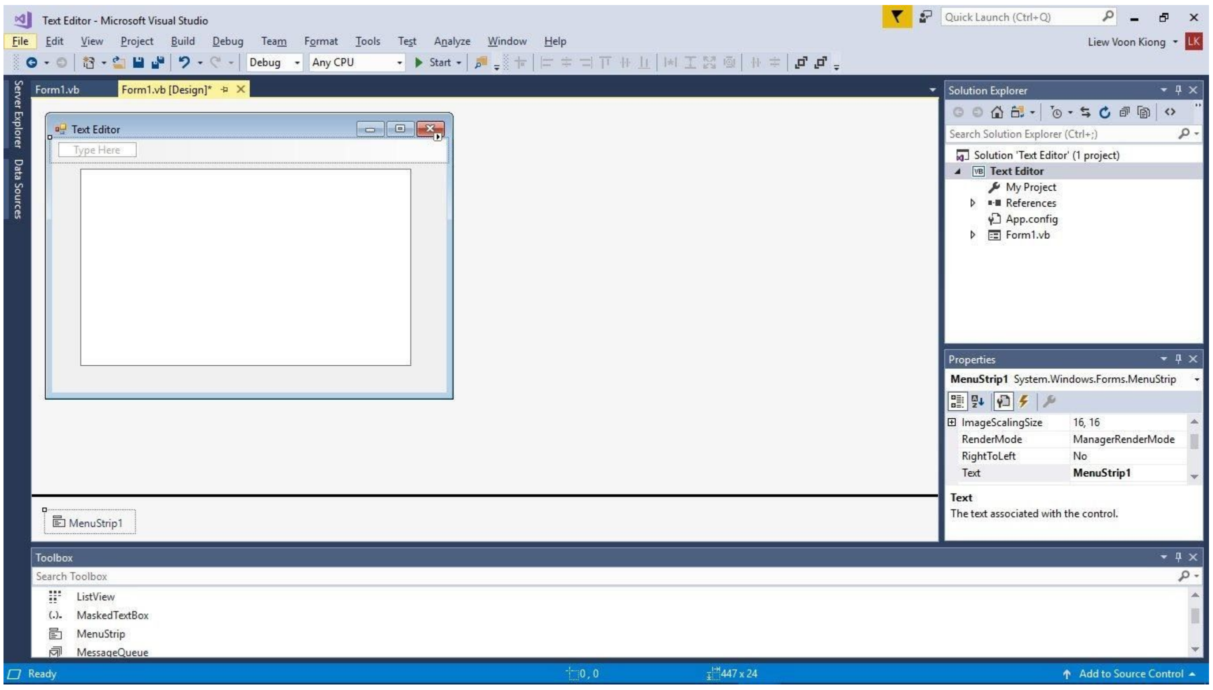

Text Editor - Microsoft Visual Studio
File Edit View Project Build Debug Team Format Tools Test Analyze Window Help
Form1.vb
Form1.vb [Design]*
Text Editor
Type Here
MenuStrip1
Solution Explorer
Solution 'Text Editor' (1 project)
Text Editor
My Project
References
App.config
Form1.vb
Properties
MenuStrip1 System.Windows.Forms.MenuStrip
Toolbox
ListView
MaskedTextBox
MenuStrip
MessageQueue
Ready

Figure 26.1

We shall now create a few items under the File menu item. To do this, click on the File item and you will get a drop-down text box will the Type Here tip, Enter &New as the first item. After you enter the first item, another drop-down text box below the New item will appear, enter Sa&ve in this text box. Enter &Open and E&xit for the next two text boxes. Now your design interface should look like the image shown in Figure 26.2. Notice the N is underlined for New, v is underlined for Save, O is underlined for Open and x is underlined for Exit.

Next, we need to write code for all the menu items so they can respond to events like clicking the mouse button or pressing a key on the keyboard. Before we write code for the items, let create two sub procedures first. One of them is to create a text file by writing the file to the hard drive or other storage devices and the other one is to open a text file from the hard drive or other storage devices. Let us name the first procedure as **`WriteFile()`** and enter the following code:

```
Sub WriteFile()
 Dim FileWriter As StreamWriter
 Dim results As DialogResult
 Dim saveFileDialog1 As New SaveFileDialog()
     results = saveFileDialog1.ShowDialog
 If results = DialogResult.OK Then
  FileWriter = New StreamWriter(saveFileDialog1.FileName, False)
  FileWriter.Write(TxtEditor.Text)
  FileWriter.Close()
 End If
 End Sub
```

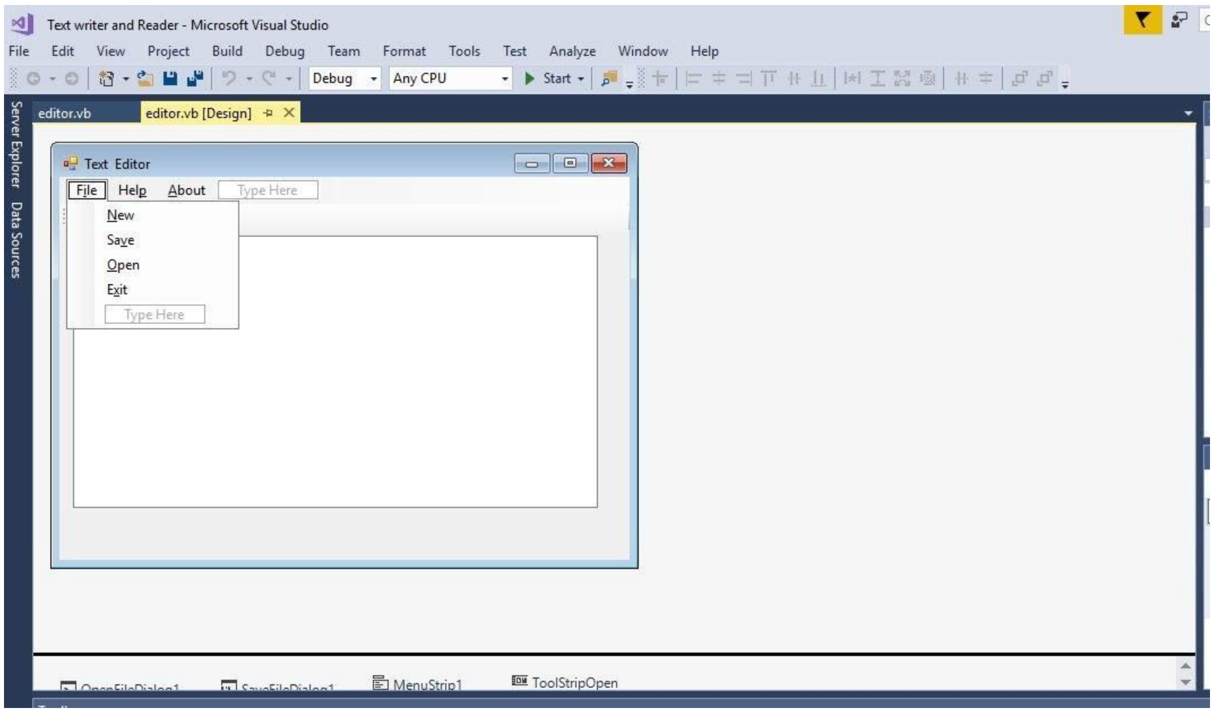

Text writer and Reader - Microsoft Visual Studio
File Edit View Project Build Debug Team Format Tools Test Analyze Window Help
Debug
Any CPU
Start
editor.vb
editor.vb [Design]
Text Editor
File Help About Type Here
New
Save
Open
Exit
Type Here
MenuStrip1
ToolStripOpen

Figure 26.2

Next, create another sub procedure **`ReadFile()`** and enter the following code:

```
Sub ReadFile()
Dim FileReader As StreamReader
Dim results As DialogResult
Dim OpenFileDialog1 As New OpenFileDialog()
    results = OpenFileDialog1.ShowDialog
 If results = DialogResult.OK Then
    FileReader = New StreamReader(OpenFileDialog1.FileName)
    TxtEditor.Text = FileReader.ReadToEnd()
    FileReader.Close()
End If
End Sub
```

For the New item, enter the following code:

```
Private Sub ToolStripNew_Click(sender As Object, e As EventArgs) Handles
ToolStripNew.Click
        TxtEditor.Text = ""
End Sub
```

For the Save item, enter the following code:

```
Private Sub ToolStripSave_Click(sender As Object, e As EventArgs) Handles
ToolStripSave.Click
        Me.WriteFile()
End Sub
```

For the Open item, enter the following code:

```
Private Sub ToolStripOpen_Click(sender As Object, e As EventArgs) Handles
ToolStripRead.Click
        Me.ReadFile()
End Sub
```

For the Exit item, enter the following code:

```
Private Sub ExitToolStripMenuItem_Click(sender As Object, e As EventArgs)
Handles ExitToolStripMenuItem.Click
        TxtEditor.Dispose()
        Me.Close()
 End Sub
```

For the Help button, enter the following code

```
Private Sub HelpToolStripMenuItem_Click(sender As Object, e As EventArgs)
Handles HelpToolStripMenuItem.Click
MsgBox("Type some text in the editor and click save to create a text
file. Click New to start a new text file. Click
Open to open a text file.")
End Sub
```

For the Help button, just enter the text “ Text Editor Ver 1.0” in the drop-down menu item, as shown in Figure 26.3

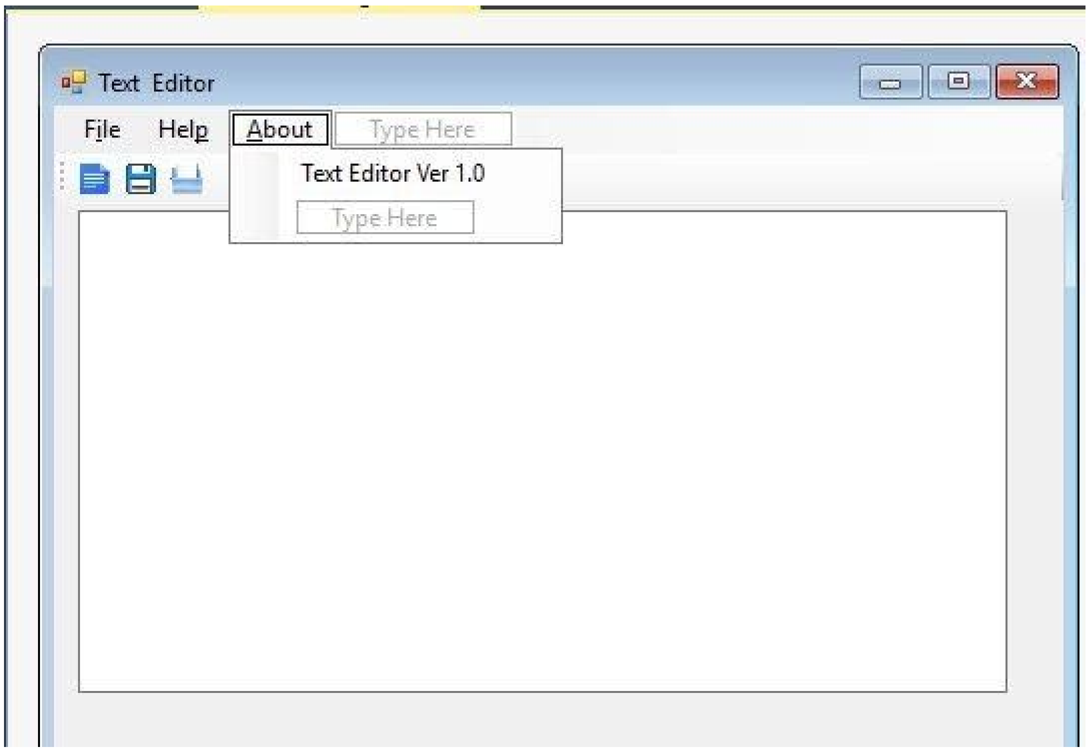
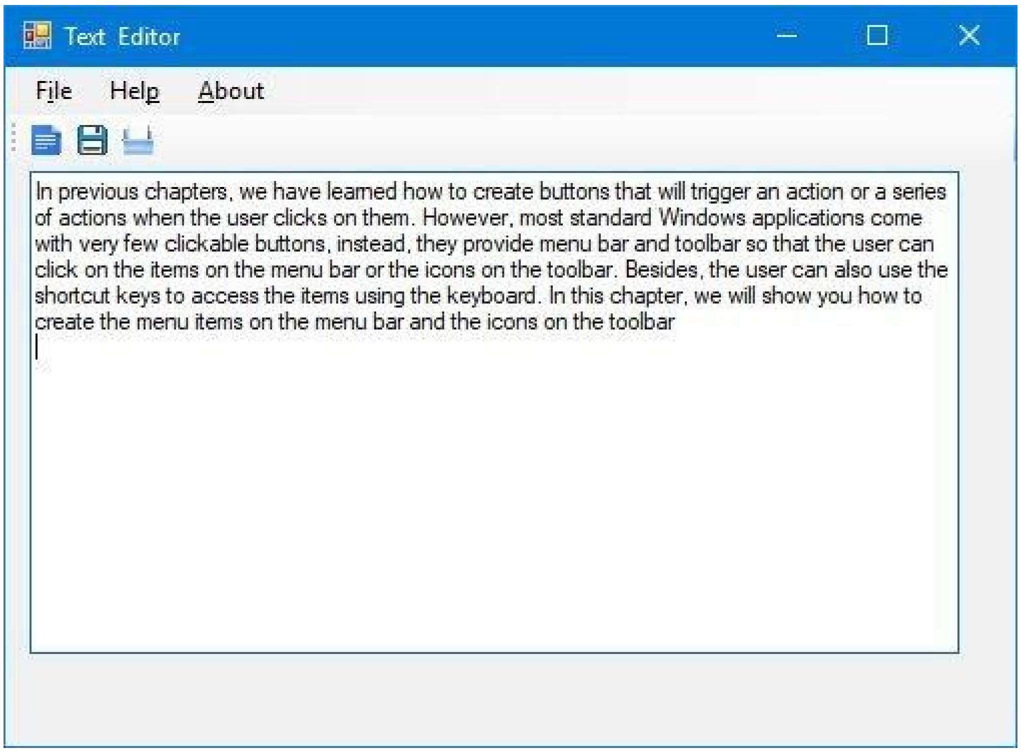


Figure 26.3

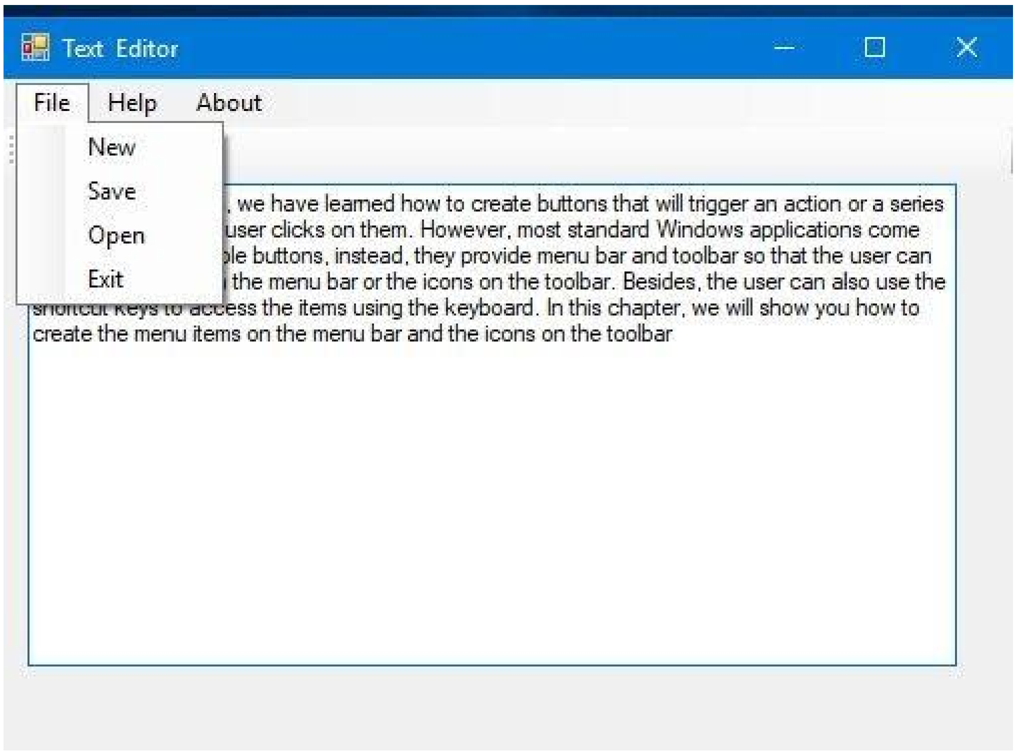
Now , run the program and enter some text, as shown in Figure 26.4



In previous chapters, we have learned how to create buttons that will trigger an action or a series of actions when the user clicks on them. However, most standard Windows applications come with very few clickable buttons, instead, they provide menu bar and toolbar so that the user can click on the items on the menu bar or the icons on the toolbar. Besides, the user can also use the shortcut keys to access the items using the keyboard. In this chapter, we will show you how to create the menu items on the menu bar and the icons on the toolbar

Figure 26.4

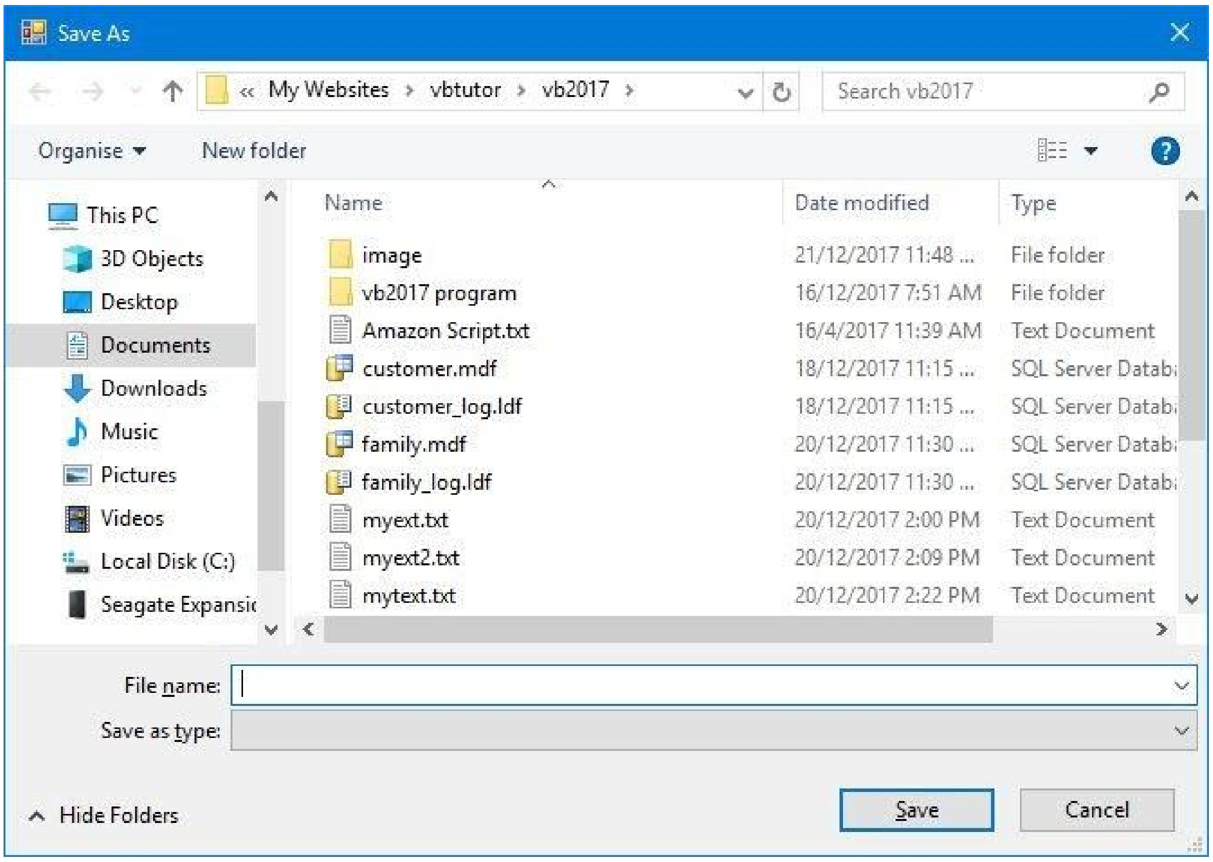
Next, click on File to bring up the drop-down menu, as shown in Figure 26.5.



, we have learned how to create buttons that will trigger an action or a series user clicks on them. However, most standard Windows applications come ole buttons, instead, they provide menu bar and toolbar so that the user can the menu bar or the icons on the toolbar. Besides, the user can also use the shortcut keys to access the items using the keyboard. In this chapter, we will show you how to create the menu items on the menu bar and the icons on the toolbar

Figure 26.5

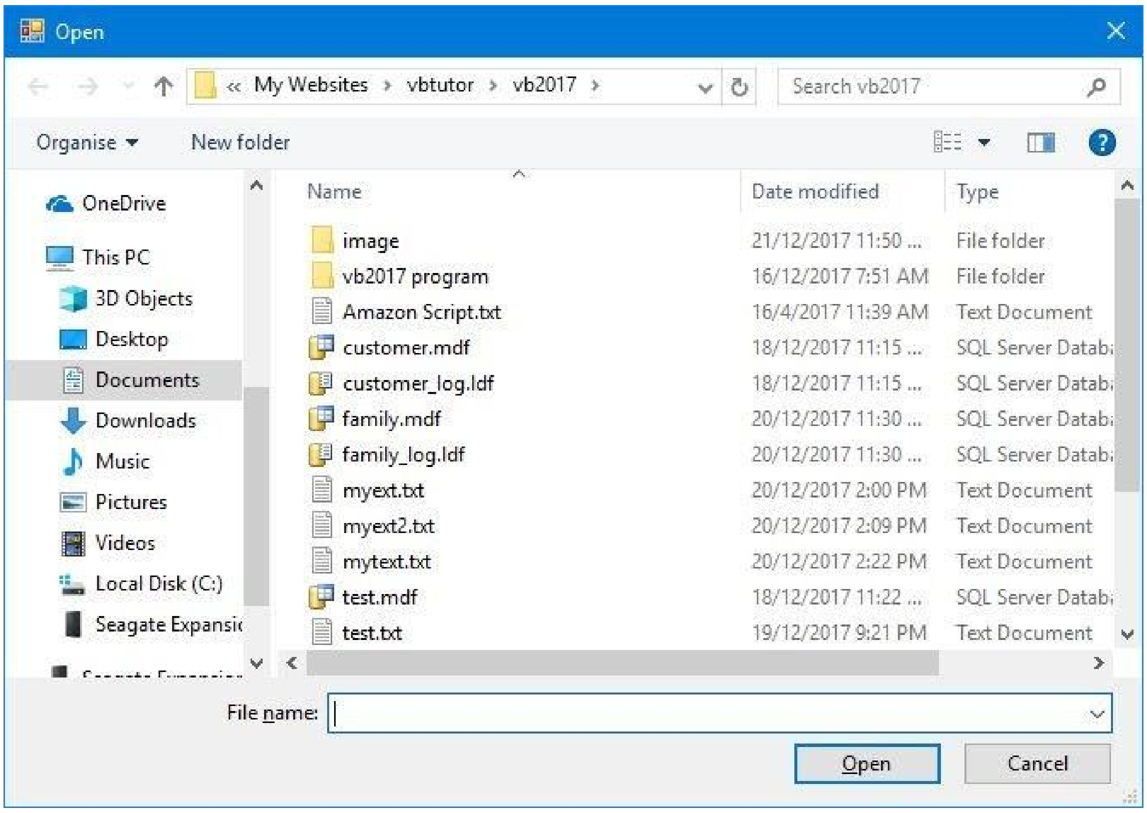
When you click on the Save button, the Save As dialog will appear, as shown in Figure 26.6. Save the file as vb2017me.txt.



Name	Date modified	Type
image	21/12/2017 11:48 ...	File folder
vb2017 program	16/12/2017 7:51 AM	File folder
Amazon Script.txt	16/4/2017 11:39 AM	Text Document
customer.mdf	18/12/2017 11:15 ...	SQL Server Datab
customer_log.ldf	18/12/2017 11:15 ...	SQL Server Datab
family.mdf	20/12/2017 11:30 ...	SQL Server Datab
family_log.ldf	20/12/2017 11:30 ...	SQL Server Datab
myext.txt	20/12/2017 2:00 PM	Text Document
myext2.txt	20/12/2017 2:09 PM	Text Document
mytext.txt	20/12/2017 2:22 PM	Text Document

Figure 26.6

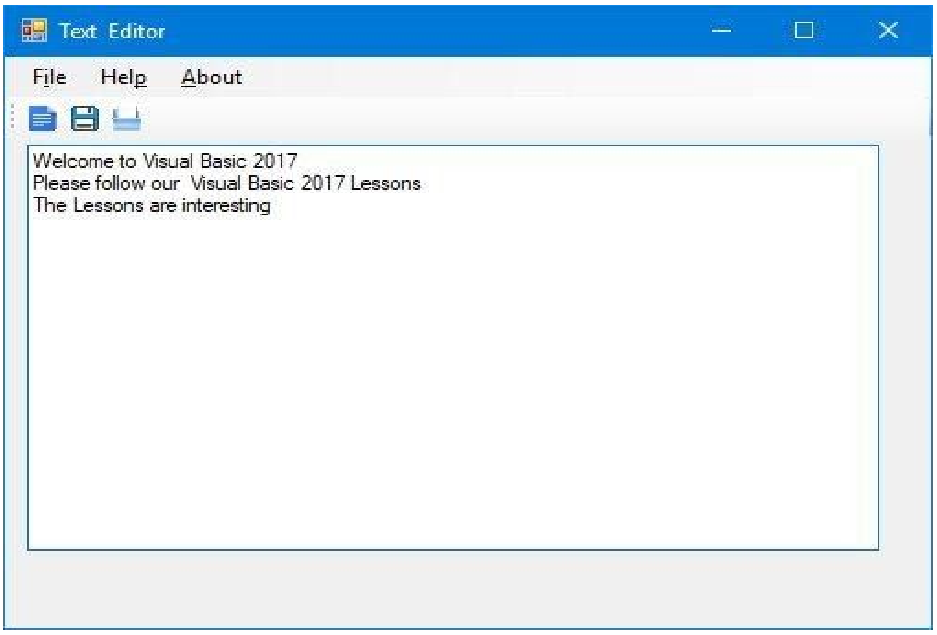
You can now click New to clear the text and type some other text and same another file. You can also open a text file by clicking the open button, and the Open dialog will appear, as shown in Figure 26.7



Name	Date modified	Type
image	21/12/2017 11:50 ...	File folder
vb2017 program	16/12/2017 7:51 AM	File folder
Amazon Script.txt	16/4/2017 11:39 AM	Text Document
customer.mdf	18/12/2017 11:15 ...	SQL Server Datab
customer_log.ldf	18/12/2017 11:15 ...	SQL Server Datab
family.mdf	20/12/2017 11:30 ...	SQL Server Datab
family_log.ldf	20/12/2017 11:30 ...	SQL Server Datab
myext.txt	20/12/2017 2:00 PM	Text Document
myext2.txt	20/12/2017 2:09 PM	Text Document
mytext.txt	20/12/2017 2:22 PM	Text Document
test.mdf	18/12/2017 11:22 ...	SQL Server Datab
test.txt	19/12/2017 9:21 PM	Text Document

Figure 26.7

Let select mytext.txt and click the Open button, you will open this text file in the text editor, as shown in Figure 26.8



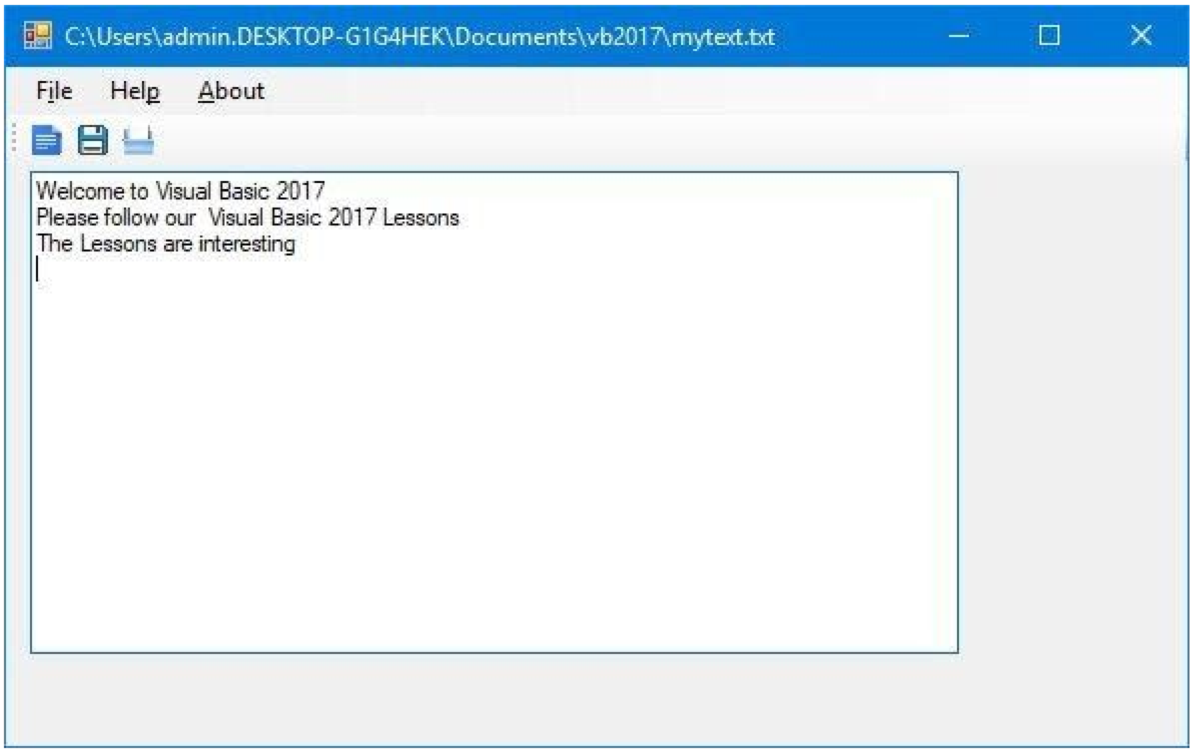
```
Welcome to Visual Basic 2017
Please follow our  Visual Basic 2017 Lessons
The Lessons are interesting
```

Figure 26.8

If you want to show the opened file name appear as the caption of the form, you enter the following line under the ReadFile() sub procedure,

```
Me.Text = OpenFileDialog1.FileName
```

Now, when you open a text file, its name will appear as the caption of the text editor, as shown in Figure 26.9



Welcome to Visual Basic 2017
Please follow our Visual Basic 2017 Lessons
The Lessons are interesting

Figure 26.9

26.2 Creating the Toolbar

In order to create the toolbar, you need to insert the ToolStrip Control into the form. It will stay below the form and remains invisible at runtime, as shown in Figure 26.10

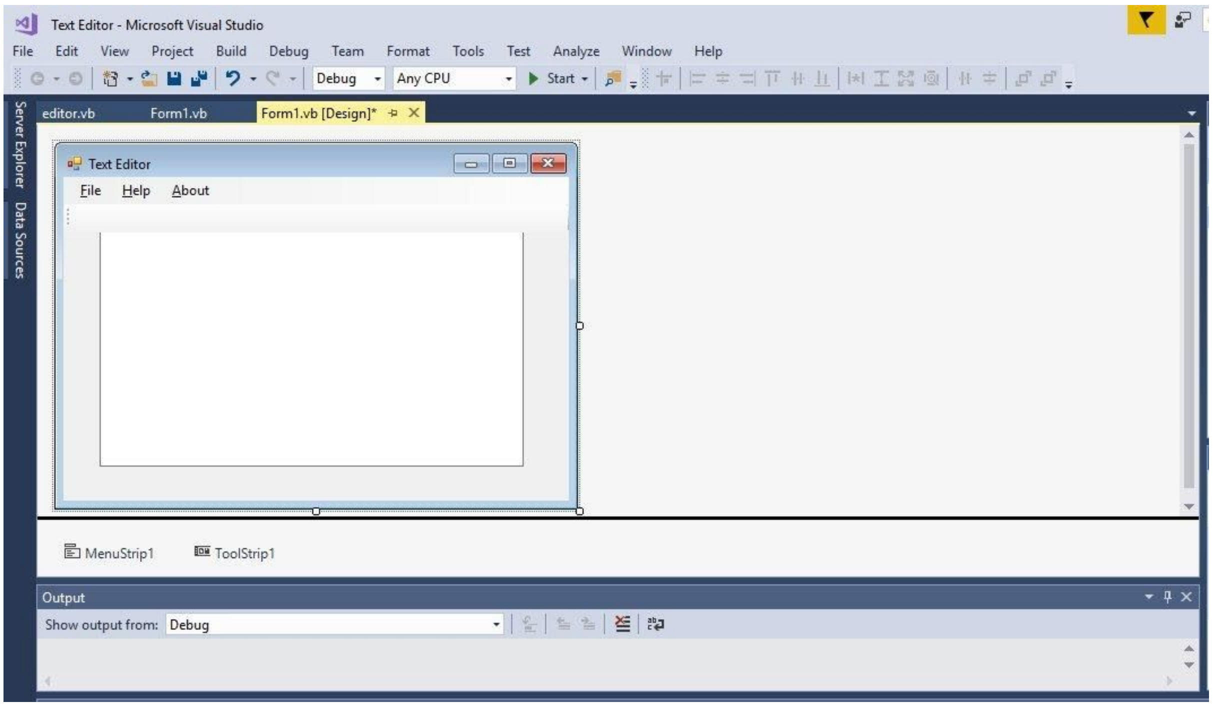


Figure 26.10

Now, click on the far left corner of the form just below the menu bar, click to bring up the first ToolStrip button with the default name ToolStripButton1. You can rename it to ToolStripNew. Now click on the image property to bring up the Select Source dialog and import an image as icon, as shown in Figure 26.11

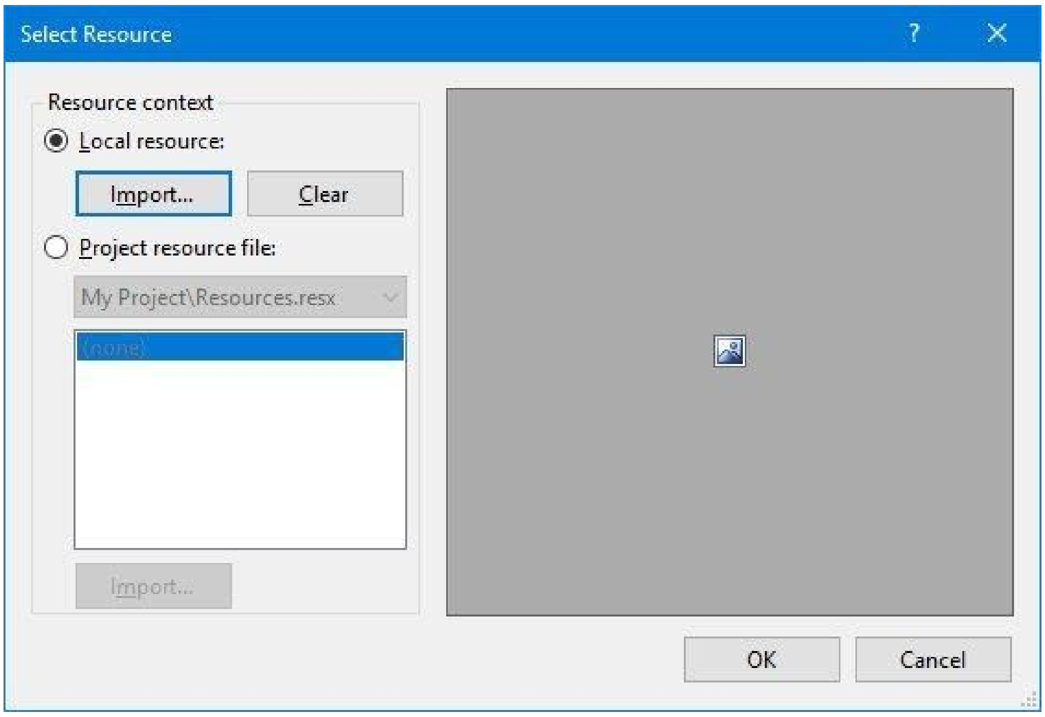


Figure 26.11

After opening an image folder and selected an image file, the image will appear in the Select Source window, as shown in Figure 26.12

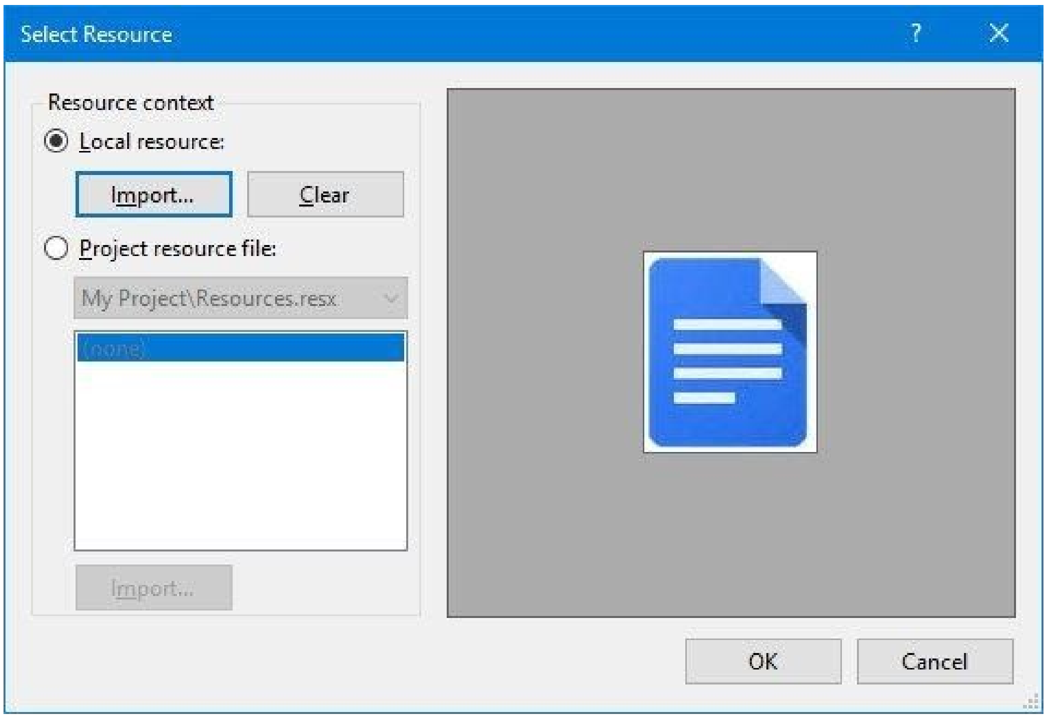


Figure 26.12

After clicking the OK button, the image will appear as an icon to replace the default toolstrip button on the toolbar, as shown in Figure 26.13

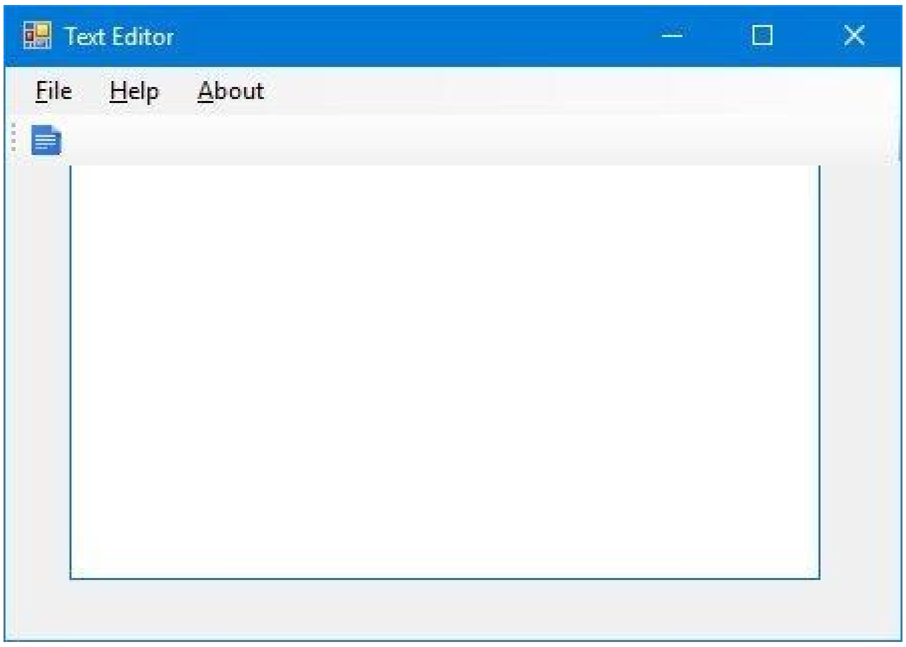
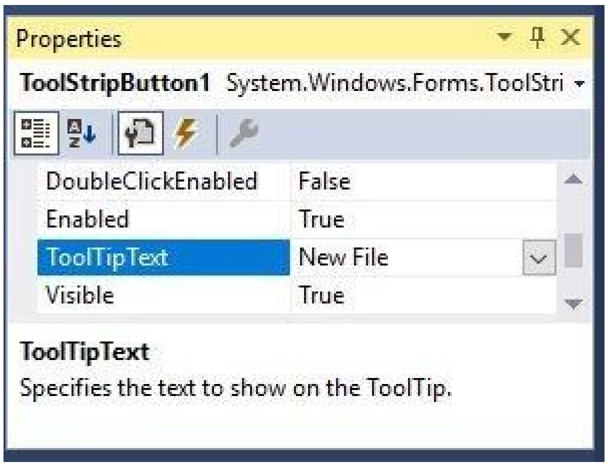


Figure 26.13

In addition, you might want to add the Tooltip Text in the properties window to the toolstrip button, so that the tip will appear when the mouse hover about this toolstrip button on the toolbar. You add the Tooltip text as shown in Figure 26.14



DoubleClickEnabled	False
Enabled	True
ToolTipText	New File
Visible	True

ToolTipText
Specifies the text to show on the ToolTip.

Figure 26.14

When you run the application and place the mouse over the ‘New’ toolstrip button, the tip “New File” will appear, as shown in Figure 26.15

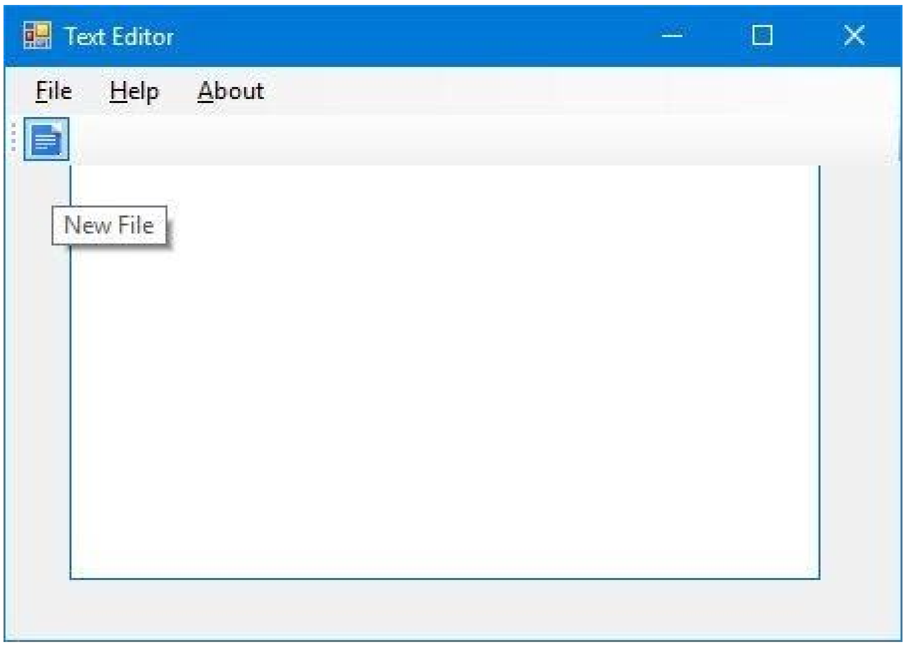


Figure 26.15

Next, continue to add two more toolbar buttons (one for saving file and the other one for opening file) on the toolbar until you obtain the design interface, as shown in Figure 26.16.

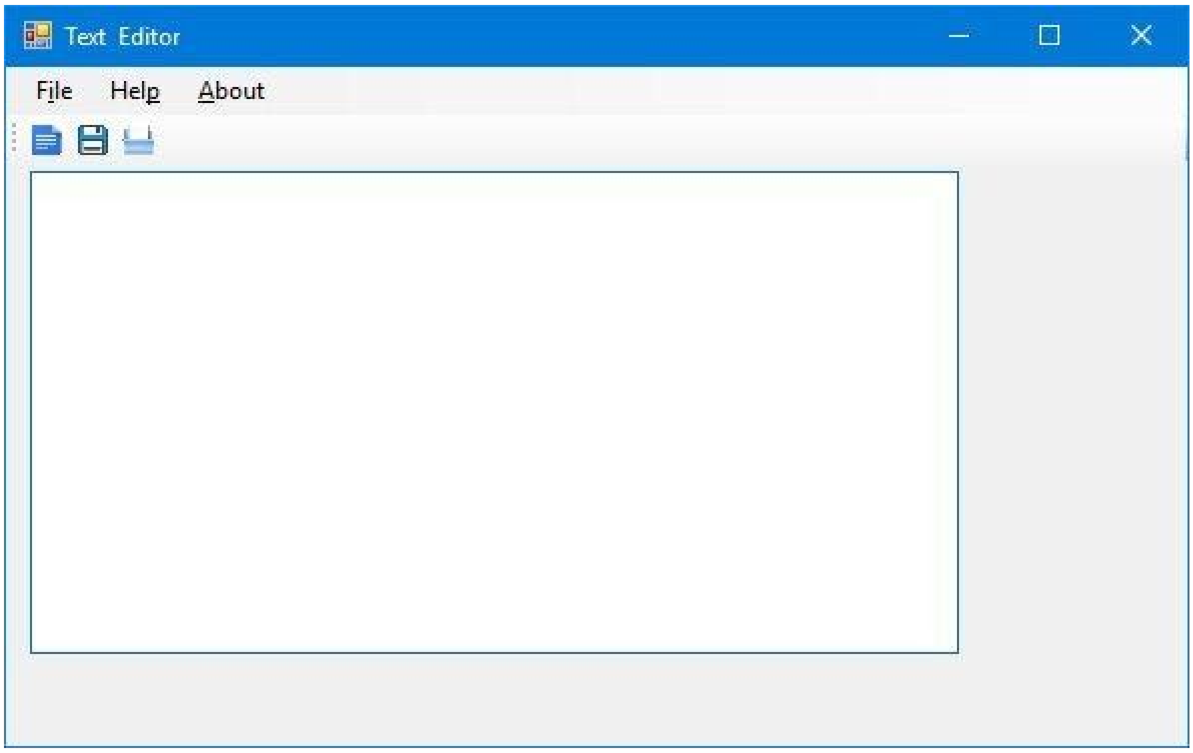


Figure 26.16

Now, it is time to write code for the three toolstrip buttons on the toolbar.

For the ‘New’ toolbar button, enter the following code:

```
Private Sub ToolStripNew_Click(sender As Object, e As EventArgs) Handles
ToolStripNew.Click
        TxtEditor.Text = ""
End Sub
```

For the “Save” toolbar button, enter the following Code:

```
Private Sub ToolStripSave_Click(sender As Object, e As EventArgs) Handles
ToolStripSave.Click
        Me.WriteFile()
End Sub
```

For the “Open” toolbar button, enter the following Code:

```
Private Sub OpenToolStripMenuItem_Click(sender As Object, e As EventArgs)
Handles OpenToolStripMenuItem.Click
        Me.ReadFile()
End Sub
```

Finally, you have added both the functional menu bar and toolbar. You can add more menu items and toolstrip buttons and experiment with them.

Summary

- In Section 26.1, you have learned how to create a menu bar by adding top-level menus as well as menu items. You have also learned how to write code for the menu items.
- In section 262, you have built the toolbar by adding toolstrip buttons to it. You have also learned how to write code for the toolstrip buttons.

Chapter 27 Deploying your VB 2017 Applications

- ❖ Learn how to publish and deploy your Visual Basic 2017 applications

Having developed a Visual Basic 2017 application, you might want to publish and distribute it to the users. Before deploying your application, you need to test and debug your application to ensure it is error-free.

Publishing your application is a quite an easy procedure, it just requires a few clicks. First, load your application in Visual Basic 2017. In this example, we wish to publish the Draw Text application. In the Visual Basic 2017 IDE, choose Build and then select Publish Draw text, as shown in Figure 27.1

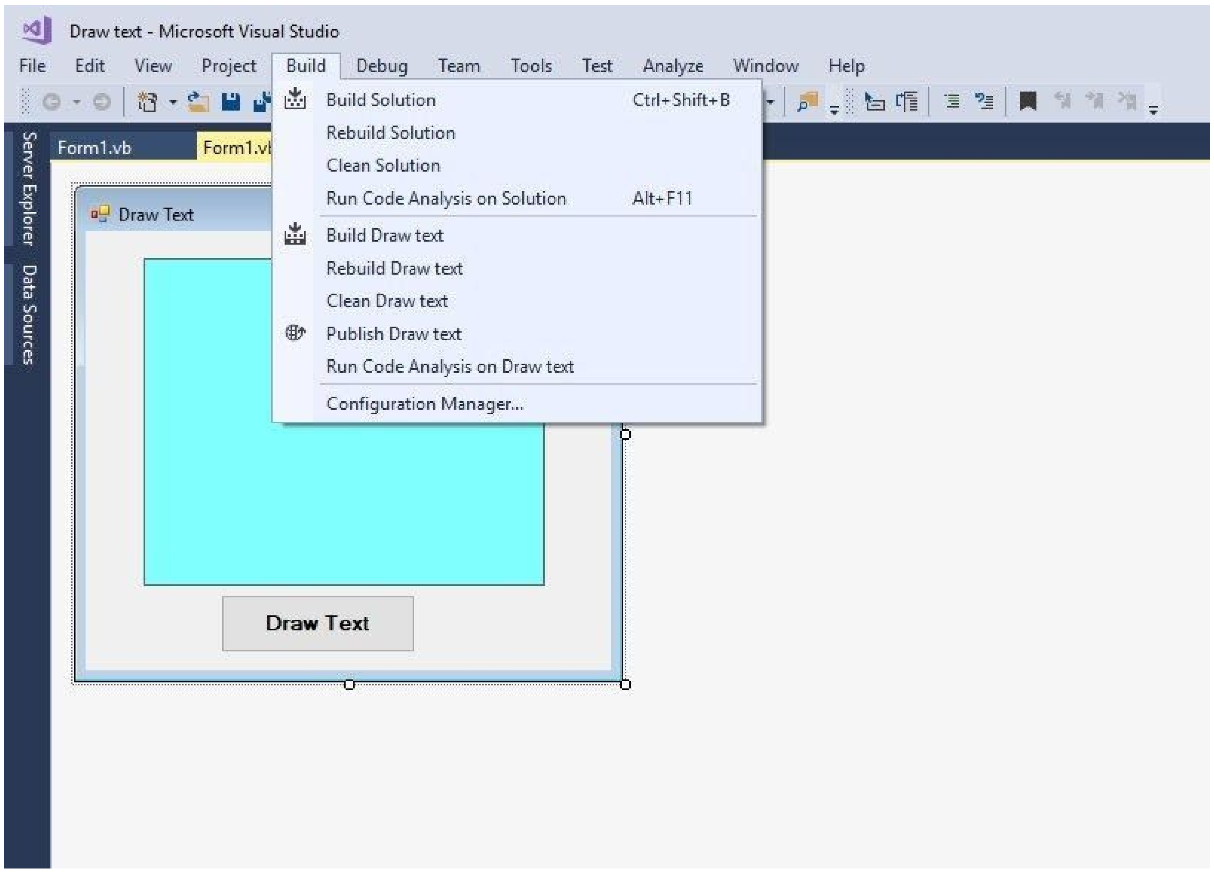
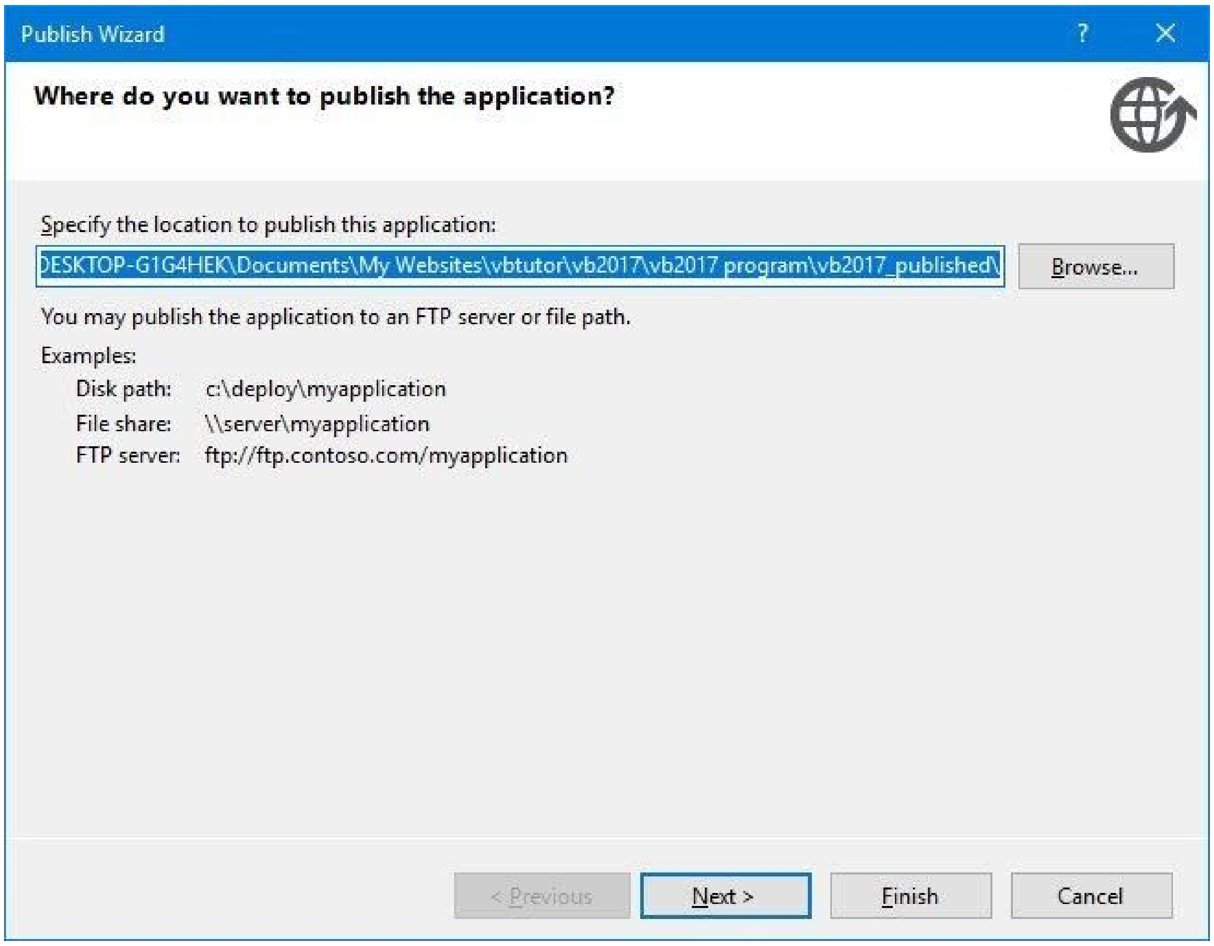


Figure 27.1

After clicking Publish Draw Text, the Publish Wizard will ask you to choose a location that you want to publish this application. You may publish the application on your local drive or to a remote location via FTP, as shown in Figure 27.2.



Where do you want to publish the application?

Specify the location to publish this application:

DESKTOP-G1G4HEK\Documents\My Websites\vbtutor\vb2017\vb2017 program\vb2017_published\

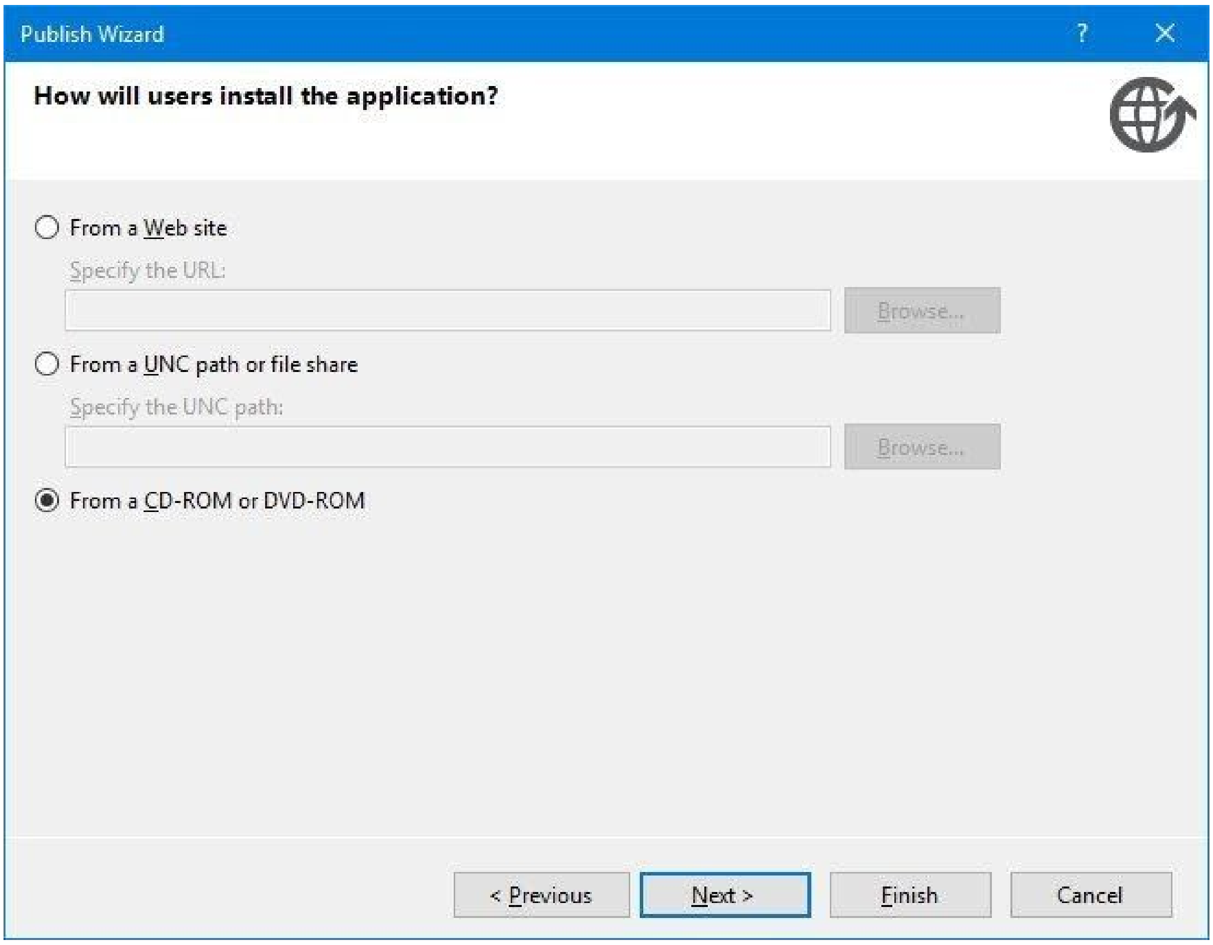
You may publish the application to an FTP server or file path.

Examples:

Disk path:	c:\deploy\myapplication
File share:	\\server\myapplication
FTP server:	ftp://ftp.contoso.com/myapplication

Figure 27.2

Click the Next button to continue. Now, the Publish Wizard will ask you how users will install the application. There are three options, from a website, from a UNC path or file share and from a CD-ROM or DVD-ROM. In this example, we choose CD-ROM or DVD-ROM, as shown in Figure 27.3


Publish Wizard

How will users install the application?

( ) From a Web site

Specify the URL:

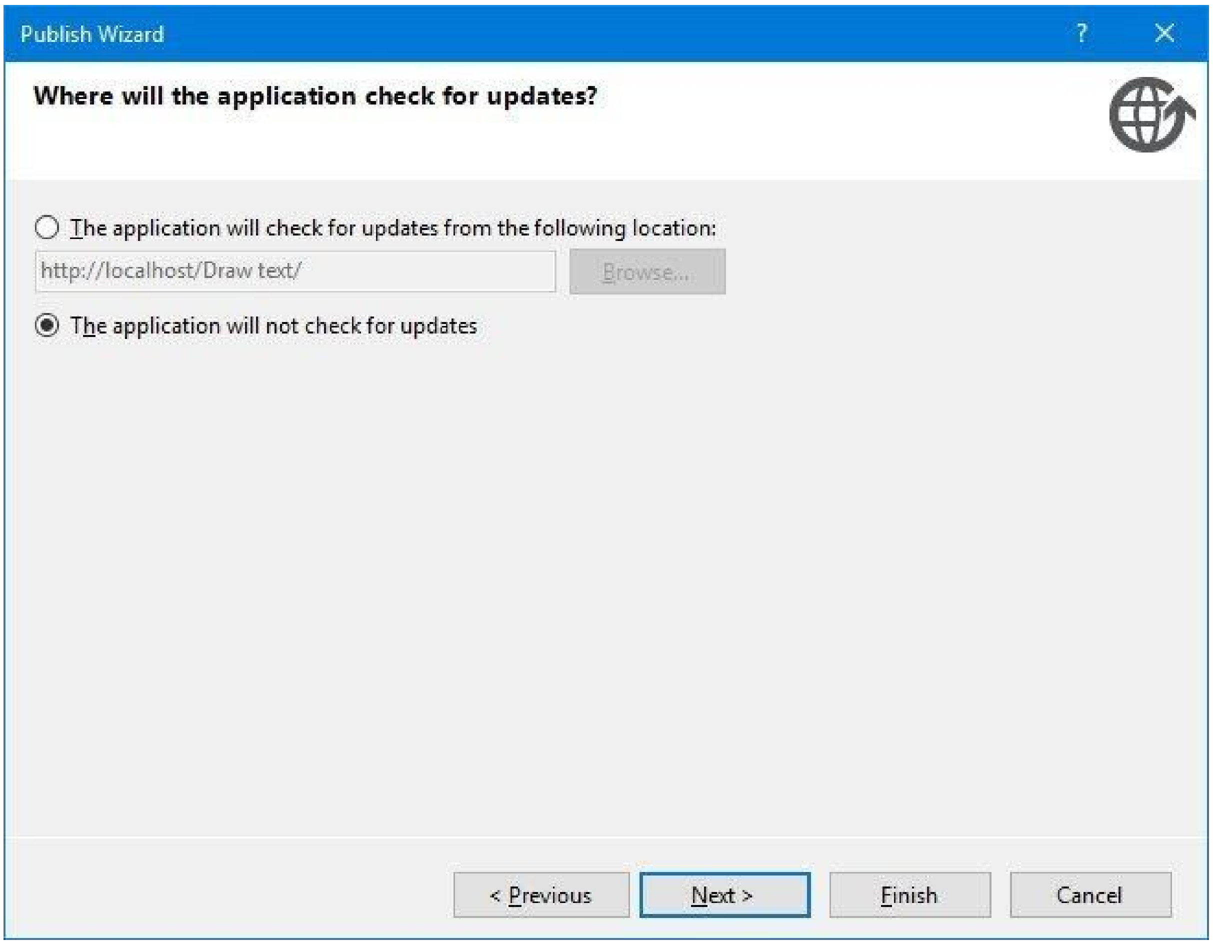
( ) From a UNC path or file share

Specify the UNC path:

(●) From a CD-ROM or DVD-ROM

Figure 27.3

You may click the Next button to continue with the Publish Wizard or the Finish button to complete the publishing process. If you click the Next button, the Publish Wizard will ask where will the application check for updates. In our example, we choose that the application will not check for updates, as shown in Figure 27.4



Publish Wizard

Where will the application check for updates?

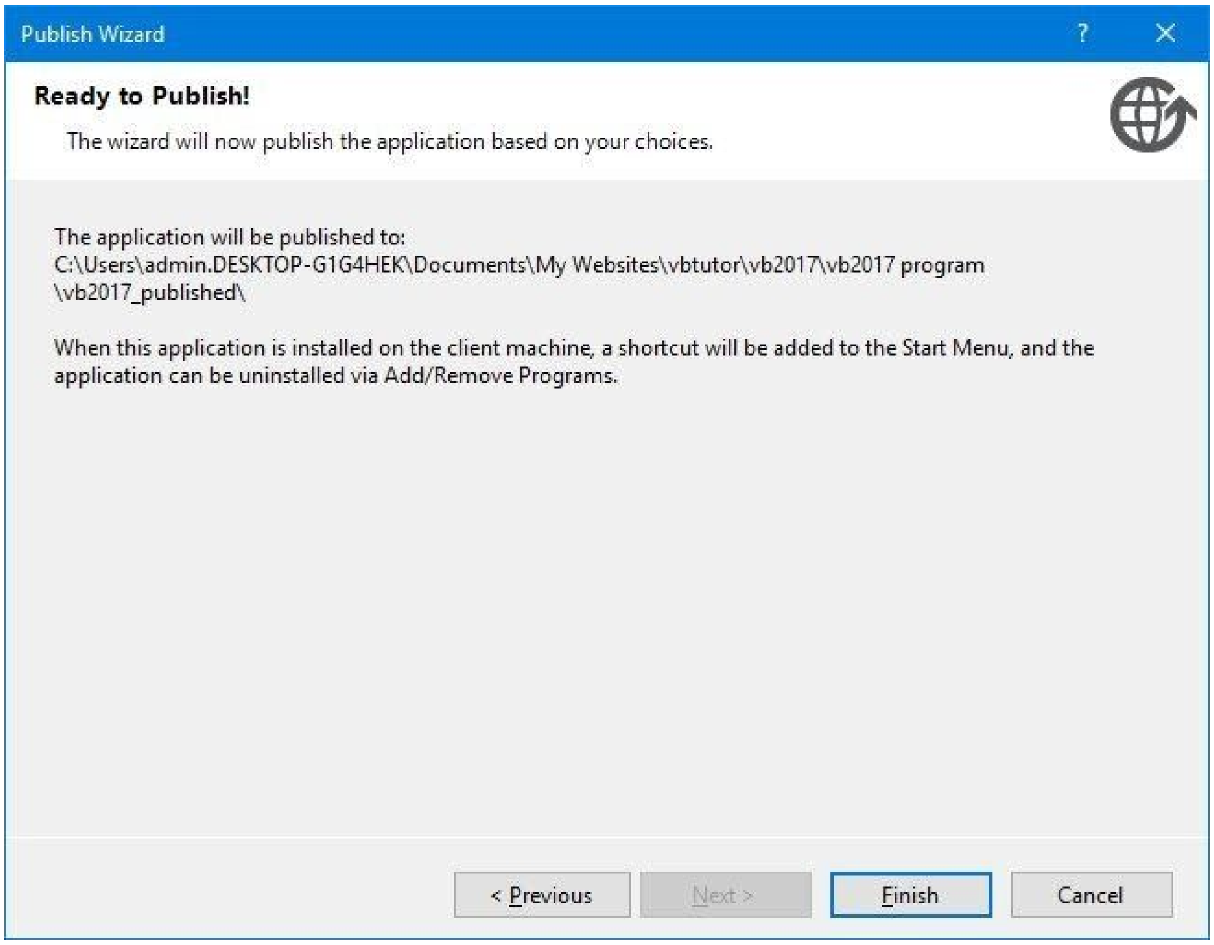
( ) The application will check for updates from the following location:

http://localhost/Draw text/

[✓ The application will not check for updates]

Figure 27.4

After clicking the Next button, the final screen of the Publish Wizard will show you where the application will be published to, as shown in Figure 27.5



Ready to Publish!

The wizard will now publish the application based on your choices.

The application will be published to:
C:\Users\admin.DESKTOP-G1G4HEK\Documents\My Websites\vbtutor\vb2017\vb2017 program
\vb2017_published\

When this application is installed on the client machine, a shortcut will be added to the Start Menu, and the application can be uninstalled via Add/Remove Programs.

Figure 27.5

Upon clicking the Finish button, the installation files and folder will be saved to the specified location. In our examples, the folder is Application files and the files are Draw text.application and setup.exe. You may now burn the folder and files into a CD-ROM, DVD-ROM or upload them to a website for distribution to the users.

Summary

- In this chapter, you have learned how to create installation files to deploy your Visual Basic 2017 Applications.

Appendix

ASCII Code

ASCII	Chr	ASCII	Chr	ASCII	Chr
8	Backspace	61	=	98	b
13	Enter key	62	>	99	c
32	Space	63	?	100	d
33	!	64	@	101	e
34	"	65	A	102	f
35	#	66	B	103	g
36	$	67	C	104	h
37	%	68	D	105	i
38	&	69	E	106	j
39	'	70	F	107	k
40	(	71	G	108	l
41	)	72	H	109	m
42	*	73	I	110	n
43	+	74	J	111	o
44	,	75	K	112	p
45	-	76	L	113	q
46	.	77	M	114	r
47	/	78	N	115	s
48	0	79	O	116	t
49	1	80	P	117	u
50	2	81	Q	118	v
51	3	82	R	119	w
52	4	83	S	120	x
53	5	84	T	121	y
54	6	85	U	122	z
55	7	86	V	123	{

56	8	87	W	124	|
57	9	88	X	125	}
58	:	89	Y	126	~
59	;	90	Z	127	DEL
60	<	97	a		

Index