DAT 20903

PRINCIPLES OF DOT NET PROGRAMMING

Mazniha binti Berahim
Rafizah binti Mohd Hanifa
Shamsul bin Mohamad
Abdul Halim bin Omar

INFORMATION TECHNOLOGY DEPARTMENT

CENTER FOR DIPLOMA STUDIES

Third Printing (Learning Module), 2017

Published and printed by:

UTHM Publisher
Universiti Tun Hussein Onn Malaysia
86400 Parit Raja, Batu Pahat,
Johor Darul Ta'zim.
Tel: 07 – 453 7454
Fax: 07 – 453 6145
E-mail: pt@uthm.edu.my
Website: www.uthm.edu.my/pt

INTRODUCTION

In general, this module is a learning guide for the course DAT20903 Principles of Dot Net Programming offered at Information Technology Department, Center for Diploma Studies, Universiti Tun Hussein Onn Malaysia (UTHM).

GOAL

This module aims to introduce students to program development environment and improve their programming skills in writing applications that can assist in solving problems using .Net programming language.

LEARNING OUTCOME

At the end of this course, students can:

1. Solve the problem using C#, ASP.NET or VB.NET programming language. (PLO4, C3, CTPS)
2. Generate a computer program using C#, ASP.Net or VB.Net programming language to solve a given problem. (PLO2, P3, PS)
3. Contribute to the development of a team project. (PLO5, A2, TS)

SYNOPSIS

This module is divided into 7 topics which covered an introduction to Visual Studio.NET programming concepts through the use of high level languages VB.NET or ASP.NET. The design of graphical user interfaces and programming style with effective coding included for windows and web application development. Accessing database using ADO.NET in application development also introduced.

ASSESSMENT

The method of assessment for this course involves two components:

(a)	Continuous Assessment		(60%)
	(i)	Test	(20%)
	(ii)	Project	(20%)
	(iii)	Laboratory	(20%)
(b)	Final examination		(40%)

REFERENCES

Reference lists are attached at the end of each topic.

CONTENT **PAGE**

INTRODUCTION **iii**

GOAL **iii**

LEARNING OUTCOME **iii**

SYNOPSIS **iv**

ASSESSMENT **iv**

TOPIC 1 INTRODUCTION TO THE VISUAL STUDIO .NET INTEGRATED DEVELOPMENT ENVIRONMENT **1**

1.1 Introduction 1

1.2 Basic Architecture of The Microsoft .NET Framework 2

1.2.1 Common Language Runtime 2

1.2.2 Class Library 3

1.2.3 Windows Forms and ASP.NET 4

1.3 Important Features of Visual Studio IDE 5

1.3.1 Quick Tour of the Visual Studio IDE 6

1.3.2 New Project Dialog 7

1.3.3 Menu Bar and Toolbar 11

1.3.4 Help Menu 12

1.4 Component of Visual Studio .NET IDE 13

1.4.1 Solution Explorer 15

1.4.2 Toolbox 16

1.4.3 Properties Window 17

1.4.4 Output Window and Error List Window 19

TOPIC 2 INTRODUCTION TO VISUAL BASIC .NET **21**

2.1 Introduction 21

2.2 Event-driven and Object Oriented Programming Concept 22

2.3 Object and Class 23

2.3.1 Basic Setting of Objects 23

2.3.2 Works with Object Control 24
2.4 Phases in Develop an Applications to Solve a Problem 25
2.4.1 Analysis 25
2.4.2 Plan or Design 25
2.4.2 Implement 26
2.5 Organize an Application Files 27
2.3 Project Debugging 28
2.6.1 Syntax Errors 28
2.6.2 Run Time Errors 29
2.6.3 Logic Errors 29

TOPIC 3 GRAPHICAL USER INTERFACE DESIGN 31
3.1 Introduction 31
3.2 Common Properties of Object Control 31
3.2.1 Objects Name 32
3.2.2 Object Appearance and Layout 33
3.2.3 Objects Behavior 34
3.3 Basic Objects 34
3.3.1 Label and Button 35
3.3.2 Text Boxes 42
3.3.3 Group Boxes 43
3.3.4 Check Boxes 43
3.3.5 Radio Buttons 44
3.4 Properties Setting of the Controls at Run Time 44
3.4.1 Clearing Text Boxes and Labels 44
3.4.2 Resetting the Focus 45
3.4.3 Setting Checked Property of Radio Buttons and Check Boxes 45
3.4.4 Changing the Color of Text 45
3.4.5 Changing Multiple Properties of a Control 46
3.5 User Interface Design with Convenience Features 47
3.5.1 General Principles in Designing the User Interface 47

3.5.2 Keyboard Access Keys 48
3.5.3 Default and Cancel Properties of Buttons 49
3.5.4 Tab Order for Controls 50
3.5.6 ToolTips 51
3.4 Menus and Submenus 51
3.5.1 Menus 52
3.5.2 Submenus 56
3.6 Dialog Boxes 57
3.6.1 Method to Show Dialog Boxes 57
3.6.2 Add Dialog Boxes Control 58
3.6.3 Message Boxes 60
3.6.4 Input Boxes 63

TOPIC 4 PROGRAMMING FOR APPLICATION DEVELOPMENT 66
4.1 Introduction 66
4.2 Programming Style 67
4.2.1 Statements, Keywords and Comments 67
4.2.3 Scope: Public or Private 68
4.3 Variables and Constants 68
4.3.1 Data Types 69
4.3.2 Declaration of a Variable 70
4.3.3 Scope of Variable 70
4.3.4 Constants: Named and Intrinsic 72
4.4 Operators 73
4.4.1 Arithmetic Operators 75
4.4.2 Comparison Operator 75
4.4.3 Assignment Operator 76
4.4.4 String Concatenation Operator 77
4.4.5 Order of Operations 78
4.5 Procedures 79
4.5.1 Difference between Functions and Subs 79

4.5.2 Create Own Sub Procedures 80
4.5.3 Using Parameters in Sub Procedure 81
4.5.4 Using ByVal and ByRef in Sub Procedure 82
4.5.5 Function Procedures 83
4.5.6 Event Handling Procedures 87
4.6 Control Structures 87
4.6.1 Conditions 88
4.6.2 Selection Control Structure 91
4.6.3 Repetition Control Structure 97
4.7 Array and Collections 103
4.7.1 Single-Dimension 104
4.7.2 Two-Dimension Array 106
4.7.3 Applications of Array in List Boxes and Combo Boxes 107

TOPIC 5 WEB APPLICATION DEVELOPMENT 117
5.1 Introduction 117
5.2 ASP.NET Web Forms 118
5.2.1 Features of ASP.NET Web Forms 119
5.2.2 Advantages of a Web Forms-Based Web Application 120
5.3 Creating a Web Application Project and a Page 121

TOPIC 6 ACCESSING DATABASE USING ADO.NET 125
6.1 Introduction 125
6.2 Database Terminologies 126
6.3 ADO.Net Object Model 126
6.3.1 Database Provider 129
6.3.2 Dataset 130
6.4 Connecting to a Database 130
6.5 View Data in DataGridView Control 133
6.5 Manipulate Database Records 137

TOPIC 7 EFFECTIVE AND SECURE CODING **138**

7.1 Input Validation 139

7.1.1 TryParse Method for Input Validation 139

7.2 Exceptions Handling 140

7.2.1 Exception Keywords 141

7.2.2 Types of Exceptions 143

TOPIC 1

INTRODUCTION TO THE VISUAL STUDIO .NET INTEGRATED DEVELOPMENT ENVIRONMENT

Learning Outcome

At the end of this topic, student should be able to:

1. Describe a basic architecture of .NET Framework,
2. List important features of Visual Studio .NET's IDE,
3. Get familiar with the types of commands contained in the IDE's menus and Toolbars,
4. Understand Visual Studio .NET's help features, and
5. Understand the use of various kinds of windows in the Visual Studio .NET IDE

Content

1.1 Introduction

Microsoft revolutionized the programming for Windows applications and became a bigger player in the development of Web applications with the introduction of the .NET Framework and Visual Studio (VS). Not only did .NET bring true object orientation to the language; it also provided great advances in the ease of developing projects for cross-platform compatibility. The two major parts of .NET are the Microsoft .NET Framework and the

Visual Studio integrated development environment (IDE). The IDE is used to develop programs and the Framework runs the program.

1.2 Basic Architecture of The Microsoft .NET Framework

The .NET Framework provides a platform for developing and running Windows applications and Web application. It consists of three main layer or parts namely common language runtime, class libraries, and user program Interfaces (ASP.NET and windows forms) as Figure 1.1.

ASP.NET: Web Services and Web Forms	Windows Forms
Framework Classes/ Libraries : ADO.NET	
Common Language Runtime	

Figure 1.1: The Architecture of .NET Framework

1.2.1 Common Language Runtime

The **common language runtime** (CLR) is an environment that manages execution of code. It provides services for tasks such as integrating components developed in different languages, handling errors across languages, providing security, and managing the storage and destruction of objects. Any code that is **compiled to run** in the CLR is called **managed code**. The managed code automatically contains **metadata**, which means data that describe data. A common language runtime **portable execution** (PE) file contains the metadata along with the code. The metadata include data types, members, references, and information needed to load classes and to call methods from a class. The CLR also manages memory used by .NET applications. Objects that are no longer being used are automatically removed from memory by the **garbage collector** component of the CLR. Code can be integrated with classes and methods of managed code written in other programming languages. The CLR has standards for data types that allow you to pass an instance of one of the classes to a method created in a different language.

1.2.2 Class Library

All of the .NET classes and interfaces are stored in a library known as the **.NET Framework class library**. The library is organized into sections or groups known as **namespaces**. Some of the common namespaces should be familiarize such as `System` and `System.Drawing`. Each namespace contains classes, structures, enumerations, delegates, and/or interfaces that can be used in a program. Table 1.1 below shows some of the namespaces in the .NET Framework class library.

Table 1.1: Selected Namespaces from the .NET Class Library

Namespace	Contents
System	Base classes and fundamentals classes for data types, events, and event handlers.
System.Collections	Definitions of collections of objects such as lists, queues, and dictionaries.
System.Data	ADO.NET architecture used to access databases.
System.Drawing	GDI+ graphics for drawing on the screen, printer, or any other graphic device.
System.IO	Types for reading and writing data streams and files.
System.Linq	Supports queries for Language-Integrated Queries.
System.Security	Base classes for permissions.
System.Threading	Classes for multithreaded programming.
System.Web.Services	Classes for building and using Web Services.
System.Windows.Forms	Classes for creating graphical components for programs that execute in the Window operating environment.
System.XML	Support for XML processing. XML is a standard for transferring data.

The classes in the library comply with published standards known as the **Common Language Specification** (CLS). The CLS specifies how a language that interacts with the CLR should behave. If you want a program to interact with programs and components written in other languages, you should make sure that it is CLS compliant. Note that all VB programs that you write using the VS IDE will be CLS compliant automatically.

The program code that you write is referred to as source code. The compiler translates your code into **Microsoft Intermediate Language** (MSIL) or sometimes referred as just IL. MSIL is a platform-independent set of instructions that is combined with the metadata to form a file called a **portable executable** (PE) file, which has an **.exe** or **.dll** extension. When your program runs, the MSIL is converted to the native code of the specific machine only as it is needed, using a just-in-time (JIT) compiler, which is part of the CLR (refer to Figure 1.2).

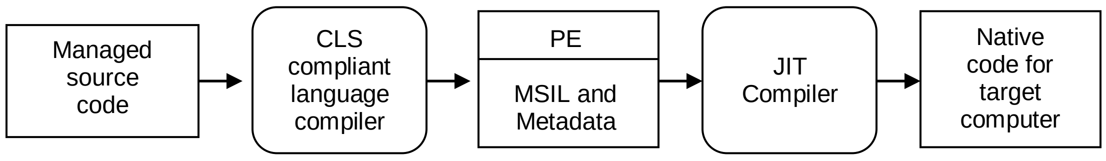

Managed source code
CLS compliant language compiler
PE
MSIL and Metadata
JIT Compiler
Native code for target computer

Figure 1.2: Steps in Compiling and Executing a Program Using .NET CLR

1.2.3 Windows Forms and ASP.NET

The Windows Forms environment simplifies the development of Windows user interface applications. Windows Forms uses the .NET platform offers full support for Web services and can connect easily to data stores by using a .NET data model called ADO.NET. The primary goals of Windows Forms are ease of use and reusability. The goal is to keep the process minimal and easy to coordinate. All the information needed for the project is contained in the code for the form.

Web Forms are the forms that are engine-supplied with ASP.NET. Their purpose is to provide Web browser-based user interfaces. This technology has been enhanced to provide the next generation of development by adding features such as drag-and-drop development. Web Forms consist of two parts: a template and a component. The template contains the layout information for the user interface elements. The component contains all the logic that is linked to the user interface. Web Forms take advantage of controls to make all the coordination required appear easy. Controls are reusable user interface elements that are used to construct the form.

In development environment, the design of windows forms is similar to Web Forms. Both environments have similar classes and some events in common, and the classes that are different are at least consistent in function and purpose. The benefits of the common language runtime and managed code are available for developing Windows applications and Web applications. Thus, developers can choose any one of the many languages. ASP.NET makes Web development easier by providing the same debugging support for Web Forms and Web Services as for Windows applications.

1.3 Important Features of Visual Studio IDE

An Integrated Development Environment (IDE) is software that facilitates application development. Some of the key features included are:

i. Single IDE for all .NET applications. Therefore no switching required to other IDEs for developing .NET applications
ii. Single .NET solution for an application which has been built on code written in multiple languages. Built-in languages include C++, Visual Basic, C# and F#.

iii. Single workspace consisting of a multiple-document interface in which activities related to code development such as editing, compiling, debugging, etc. is easily possible.

iv. Customizable environment to help the user to configure the IDE based on the required settings

v. Browser that is built-in within the IDE helps to view content from internet such as help, source-code, etc. in online mode.

vi. Code editor supporting IntelliSense and code refactoring. IntelliSense is implementation of code completion, best known in Visual Studio. It will speeds up the process of coding applications by reducing typos and other common mistakes.

vii. Visual Studio supports different programming languages and allows the code editor and debugger to support (to varying degrees) nearly any programming language, provided a language-specific service exists.

1.3.1 Quick Tour of the Visual Studio IDE

Visual Studio .NET is Microsoft's Integrated Development Environment (IDE) for creating, running and debugging programs (also called applications) written in a variety of .NET programming languages. This IDE is a powerful and sophisticated tool that is used to create applications with special features. Two types of applications that will be creating in this course are Windows Application and Web Based Windows Application**.** Web application uses a Browser such as Internet.

In this topic, an overview of the Visual Studio .NET IDE will be given and also demonstrate how to create simple program by dragging and dropping predefined building blocks into place – a technique called "visual programming". Once the Visual Studio .NET is executed, the Start Page will be displayed as shown in Figure 1.3. The Start Page might be slightly different depending on the version of Visual Studio used. In this module, Microsoft Visual Studio 2010 will be used.

When loading Visual Studio .NET for the first time, the list of Recent Projects will be empty. Regardless of the environment settings you selected, you will see the Start Page in the center of the screen. However, the contents of the Start Page and the surrounding toolbars and tool windows can vary. At this stage it is important to remember that your selection only determined the default settings, and that over time you can configure Visual Studio to suit your working styles.

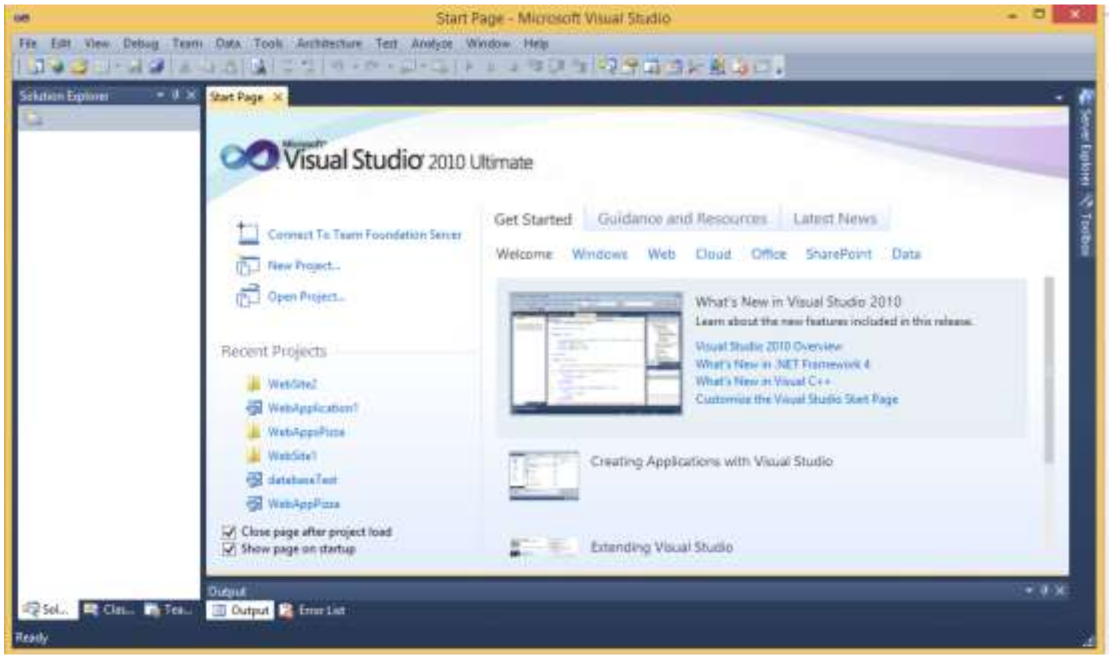

Start Page - Microsoft Visual Studio
File Edit View Debug Team Data Tools Architecture Test Analyze Window Help
Solution Explorer
Start Page
Visual Studio 2010 Ultimate
Connect To Team Foundation Server
New Project...
Open Project...
Recent Projects
WebSite2
WebApplication1
WebAppsPizza
WebSite1
databaseTest
WebAppPizza
Close page after project load
Show page on startup
Get Started
Guidance and Resources
Latest News
Welcome
Windows
Web
Cloud
Office
SharePoint
Data
What's New in Visual Studio 2010
Learn about the new features included in this release.
Visual Studio 2010 Overview
What's New in .NET Framework 4
What's New in Visual C++
Customize the Visual Studio Start Page
Creating Applications with Visual Studio
Extending Visual Studio
Output
Error List
Ready
Server Explorer
Toolbox

Figure 1.3: Start Page in Microsoft Visual Studio 2010

1.3.2 New Project Dialog

To create a new Visual Basic program, start by clicking **File** → **New** → **Project** as shown in Figure 1.4 below. Projects are groups of related files that form a Visual Basic program. The Visual Studio .NET IDE provides project types for a variety of programming languages such as Visual Basic, C# and C++.

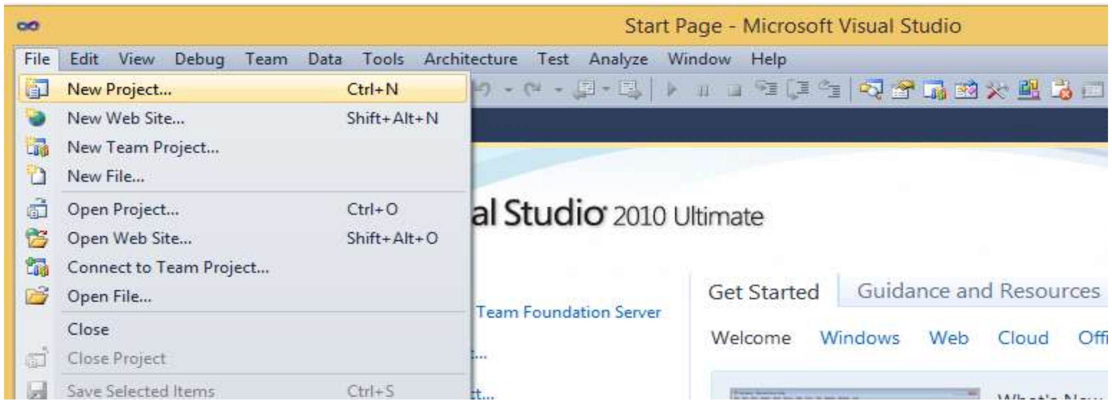
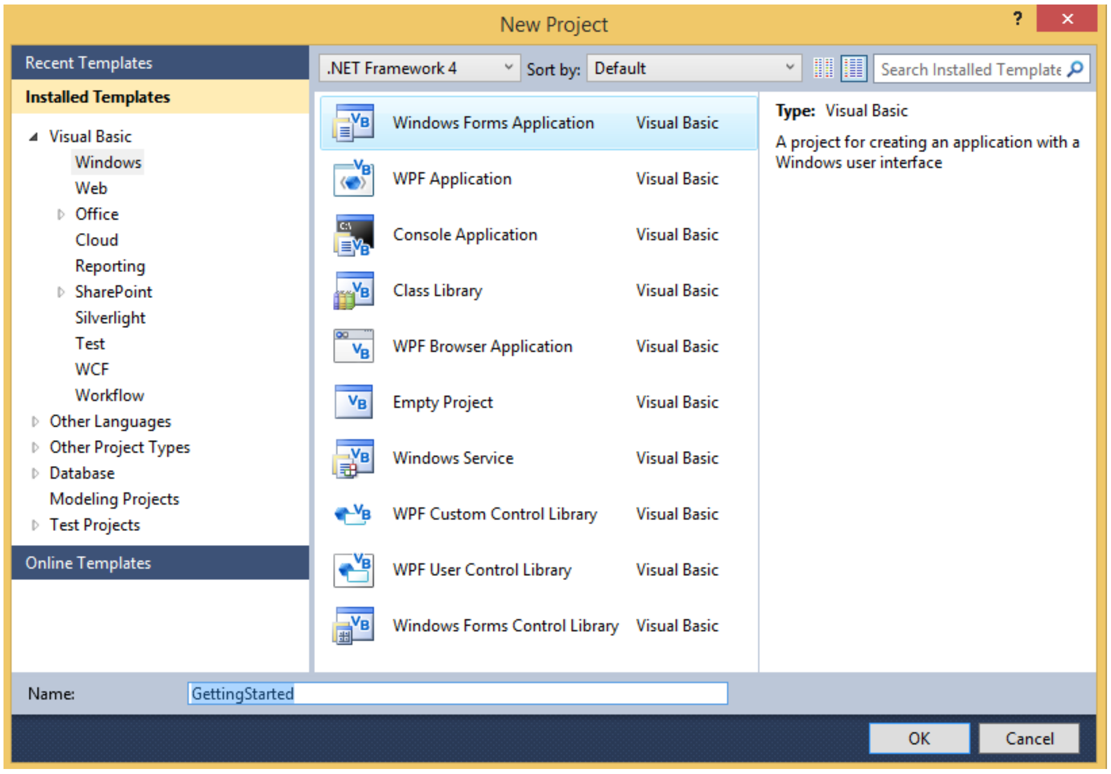

Start Page - Microsoft Visual Studio
File Edit View Debug Team Data Tools Architecture Test Analyze Window Help
New Project... Ctrl+N
New Web Site... Shift+Alt+N
New Team Project...
New File...
Open Project... Ctrl+O
Open Web Site... Shift+Alt+O
Connect to Team Project...
Open File...
Close
Close Project
Save Selected Items Ctrl+S
al Studio 2010 Ultimate
Team Foundation Server
Get Started
Guidance and Resources
Welcome
Windows
Web
Cloud

Figure 1.4: To Start Creating a New Project

This will open the **New Project dialog** as in Figure 1.5. A couple of new features are worth to mention here. Based on numerous feedback requests, this **dialog** is now **resizable**. More importantly, there is an additional drop-down box in the top right-hand corner, which is used to select the version on the .NET Framework that the application will target. The ability to use a single tool to create applications that target different framework versions means that developers can use fewer products and can take advantage of all the new features, even if they are maintaining an older product.



Windows Forms Application	Visual Basic
WPF Application	Visual Basic
Console Application	Visual Basic
Class Library	Visual Basic
WPF Browser Application	Visual Basic
Empty Project	Visual Basic
Windows Service	Visual Basic
WPF Custom Control Library	Visual Basic
WPF User Control Library	Visual Basic
Windows Forms Control Library	Visual Basic

Type: Visual Basic

A project for creating an application with a Windows user interface

Name: GettingStarted

Figure 1.5: New Project Dialog

Select the **Windows Forms Application** from the Templates area (this item exists under the root Visual Basic and Visual C# nodes, or under the sub-node Windows) and set the **Name** to “**GettingStarted**”, before selecting OK. This should create a new windows application project, which includes a single startup form and is contained within a “GettingStarted” solution, as shown in the **Solution Explorer window** of Figure 1.6. This startup form has automatically opened in the visual designer, giving you a graphical representation of what the form will look like when you run the application. You will notice that there is now an additional command bar visible and that the Properties tool window is in the right tool windows area.

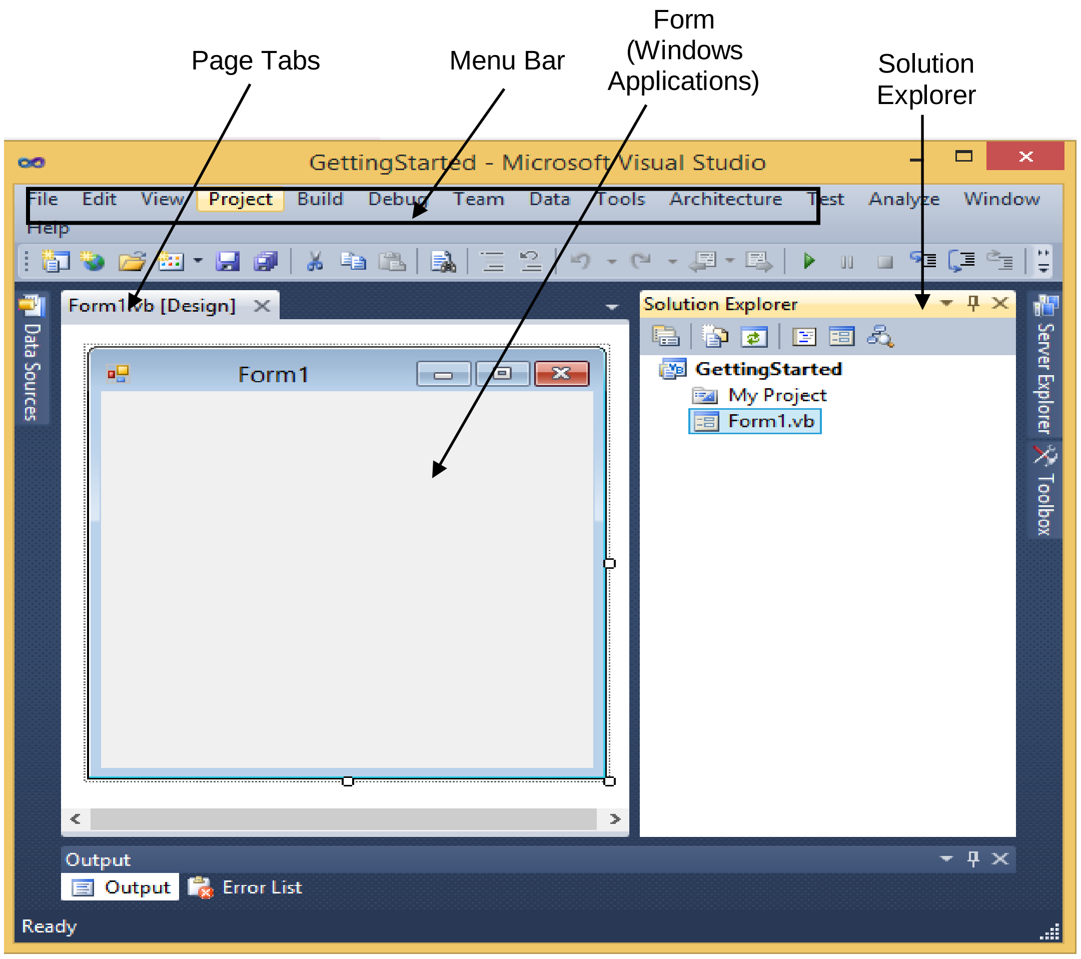


Figure 1.6: Design View of Visual Studio .NET IDE

The gray rectangle (called a form) titled **Form1** represents the Windows application that the programmer is creating. Collectively, the form and controls constitute the program's Graphical User Interface (GUI), which is the visual part of the program with which the user interacts. Users enter data (inputs) into the program by typing at the keyboard, by clicking the mouse buttons and in a variety of other ways. Programs display instructions and other information (outputs) for users to read in the GUI.

The name of each open document is listed on a tab: **Form1.vb[Design]** (upper left portion). To view a document, click its tab. Tabs save space and facilitate easy access to multiple documents. The active tab, or the tab of the document currently displayed in the IDE, is displayed in bold text and is positioned in front of all the other tabs.

1.3.3 Menu Bar and Toolbar

Commands for managing the IDE and for developing, maintaining and executing programs are contained in the menus, which are located on the **menu bar** (see Figure 1.7).

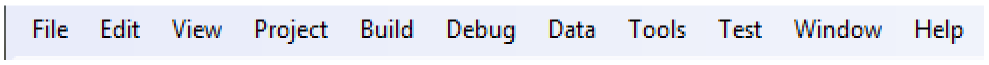


Figure 1.7: Visual Studio .NET IDE Menu Bar

Menus contain groups of related commands (also called menu items) that, when selected, cause the IDE to perform specific actions (e.g., open a window, save a file, print a file and execute a program). For example, new projects are created by selecting File → New → Project…. The menus depicted in Figure 1.7 are summarized in Table 1.2.

Table 1.2: Summary of Visual Studio .NET IDE Menus

Menu	Description
File	Contains commands for opening projects, closing projects, printing project data, etc.
Edit	Contains commands such as cut, paste, find, undo, etc.
View	Contains commands for displaying IDE windows and toolbars.
Project	Contains commands for managing a project and its files.
Build	Contains commands for compiling a program.
Debug	Contains commands for debugging (i.e., identifying and correcting problems in a program) and running a program.
Data	Contains commands for interacting with databases (i.e., files that store data)
Format	Contains commands for arranging and changing the appearance of a form's controls.
Tools	Contains commands for accessing additional IDE tools and options that enable customization of the IDE.
Windows	Contains commands for arranging and displaying windows.
Help	Contains commands for accessing the IDE's help features.

Rather than having to navigate the menus for certain commonly used commands, the programmer can access them from the toolbar (refer to

Figure 1.8), which contains pictures, called icons that graphically represent commands. To execute a command via the toolbar, click its icon. Some icons contain a down arrow that, when clicked, displays additional options.

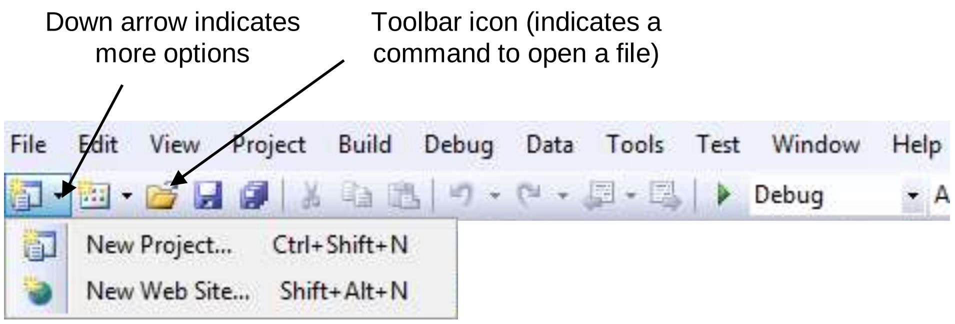

Down arrow indicates more options
Toolbar icon (indicates a command to open a file)
File Edit View Project Build Debug Data Tools Test Window Help
Debug
New Project... Ctrl+Shift+N
New Web Site... Shift+Alt+N

Figure 1.8: Toolbar Demonstration

Positioning the mouse pointer over an icon highlights the icon and, after a few seconds, displays a description called a tool tip (see Figure 1.9). Tool tips help novice programmers become familiar with the IDE’s features.

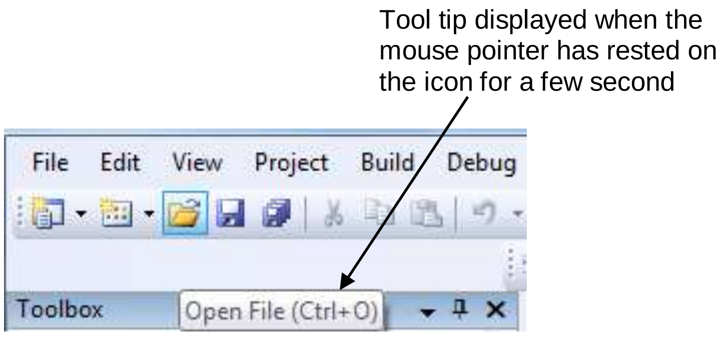

Tool tip displayed when the mouse pointer has rested on the icon for a few second
File Edit View Project Build Debug
Toolbox
Open File (Ctrl+O)

Figure 1.9: Tool Tip Demonstration

1.3.4 Help Menu

Visual Studio .NET IDE provides extensive help features. The Help menu contains a variety of commands, which are summarized in Table 1.3.

Table 1.3: Help Menu Commands

Command	Description
Contents	Displays a categorized table of contents in which help articles are organized by topic.
Index	Displays an alphabetized list of topics through which the programmer can browse.
Search	Allows programmers to find help articles based on search keywords.

Dynamic help (see Figure 1.10) is an excellent way to get information about the IDE and its features, as it provides a list of articles based on the current content (i.e., the items around the location of the mouse cursor). To open the Dynamic Help window (if it is not already open), select **Help** → **Dynamic Help**. Then when you click a word or component (such as a form or a control) in the IDE, links to relevant help articles appear in the Dynamic Help window. The window lists relevant help topics, samples and "Getting Started" information. There is also a toolbar that provides access to the Contents, Index and Search help features.

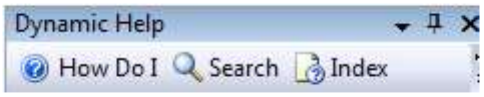


Figure 1.10: Dynamic Help Menu

1.4 Component of Visual Studio .NET IDE

The IDE provides windows for accessing project files and customizing controls. Several windows that are essential in the development of Visual Basic applications will be introduced. These windows can be accessed via the toolbar icons (refer to Figure 1.11) or by selecting the name of the desired window in the **View** menu.

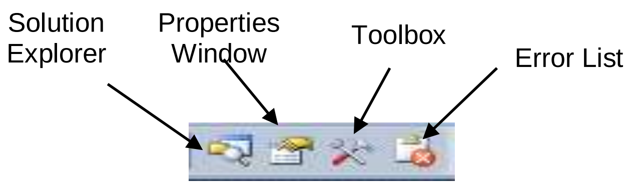


Figure 1.11: Toolbar Icons for Four Visual Studio .NET IDE Windows

Visual Studio .NET provides a **space-saving feature** called **auto-hide**, which can be activated by clicking the pin icon in the upper right corner of a window (see Figure 1.11).

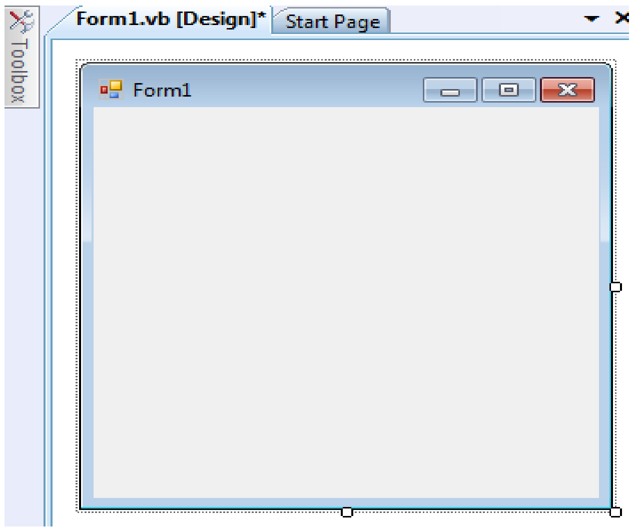


Figure 1.11: Auto-hide of Toolbox

When auto-hide is enabled, a toolbar appears along one of the edges of the IDE. This toolbar contains one or more icons, each of which identifies a hidden window. Placing the mouse pointer over one of these icons displays that window, but the window is hidden once the mouse pointer is moved outside the window's area. To "pin down" a window (i.e., to disable auto-hide and keep the window open), click the pin icon. Notice that when a window is "pinned down", the pin icon has a vertical orientation, whereas when auto-hide is enabled, the pin icon has a horizontal orientation (refer to Figure 1.12).

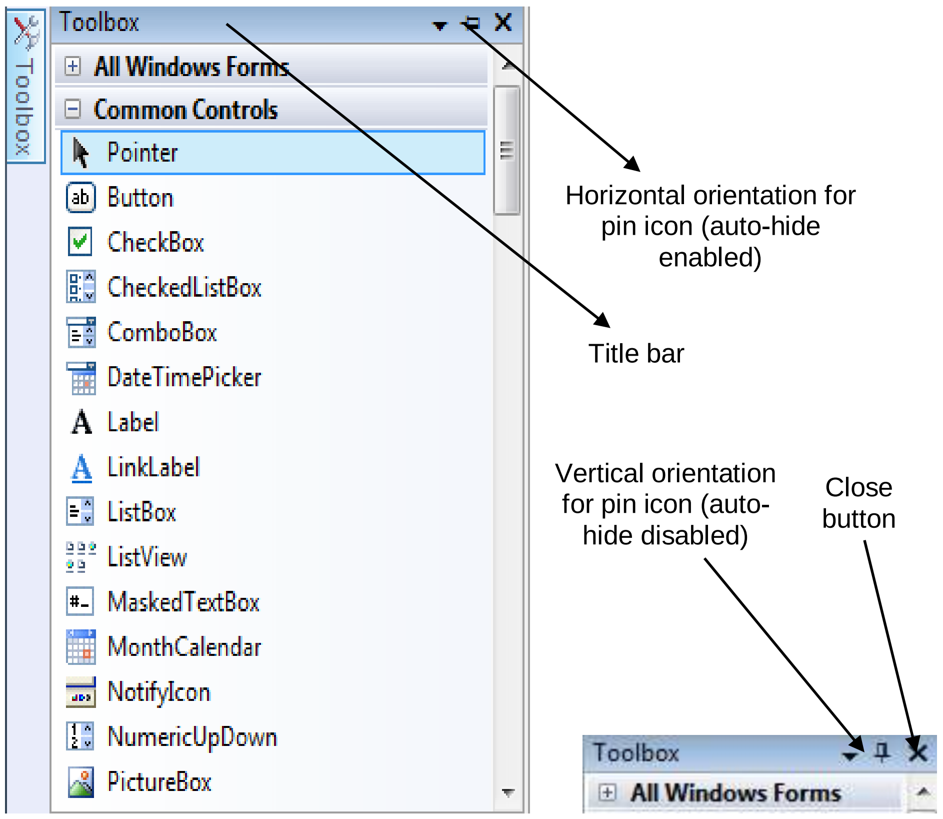


Figure 1.12: Demonstrating Auto-hide Feature

1.4.1 Solution Explorer

The Solution Explorer window (refer to Figure 1.13) provides access to all the files in the solution. When the Visual Studio .NET IDE is first loaded, the Solution Explorer is empty; there are no files to display. Once a solution is open, the Solution Explorer displays that solution's contents.

The solution's startup project is the project that runs when the program is executed and appears in bold text in the Solution Explorer. For single-project solution, the startup project is the only project (GettingStarted). The Visual Basic file, which corresponds to the form shown in Figure 1.6, is named **Form1.vb**. Visual Basic files use the .vb filename extension, which is short for "Visual Basic".

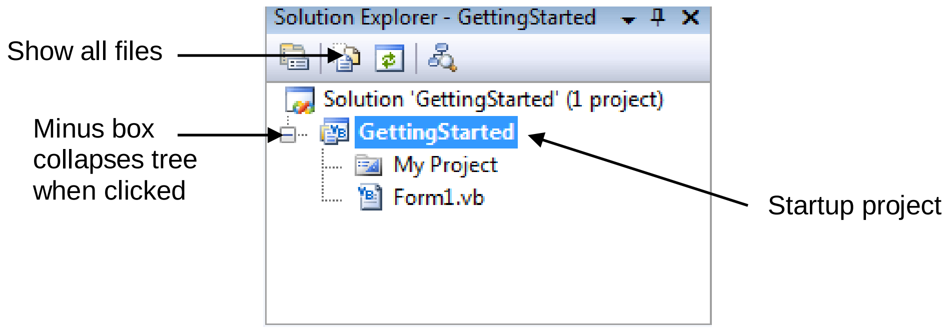

Show all files
Minus box collapses tree when clicked
Startup project
Solution Explorer - GettingStarted
Solution 'GettingStarted' (1 project)
GettingStarted
My Project
Form1.vb

Figure 1.13: Solution Explorer with an Open Solution

1.4.2 Toolbox

The **Toolbox** (see Figure 1.13) contains controls used to customize forms. Using visual programming, programmers can “drag and drop” controls onto the form instead of writing code to build them. Just as people do not need to know how to build an engine in order to drive a car, programmers do not need to know how to build a control in order to use the control. The use of preexisting controls enables developers to concentrate on the big picture, rather than the minute and complex details of every control. The wide variety of controls that are contained in the Toolbox is a powerful feature of the Visual Studio .NET IDE.

The **Toolbox** contains groups of related controls (e.g., Common Controls, Containers, Menus & Toolbars, Data and Components in Figure 1.14). When the name of a group is clicked, the list expands to display the various controls contained in the group. Users can scroll through the individual items by using the black scroll arrows to the right of the name of the group. The first item in the group is not a control – it is the mouse pointer. The mouse pointer is used to navigate the IDE and to manipulate a form and its control.

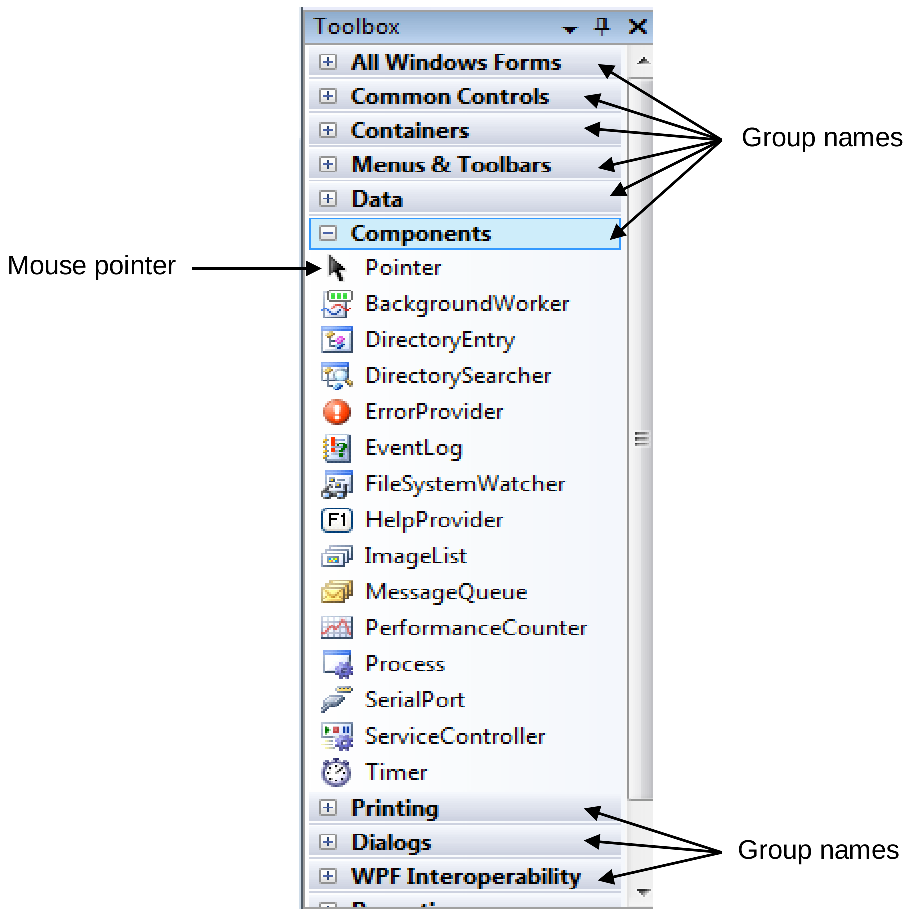
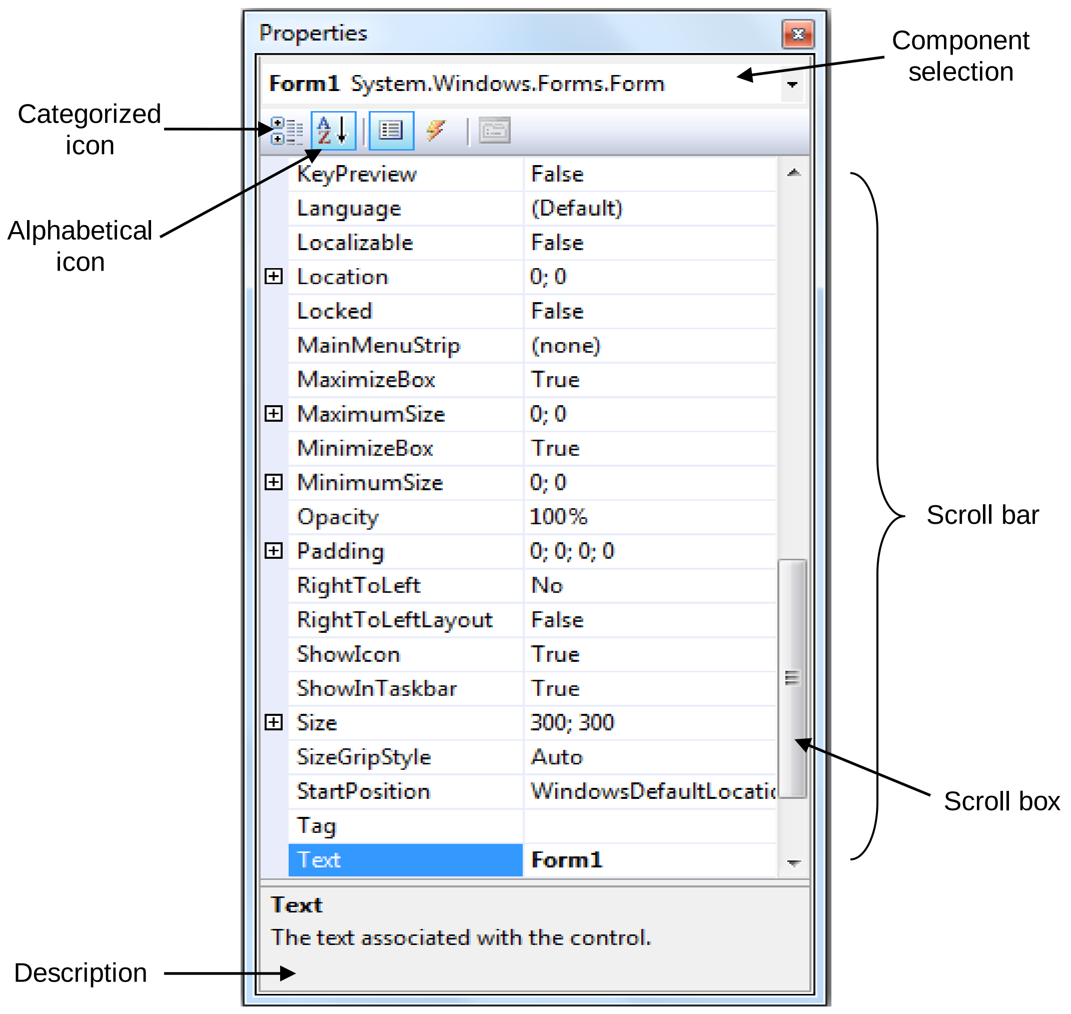


Figure 1.14: Toolbox Window

1.4.3 Properties Window

The **Properties** window (refer to Figure 1.15) displays the properties for a form or control. Properties specify information such as size, color and position. Each form or control has its own set of properties; a property's description is displayed at the bottom of the Properties window whenever that property is selected. If the Properties window is not visible, accessing **View** → **Properties Window**, or pressing **F4**, displays the Properties window. The left column of the Properties window lists the form's properties; the right column displays the current value of each property. Icons on the toolbar sort the properties either alphabetically (by clicking the **Alphabetic** icon) or categorically (by clicking the **Categorized** icon). Users can scroll through the list of properties by dragging the scrollbar's scrollbox up or down.



Properties

Form1 System.Windows.Forms.Form

KeyPreview	False
Language	(Default)
Localizable	False
⊞ Location	0; 0
Locked	False
MainMenuStrip	(none)
MaximizeBox	True
⊞ MaximumSize	0; 0
MinimizeBox	True
⊞ MinimumSize	0; 0
Opacity	100%
⊞ Padding	0; 0; 0; 0
RightToLeft	No
RightToLeftLayout	False
ShowIcon	True
ShowInTaskbar	True
⊞ Size	300; 300
SizeGripStyle	Auto
StartPosition	WindowsDefaultLocatio
Tag	
Text	**Form1**

Text
The text associated with the control.

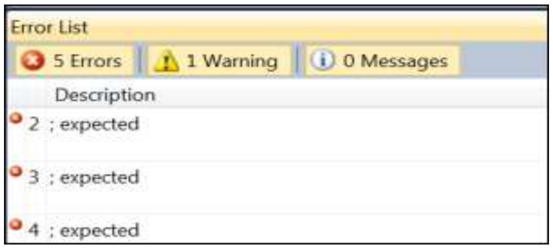
Figure 1.15: Properties Window

The Properties window is crucial to visual programming; it allows programmers to modify control visually, without writing code. This capability provides a number of benefits. First, programmers can see which properties are available for modification and, in many cases, can learn the range of acceptable values for a given property. Second, the programmer does not have to remember or search the Visual Studio .NET documentation for the possible settings of a particular property. Third, this window also displays a brief description of the selected property, helping programmers understand the property's purpose. Fourth, a property can be set quickly using this window. Usually, only a single click is required, and no code needs to be written. All of these features are designed to help programmers avoid repetitive tasks while ensuring that settings are correct and consistent throughout the project. At the top of the Properties window is the component selection drop-down list, which allows programmers to select the form or control whose properties are displayed in the Properties window. When a form or control in the list is clicked, the properties of that form or control appear in the Properties window.

1.4.4 Output Window and Error List Window

The Output Window will invoke when select Build|Build Solution at Menu Bar. The Output Window displays the results of the Compiler process. The program is fully compiled when all compiler errors are solved. Concurrently, the Error List window will show when there are build errors, warnings, and messages produced while developer write code (refer Figure 1.16). To display the Error List, click View / Error List**, or** CTRL+\+E. Using Error List Window, developer can perform the following tasks:

i. Find syntax errors noted by IntelliSense.
ii. Find deployment errors, certain Static Analysis errors, and errors detected while applying Enterprise Template policies.
iii. Double-click any error message entry to open the file where the problem occurs, and move to the error location.



2 ; expected
3 ; expected
4 ; expected

Figure 1.16: Error List Window

Activities

1. Fill in the blanks in each of the following statements:

   (a) The technique of ____________ allows programmers to create GUIs without writing any code.

   (b) _____________ are groups of related files that collectively form a Visual Basic program.

   (c) The ____________ feature hides a window when the mouse pointer is moved outside the window's area.

   (d) A _________ appears when the mouse pointer hovers over an icon.

   (e) The ___________ window allows programmers to browse solution files.

2. State whether each of the following is TRUE or FALSE, explain why.

   (a) The title bar displays the IDE's mode.

   (b) The x button toggles auto hide.

   (c) The toolbar icons represent various menu commands.

   (d) A form's sizing handles are always enabled.

   (e) Controls can be modified only by writing code.

3. What is meant by the term .NET Framework?

References

1. N. Randolph and D. Gardner, 2009. ***Professional Visual Studio 2008,*** Wiley Publishing.

2. D.I. Schneider, 2006. ***An Introduction to Programming Using Visual Basic 2005***. Prentice Hall.

3. J.C. Bradley and A.C. Millspaugh, 2010. ***Advanced Programming Using Visual Basic 2008***. Mc Graw Hill.

TOPIC 2

INTRODUCTION TO VISUAL BASIC .NET

Learning Outcome

At the end of this topic, student should be able to:

1. Explain the term event-driven and object-oriented programming,
2. Explain the concepts of classes, objects, properties, methods, and events,
3. Describe the three phase for develop a VB.Net project, and
4. Identify syntax errors, run-time errors, and logic errors.

Content

2.1 Introduction

Microsoft Windows uses a graphical user interface, or **GUI** (pronounced as “gooey”). The Windows GUI defines how the various elements look and function. Visual programming is useful for building GUI-intensive programs that require a large amount of user interaction. Some programs are designed not to interact with users and therefore do not have GUIs.

2.2 Event-driven and Object Oriented Programming Concept

Most **traditional languages**, such as **BASIC**, **C**, **COBOL**, **FORTRAN** and **Pascal**, are considered **procedural languages**. That is, the program **specifies** the **exact sequence** of all operations. Program logic determines the next instruction to execute in response to conditions and user requests.

The newer programming languages, such as **Visual Basic .NET**, **C#** and **Java**, use a different approach: **object-oriented programming** (OOP). As a stepping stone between procedural programming and object-oriented programming, the early versions of Visual Basic provided many (but not all) elements of an object-oriented language. For that reason, Microsoft referred to **Visual Basic** (**version 6 and earlier**) as an **event-driven programming** language rather than an object-oriented language. But with the release of Visual Studio .NET, which includes **Visual Basic .NET, C#,** and **J#,** Microsoft has produced three programming languages that are **truly object-oriented**.

In the OOP model, programs are **no longer procedural**. They **do not follow** a **sequential logic**. As the programmer, do not take control and determine the sequence of execution. Instead, the user can press keys and click various buttons and boxes in a window. Each user action can cause an event to occur, which triggers a basic procedure that have to be written. For example, the user clicks on a button labeled **Calculate**. The clicking causes the button's click event to occur, and the program automatically jumps to a procedure you have written to do the **calculation**. This means that a program's interface is comprised of objects (controls, forms, and so forth); the program is taught what actions to perform when events happen to those objects. An event is usually initiated action by the user. By anticipating the possible events that can occur to the various objects in program, programmer will **write the code to respond to those events appropriately.**

2.3 Object and Class

In Visual Basic .Net, developer will work with objects, which have properties, methods, and events. Each object is based on a class. A class is a template or blueprint used to create a new object.

Classes contain the definition of all available properties, methods, and events. When a new object created, it must be based on a class. For example, a developer may decide to place three buttons on a form. Each button is based on the Button class and is considered one object, called an instance of the class. Each button (for instance) has its own set of properties, methods, and events. One button may be labeled “OK”, one “Cancel”, and one “Exit”. When the user clicks the OK button, that button’s Click event occurs; if the user clicks on the Exit button, that button’s Click event occurs. And of course, different program instructions for each of the button’s click events have been written.

2.3.1 Basic Setting of Objects

Think of an object as a **thing**, or a **noun**. Examples of objects are **forms** and **controls**. Forms are the windows and dialog boxes that place on the screen; controls are the components place inside a form, such as text boxes, buttons, and list boxes. The GUI aspects of VB.NET programming involves the use of the various **controls** that are available in the toolbox. With controls come three basics setting which are **properties**, **methods**, and **events**.

2.3.1.1 Properties

Properties tell something about or control the appearance of an **object** such as its **name**, **color**, **size**, or **location**. Think of properties as **adjectives** that describe objects. Start set the properties by giving a unique name to the object. Then, set another related properties. For example, refer to the Text property of a form called salesForm as `salesForm.Text` (pronounced as *sales form dot text*)

2.3.1.2 Methods

Actions associated with objects are called methods. Methods are the verbs of object-oriented programming. It was procedures built into a control that tell it how to do things. Typical methods are **Close**, **Show**, and **Clear**. Each of the predefined objects has a set of methods that will be use. Refer to methods as Object.Method (*object dot method*). For example, a Show method can apply to different objects: `billingForm.Show` shows the form object called billingForm; `exitButton.Show` shows the button object called exitButton.

2.3.1.3 Events

Procedures that execute when a particular event occurs can be write by a programming statements. An event occurs when the user takes an action to an application, such as clicking a button, pressing a key, scrolling, or closing a window. Events also can be triggered by actions of other objects, such as repainting a form or a timer reaching a preset point. A programmer can write procedures that execute when a particular event occurs.

2.3.2 Works with Object Control

Visual Basic allows programmer to work with controls in two ways either at design time (visible Graphical Aspect) or run time (invisible aspect). Every design-time property for a given control by looking at the Properties window in the IDE (some controls have properties that are available only at run-time such as the *SelectionStart* and *SelectionLength* properties of the Textbox).

Table 2.1: Ways Use with Controls

Ways	Descriptions	Example
Working at design time	Controls are visible and work with them by dragging and dropping them from the Toolbox and setting their properties in the properties window.	Setting at Properties Window the Text of `btnOK` as Exit
Working at run time.	Controls are not visible while designing, are created and assigned properties in code and are visible only when the application is executed.	`btnOK.Text = "Exit"`

2.4 Phases in Develop an Applications to Solve a Problem

There are three phase involved in develop a VB .Net application: Analysis, Design and Implement. Each phase follow a three-step process. The three steps involve setting up the **user interface**, set the **properties**, and then the programming **code**.

2.4.1 Analysis

Analyzed the problem required to be solved. The analysis involved identify input or data to be entered or manipulate by application, identify output or results desired and required process to obtain the output.

2.4.2 Plan or Design

In this phase, the input, output and process listing in the analysis phase will be designed in order to solve a problem. A programmer will design the control flow of applications via Forms. In other words, the Form was the start up object of the application. There are three step used in design phase:

i. **Design the Graphical User Interface (GUI): Sketch** of the screen that user will see when running the project. On your sketch, show the forms and all the object controls that plan to use as input, output and process for the expected interactions between the user and application.

ii. **Plan the Properties:** For each object, indicate the names that you plan to give the form and each of the objects on the form. Then, write down the properties that you plan to set or change during the design of the form such as its appearances, layout and behaviors.

iii. **Algorithm:** Determine which events require action to be taken and then make a step-by-step plan for those actions. Include the plan of classes and procedures that will execute when the project runs.

A programmer need to create an algorithm using UML to illustrate the object model, class diagram and work flow.

2.4.2 Implement

After completing the planning steps and have approval from the user, begin the actual construction of the project. Use the same three-step process that used during design phase as below:

i. **Create the GUI**: Create the forms and placing controls that designed in the planning phase onto the Form.

ii. **Set the Properties:** Give each object a name and define such attributes as planning phase such as the contents of a label, the size of the text, and the words that appear on top of a button and in the form’s title bar.

iii. **Write the Programming Code:** Use the syntax or programming statements to carry out the actions needed by the applications. This involves coding in the Methods and Event Procedures in order to implement the algorithm.

After completing all the above steps, compile the program and resolve all Compiler Syntax Errors occur. Then, execute the program and resolve all Run Time Errors as well as Logical Errors.

2.5 Organize an Application Files

When creating a Visual Basic program a folder is automatically created in computer which contains the files of the program. A Visual Basics application is called a ***Solution***. The Solution is composed of one or more ***Project***. The Table 2.2 below lists several important files and their file extensions:

Table 2.2: Type of Important Visual Basic Project Files

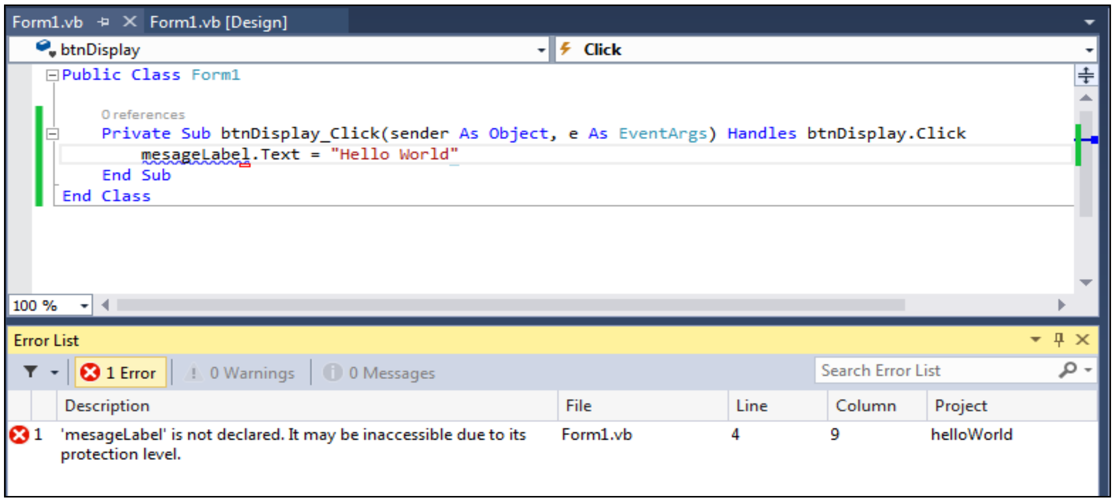
File Extension	Description	Example
.sln	Solution – File that holds information about all the projects in the application	**StudentSys.sln**
.vb	Object File – Contain definition and code for Forms, Modules, Class Modules etc.	**frmLogin.vb**
.resx	Resource File – This is a resource file for the Forms in the project. It contains information about all resources used by the form.	**frmLogin.resx**
.vbproj	Project File – This file describes the project and lists the files included in the project.	**StudentSys.vbproj**
.xsd	Dataset - A file for creating an XML schema with **DataSet** classes	**StudData.xsd**
.mdf	SQL Database	**StudDB.mdf**
.aspx	Web Form - A form for creating Web applications	**Home.aspx**
.master	A Master Page for Web Applications.	

2.3 Project Debugging

Programming errors come in three varieties: syntax errors, run-time errors, and logic errors. There is a situation when a program doesn't have errors, but that programs get “bugs” in them. Finding and fixing these bugs is called debugging. Fixing syntax errors and run-time errors are easier. Visual Basic .Net displays Editor Window with the offending line highlighted. However, a programmer must identify and locate logic errors himself. After locate the problem and fix it, recompile the program and run it again. Each time compiling the program, have a clean compile, which means zero errors. Programmer can use debugging tools such as breakpoints, watches and immediate windows to fix the error.

2.6.1 Syntax Errors

When a program breaks VB’s rules for punctuation, format, or spelling, it will generate a syntax error. The syntax errors that the editor cannot identify are found and reported by the compiler as it attempts to convert the code into intermediate machine language. A compiler-reported syntax error may be referred to as a compile error. The editor can correct some syntax errors by making assumptions, and not even report the error to you. For example, if the programmer type the opening quote of “Hello World” but forget the closing quote, the editor automatically adds the closing quote when move to the next line. And if forget the opening and closing parentheses after a method name, such as `Close()`, again, the editor will add them when move off the line. A blue squiggly line appears under the part of the line that the editor cannot interpret and a message appears in the Error list at the bottom of the screen (refer to Figure 2.1). Notice also that the Error list shows the line number of the statement that caused the error.



```
Public Class Form1

    Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
        mesageLabel.Text = "Hello World"
    End Sub
End Class
```

	Description	File	Line	Column	Project
1	'mesageLabel' is not declared. It may be inaccessible due to its protection level.	Form1.vb	4	9	helloWorld

Figure 2.1: The Editor Identifies a Syntax Error with a Squiggle Blue Line

2.6.2 Run Time Errors

If a project halts during execution, it is called a run-time error or an exception. Visual Basic displays a dialog box and highlights the statement causing the problem. Statements that cannot execute correctly cause run-time errors. The statements pass the syntax checking; however, the statements fail during execution. It cause by improper arithmetic operations such as calculating with nonnumeric data, dividing by zero, or finding the square root of a negative number and attempting to access resources that are not available. These problems are difficult to solve since they only show up when the program runs.

2.6.3 Logic Errors

When a program contains logic errors, the project runs but produces incorrect results or the project is not doing what is supposed to do. Perhaps the results of a calculation are incorrect or the wrong text appears or the text is okay but appears in the wrong location. Beginning programmers often overlook their logic errors. If the project runs, it must be right. All too often, that statement is not correct. These problems are even more difficult to solve since need to re-think and go back to the planning phase and review the algorithm. If involve a calculation, the programmer may need to use a calculator to check the output. Check all aspects of the project output such as computations, text, and spacing.

Activities

1. What are objects and properties? How are they related to each other?
2. What are the three steps for planning and creating Visual Basic projects? Describe what happens in each step?
3. What is the purpose of the Name property of a control?
4. What is meant by the term debugging?
5. What is a syntax error, when does it occur, and what might cause it?

References

1. N. Randolph and D. Gardner, 2009. ***Professional Visual Studio 2008,*** Wiley Publishing.
2. J.C. Bradley and A.C. Millspaugh, 2005. ***Programming in Visual Basic .NET***, McGraw Hill Technology Education.
3. D.I. Schneider, 2006. ***An Introduction to Programming Using Visual Basic 2005***. Prentice Hall.
4. File Types and File Extensions in Visual Basic and Visual C# retrieved from https://msdn.microsoft.com/en-us/library/

TOPIC 3

GRAPHICAL USER INTERFACE DESIGN

Learning Outcome

At the end of this topic, student should be able to:

1. Set a suitable conventions for object and use correct naming for each objects control,
2. Choose and use objects control effectively,
3. Use multiple statements for one control using programming code, and
4. Design a user interface with convenient features.

Content

3.1 Introduction

Controls are objects that can be inserted onto a form, simply by dragging the objects to enable or enhance user interaction with the application. Each control has unique properties, events, methods to use these controls effectively using code. The GUI aspects of VB.NET programming involve the use of the various controls that are available in the toolbox.

3.2 Common Properties of Object Control

Properties are a set of variables that describe the appearance, behavior, and other aspects of an object. In order to make VB easier to learn, Microsoft gave most of the controls and objects similar properties. For examples, most controls have Name Property (Name used in the Code to identify the Control), Text Property (Text displayed on the Control), Enabled Property (Indicates if Control is enabled or can be used) and Visible Property (Indicates if the Control can be seen). The descriptions of the each property will appear at below properties window when the property is clicked.

3.2.1 Objects Name

Consistent names for objects can make a project easier to read and understand, as well as easier to debug. Follow the VB rules for naming objects, procedures, and variables. In addition, conscientious programmers also follow certain naming conventions. A good programmer should have a set of standards and always follow them.

3.2.1.1 Naming Rules

When you select a name for an object, Visual Basic requires the name to **begin** with a **letter or** an **underscore**. The name can contain letters, digits, and underscores. An object name cannot include a space or punctuation mark and cannot be a reserved word, such as Exit or If, but can contain one. For example, **ExitCalculation** and **IfGood** are **legal**.

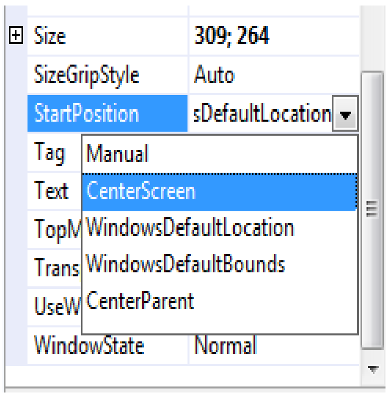
3.2.1.2 Naming Conventions

When naming controls, use **camel casing**, which means that you begin the name with a lowercase character and capitalize each additional word in the name. Make up a meaningful name and append the full name of the control's class. Do not use abbreviations unless it is a commonly used term that everyone will understand. All names must be meaningful and indicate the purpose of the object. Examples: **lblMessage**, **btnExit**, **frmDataEntry**, etc. Do not keep the default names assigned by Visual Basic, such as Button1 and Label1. Also, do not name your objects with numbers. The exception to this rule is for labels that never change during project execution. These labels usually hold items such as titles, instructions, and labels for other controls.

3.2.2 Object Appearance and Layout

Programmer can customize the form and all controls by changing some of the physical appearance and layout properties of the object. Common appearance properties for all objects such as background color, background image, border color, border style and fore color. For example, most controls can appear to be three-dimensional or flat. Labels, text boxes, and picture boxes all have a BorderStyle property with choices of **None**, **FixedSingle**, or **Fixed3D**. Text boxes default to Fixed3D; labels and picture boxes default to None. Other appearance setting will differ for each objects. For example, those objects use text will have appearance setting of text, text align, and font.

Object layout also have its setting such as Width, Height, Anchor, dock, location, size and etc. For example, to set **Form's Location on the Screen.** When a project runs, the form appears in the upper-left corner of the screen by default. The form's screen position can be set at the StartPosition property of the form. Figure 3.1 shows the choices for the property setting. To center you the form on the user's screen, set the StartPosition property to Center Screen.



⊞ Size	309; 264
SizeGripStyle	Auto
StartPosition	sDefaultLocation
Tag	
Text	
TopM	
Trans	
UseW	
WindowState	Normal

Figure 3.1: Start Position Property of the Form

3.2.3 Objects Behavior

Each object has its own behavior. The changes of the Behaviors will appear when the application is executed. Common behavior property used for all object is Enabled property. Another common behavior property which differ to other objects are context menu strip, tab index, tab stop and visible property.

As conclusion, not all controls have the same properties, but some properties are shared by many controls.

3.3 Basic Objects

In this topic, you will learn to use several control types: labels, buttons, text boxes, group boxes, check boxes, and radio buttons. Figure 3.2 shows the toolbox with the tools for these controls labeled. Each class of controls has its own set of properties. To see a complete list of the properties for any class of control, you can place a control on a form and examine the properties list or you can click on a tool or a control and press F1 for context-sensitive Help.

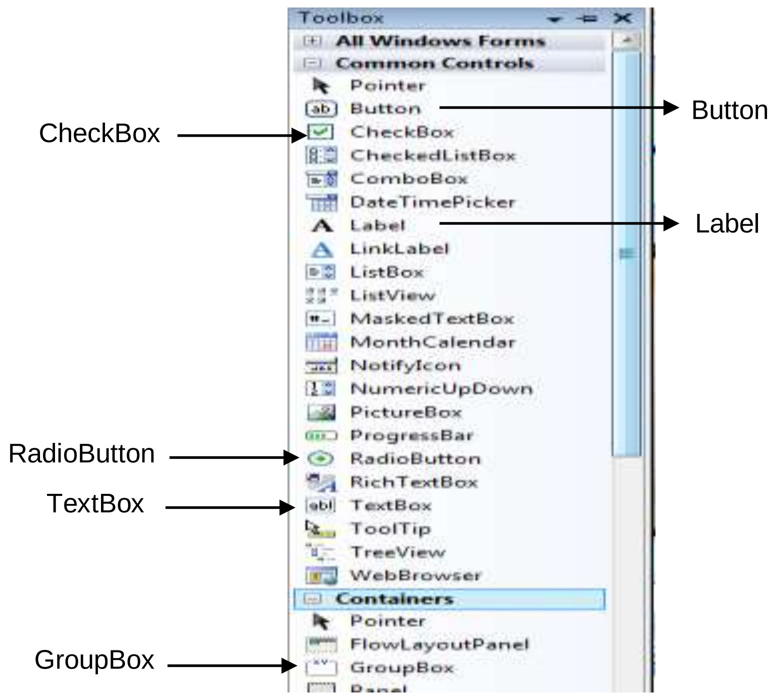


Figure 3.2: Toolbox Show Controls Covered in This Topic

3.3.1 Label and Button

Point to the **Label** tool in the toolbox and click. Then move the pointer over the form. Notice that the pointer becomes a crosshair with a big A, and the Label tool looks as if it has been pressed, indicating it is the active tool. The label has **eight** small **square handles**, indicating that the control is currently selected. Now, draw a button on the form (refer to Figure 3.3). Create another button using this alternative method: Point to the Button tool in the toolbox and double-click. A new button of the default size will appear on top of the last-drawn control (refer to Figure 3.4).

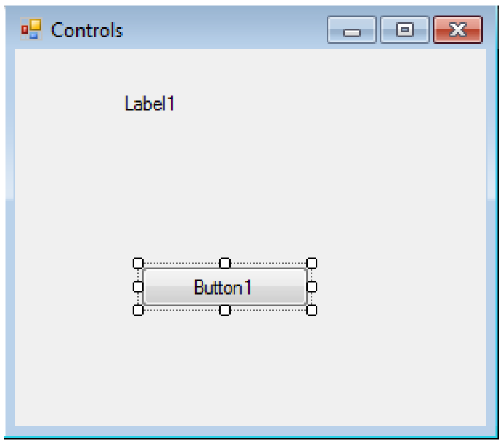


Figure 3.3: Label and Button on the Control Form

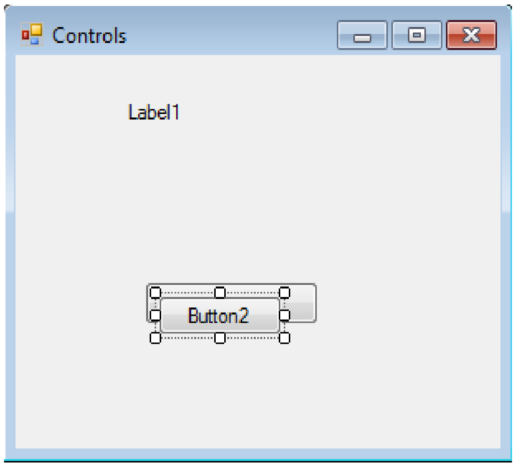


Figure 3.4: Creation of new button (Button2) appears on top of Button1

Select each control and move and resize the control as necessary. Make the two buttons the same size and line them up (refer to Figure 3.5). Point to one of the controls and click the right mouse button to display a context menu. On the context menu, select Lock Controls (see Figure 3.5). Locking prevents you from accidentally moving the controls. Please take note that you can unlock the controls at any time if you wish to redesign the form. Just click again on Lock Controls on the context menu to deselect.

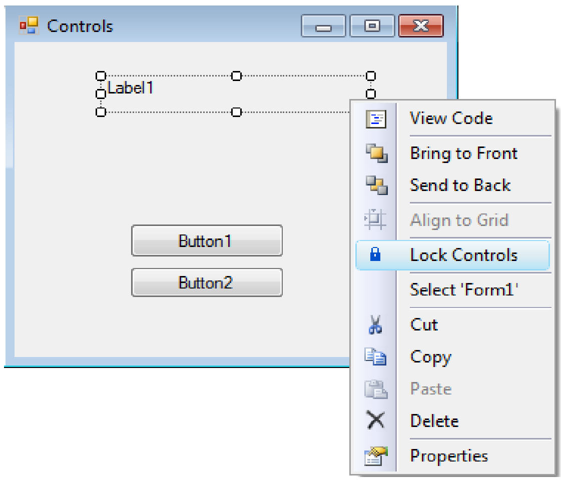
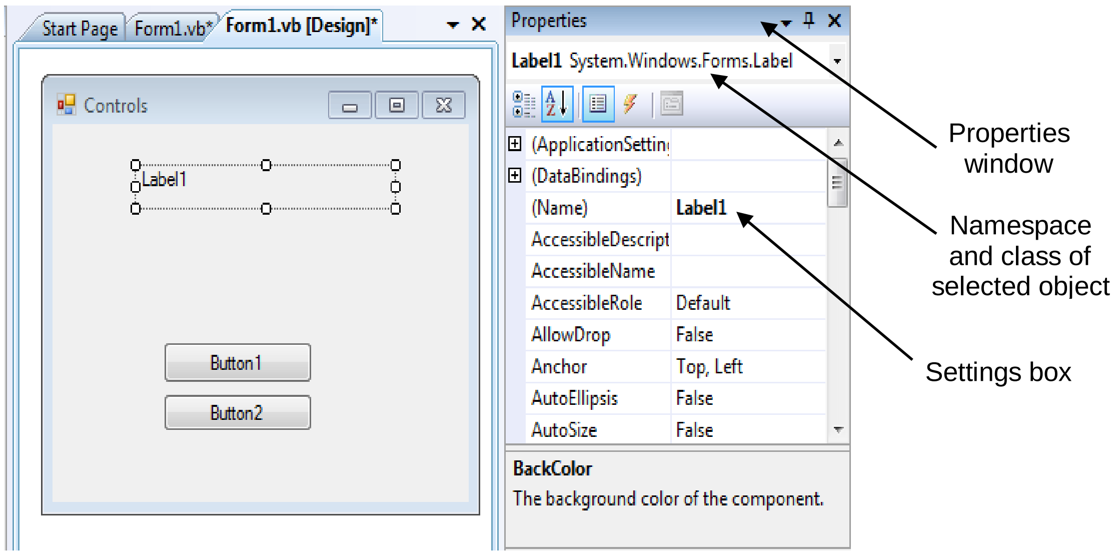


Figure 3.5: Controls are Placed into the Desired Location, Lock Them In Place By Selecting Lock Controls From The Context Menu

At this point you have designed the user interface and are ready to set the properties. Click on the label you placed on the form; a shaded outline appears around the control. Next click on the title bar of the Properties window to make it the active window (refer to Figure 3.6).



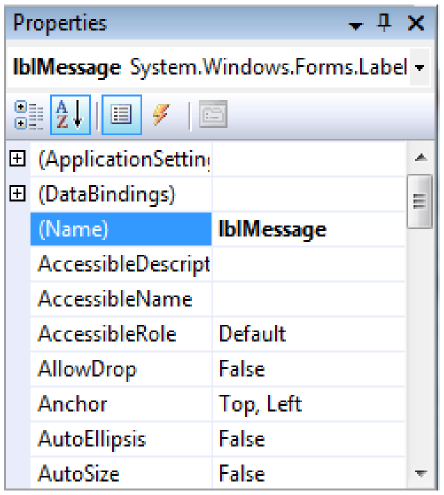
Label1 System.Windows.Forms.Label

(ApplicationSettin	
(DataBindings)	
(Name)	Label1
AccessibleDescript	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
Anchor	Top, Left
AutoEllipsis	False
AutoSize	False

BackColor
The background color of the component.

Figure 3.6: Currently selected control is shown in the Properties window

Select the **Name** property. You may have to scroll up; Name is located near the top of the list. Click on Name and notice that the Settings box shows **Label1**, the default name of the label. Change **Label1** to **lblMessage** (refer to Figure 3.7).

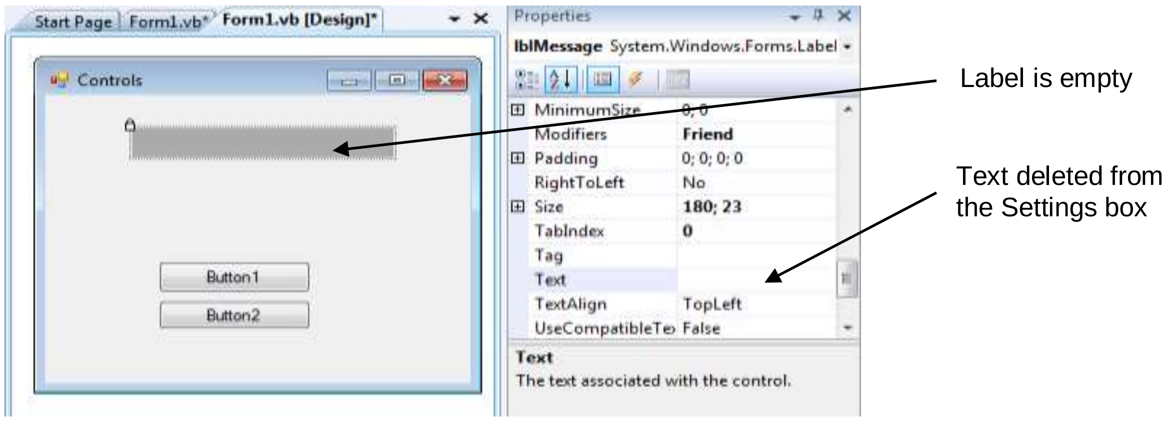

Properties

lblMessage System.Windows.Forms.Label

⊞ (ApplicationSettin	
⊞ (DataBindings)	
(Name)	lblMessage
AccessibleDescript	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
Anchor	Top, Left
AutoEllipsis	False
AutoSize	False

Figure 3.7: Type lblMessage into the Settings Box for Name Property

Click on the Text property to select it. The Text property of a control determines what will be displayed on the form. Because nothing should display when the program begins, you must delete the value of the Text property (see Figure 3.8).



Properties

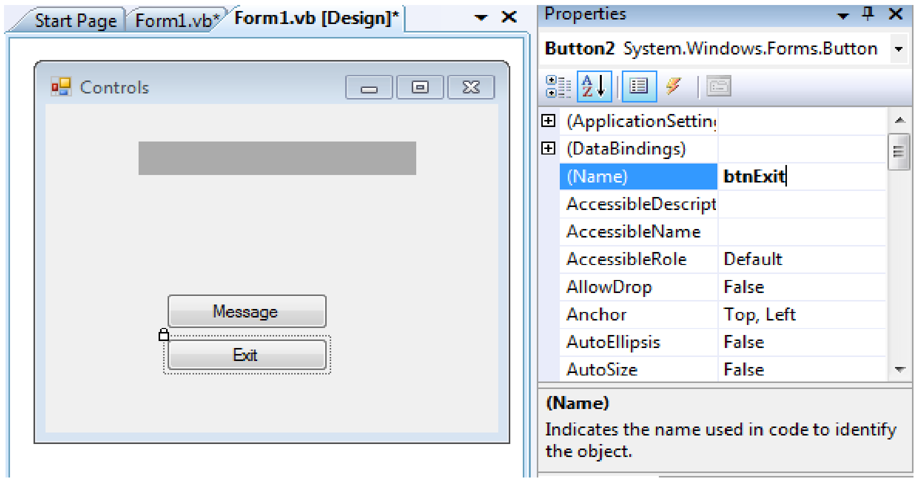
lblMessage System.Windows.Forms.Label

⊞ MinimumSize	0; 0
Modifiers	Friend
⊞ Padding	0; 0; 0; 0
RightToLeft	No
⊞ Size	180; 23
TabIndex	0
Tag	
Text	
TextAlign	TopLeft
UseCompatibleTe	False

Text
The text associated with the control.

Figure 3.8: Delete the Value for the Text; the Label on the Form Also Appears Empty

Now, click Button1 to select it and then look at the Properties window. The Object box should show the name **Button1** and class **System.Windows.Forms.Button**. Change the Name property of the button to **btnMessage** and the Text property to **Message**. Select Button2 and change its Name property to **btnExit** and the Text property to **Exit**. Refer to Figure 3.9.



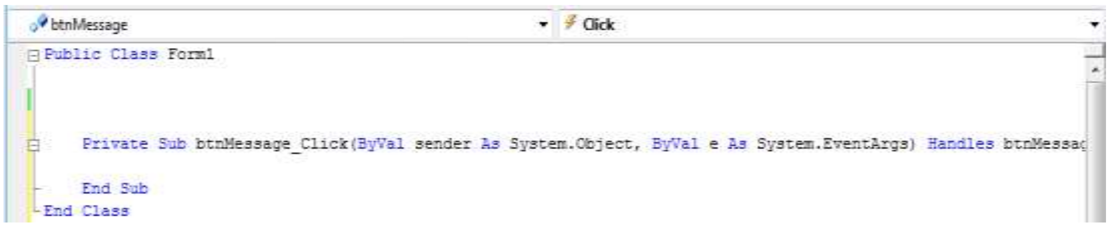
Button2 System.Windows.Forms.Button

(ApplicationSettin	
(DataBindings)	
(Name)	btnExit
AccessibleDescript	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
Anchor	Top, Left
AutoEllipsis	False
AutoSize	False

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

Figure 3.9: The New Layout of Form

Now you are ready to write Visual Basic .NET command codes for these buttons. Double click on the Message button. The Visual Studio editor opens with the first and last lines of your sub procedure already in place, with the insertion point indented inside the sub procedure (refer to Figure 3.10).



```
Public Class Form1


    Private Sub btnMessage_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnMessag

    End Sub
End Class
```

Figure 3.10: The Editor Window Showing the First and Last Lines of the `btnMessage_Click` Sub Procedures

The program code will be written in Visual Basic in procedures. For now, each of your procedures will be a sub procedure, which begins with the words `Private Sub` and ends with `End Sub`.

The project requires two Visual Basic statements: the **remark** and the **assignment statement**. Remark statements, sometimes called **comments** are used for **project documentation** only. They are not executable and have no effect when the project runs. The purpose of remarks is to make the project more readable and understandable by the people who read it. Remarks begin with an apostrophe (‘). Most of the time, remarks will be on a separate line that starts with an apostrophe. You can also add an apostrophe and a remark to the right end of a line of code.

Examples:

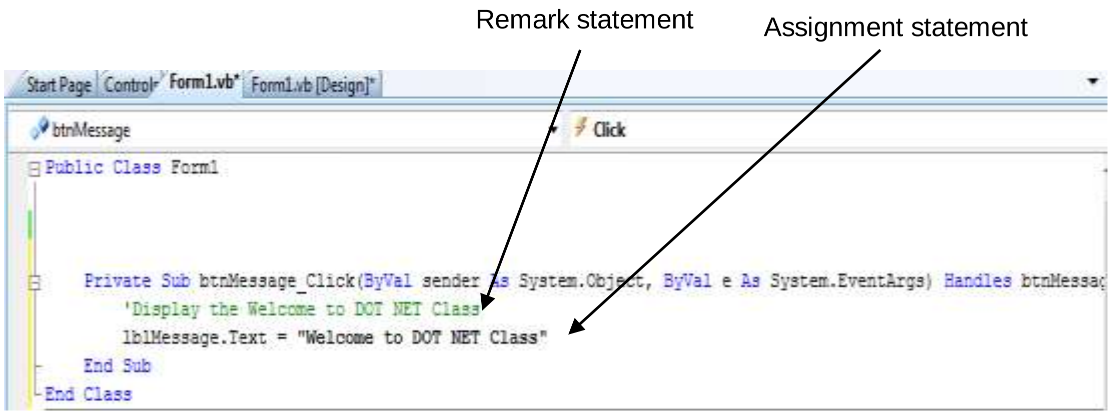
```
‘This project was written by Rafizah Mohd Hanifa
lblMessage.Text = “Welcome to DOT NET Class”  ‘Assign message to Text
```

The assignment statement assigns a value to a property or variable. Assignment statements operate from right to left; that is, the value appearing on the right side of the equal sign is assigned to the property named on the left of the equal sign. It is often helpful to read the equal sign as “is replaced by”. For example, the assignment statement in the example would read “`lblMessage.Text` is replaced by Welcome to DOT NET Class”.

Examples:

```
lblAddress.Text = “No. 3 Jalan Saujana Bestari 1”
intAge = 41;
```

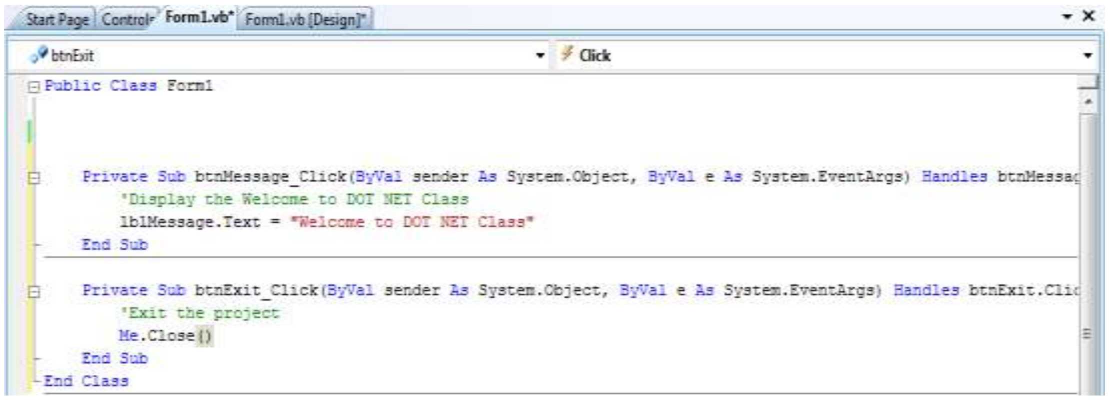
Now, you are ready to complete your project. Type remark statement in your `btnMessage_Click`: **`‘Display the Welcome to DOT NET Class`**’ message. Notice that the editor automatically displays remarks in green (unless you changed the color with the Environment option). Then, type the following assignment statement: `lblMessage.Text = "Welcome to DOT NET Class"`.



```
Public Class Form1



    Private Sub btnMessage_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnMessag
        'Display the Welcome to DOT NET Class
        lblMessage.Text = "Welcome to DOT NET Class"
    End Sub
End Class
```

Figure 3.11: Remark and Assignment Statement for the `btnMessage_Click` Event Sub Procedures

Now, `btnMessage_Click` have been completed. Double click on the Exit button to open the editor for the `btnExit_Click` event. Type this remark: **'Exit the project** and type this Basic statement: **`Me.Close( )`** (refer to Figure 3.12).



```
Public Class Form1


    Private Sub btnMessage_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnMessag
        'Display the Welcome to DOT NET Class
        lblMessage.Text = "Welcome to DOT NET Class"
    End Sub

    Private Sub btnExit_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnExit.Clic
        'Exit the project
        Me.Close()
    End Sub
End Class
```

Figure 3.12: Code for the `btnExit_Click` Event Procedure

You have finished writing the code and ready to run the project. Use one of these three methods:

1. Open the Debug menu and choose Start.

2. Press the Start button on the toolbar.
3. Press F5, the shortcut key for the Start command.

If all went well, the Visual Studio title bar now indicates that you are in run time. Click the Message button. Your "Welcome to DOT NET Class" message appears in the label as shown in Figure 3.13.

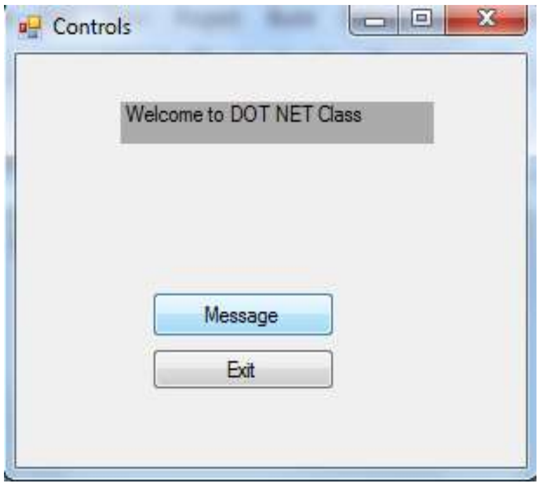


Figure 3.13: Click on the Message Button and "Welcome to DOT NET Class" appears in the Label

3.3.2 Text Boxes

Use a text box control when you want the user to type some input. The user can move from one box to the next, make corrections, cut and paste if desired, and click the Display button when finished. In your program code, you can use the Text property of each text box.

Example: **`lblName.Text = txtName.Text`**

In the previous example, whatever the user enters into the text box is assigned to the Text property of lblName. If you want to display some text in a text box during program execution assign a literal to the Text property:

```
txtMessage.Text = "Samir, come here."
```

You can set the **TextAlign property** of text boxes to change the alignment of text within the box. The values for the TextAlign property are:

```
HorizontalAlignment.Left
HorizontalAlignment.Right
HorizontalAlignment.Center
```

The three-letter prefix for naming a text box is “**txt**”.
Examples: **`txtName,txtAddress`**

3.3.3 Group Boxes

Group boxes are used as containers for other controls. Usually, groups of radio buttons or check boxes are placed in group boxes. Using group boxes to group controls makes your forms easier to understand. Set a group box’s Text property to the words you want to appear on the top edge of the group box. The three-letter prefix for naming a group is “**grp**”.
Example: **`grpColor`**

3.3.4 Check Boxes

Check boxes allow the user to select (or deselect) an option. In any group of check boxes, any number can be selected. The Checked property of a check box is set to **False** if **unchecked** or **True** if **checked**. You can write an event procedure for the CheckChanged event, which executes when the user clicks in the box. Use the Text property of a check box for the text you want to appear next to the box. The three-letter prefix for naming a check box is “**chk**”.
Examples: **`chkRed,chkBlue`**

3.3.5 Radio Buttons

Use radio buttons when **only one button** of a group may be **selected**. Any radio buttons that you place directly on the form (not in a group box) function as a group. A group of radio buttons inside a group box function together. The best method is to first create a group box and then create each radio button inside the group box. The Checked property of a radio button is set to **True** if **selected** or to **False** if **unselected**. You can write an event procedure to execute when the user selects a radio button using the control's CheckChanged event. Set a radio button's Text property to the text you want to appear next to the button. The three-letter prefix for naming a radio button is "rad".

Examples: `radHotel, radApartment`

3.4 Properties Setting of the Controls at Run Time

An initial properties have been set for controls at design time. Some setting of properties can be done in code, as your project executes. It is called design at run time for example clear out the contents of text boxes and labels; reset the focus; and change the color of text.

3.4.1 Clearing Text Boxes and Labels

In order to clear out the contents of a text box or label setting the property to an empty string, use "" (no space between the two quotation marks). This empty string is also called a null string or a zero-length string. It also clear out a text box using the Clear method.

Examples:

```
txtName.Text = ""          'Clear the contents
lblMessage.Text = ""       'Clear the contents
txtDataEntry.Clear ()      'Clear the contents
```

3.4.2 Resetting the Focus

As program runs, if the insertion point want to appear in the text box where the user is expected to type. The focus should therefore begin in the first text box. But what about later? When clear the form's text boxes, reset the focus to the first text box. The Focus method handles this situation. Remember, the convention is Object.Method. If the statement to set the insertion point in the text box called txtName. Thus, the code will be as follows:

```
txtName.Focus()       'Make the insertion point appear here
```

3.4.3 Setting the Checked Property of Radio Buttons and Check Boxes

The purpose of radio buttons and check boxes is to allow the user to make selections. However, often select or deselect a control is needed in program code. The selection or deselection radio buttons and check boxes can be done at design time (to set initial status) or at run time (to respond to an event). To make a radio button appear selected initially, set its Checked property to True in the Properties window. In code, assign True to its Checked property.

Examples:

```
radRed.Checked = True                'Make button selected
chkDisplay.Checked = True            'Make box checked
chkVisualBasic.Checked = False       'Make box unchecked
```

3.4.4 Changing the Color of Text

The color of text can be change by changing the ForeColor property of a control. Actually, most controls have a ForeColor and a BackColor property. The ForeColor property changes the color of the text; the BackColor property controls the color around the text. Visual Basic provides an easy way to specify a large number of colors. These color constants are in the Color class. When type the keyword Color and a period in the editor, a full list of colors will be appear. Some of the colors are listed below:

```
Color.AliceBlue
Color.AntiqueWhite
Color.Bisque
Color.BlanchedAlmond
Color.Blue
```

To change the control's fore color at run time, a code statement can be write as examples:

```
txtName.ForeColor = Color.Red
lblMessage.ForeColor = Color.Blue
```

3.4.5 Changing Multiple Properties of a Control

There are some situation is needed to change several properties of a single control. In previous version of Visual Basic a programmer had to write out the entire name (Object.Property) for each statement.

Examples:

```
txtTitle.Visible = True
txtTitle.ForeColor = Color.Blue
txtTitle.Focus( )
```

The statements still can be specify, but Visual Basic provides a better way: the `With` and `End With` statements.

Example:

```
With txtTitle
      .Visible = True
      .ForeColor = Color.Blue
      .Focus( )
End With
```

The statements beginning with `With` and ending with `End With` are called a With block. The statements inside the block are indented for readability. Although indentation is not required by VB, it is required by good programming practices and aids in readability. The real advantage of using the With statement, rather than spelling out the object for each statement, is that With is more efficient. Your Visual Basic projects will run a little faster if you use With. On a large, complicated project, the savings can be significant.

3.5 User Interface Design with Convenience Features

One of the good programming goal is to create programs that easy to use. The user interface should be clear and consistent. One school of thought says that if users misuse a program, it’s the fault of the programmer, not the users. Because most of users will already know how to operate Windows programs, programmer should strive to make your programs look and behave like other Windows programs. Some of the ways to accomplish this are to make the controls operate in the standard way, define keyboard access keys, set a default button, and make the Tab key work correctly. A programmer can also define ToolTips, which are those small labels that pop up when the user pause the mouse pointer over a control.

3.5.1 General Principles in Designing the User Interface

The design of the screen should be easy to understand and comfortable for the user. The best way that we can accomplish these goals is to follow industry standards for the color, size, and placement of controls. Once users become accustomed to a screen design, they will expect (and feel more familiar with) that the applications follow the same design criteria. A programmer should design the applications to match other Windows applications. Microsoft has done extensive program testing with users of different ages, genders, nationalities, and disabilities.

One recommendation about interface design concerns color. A programmer have probably noticed that Windows applications are predominantly gray. A reason for this choice is that many people are color blind. Also, gray is easiest for the majority of users. Although you may personally prefer brighter colors, you will stick with gray, or the system palette the user chooses, if you want your applications to look professional. Colors can indicate to the user what is expected. Use a white background for text boxes to indicate that the user should input information. Use a gray background for labels, which the user cannot change. Labels that will display a message or the result of a calculation should have a border around them; labels that provide text on the screen should have no border (the default).

Group the controls on the form to aid the user. A good practice is to create group boxes to hold related items, especially those controls that require user input. This visual aid helps the user understand the information that is being presented or requested. Use a Sans Serif font on your forms, such as the default MS Sans Serif, and do not make them boldface. Limit large font sizes to a few items, such as the company name.

3.5.2 Keyboard Access Keys

Windows is set up so that most functions can be done with either the keyboard or a mouse. Make a project respond to the keyboard by defining access keys, also called hot keys. For example, in Figure 3.14 select the **Message** button with **Alt + m** and the **Exit** button with **Alt + x**.

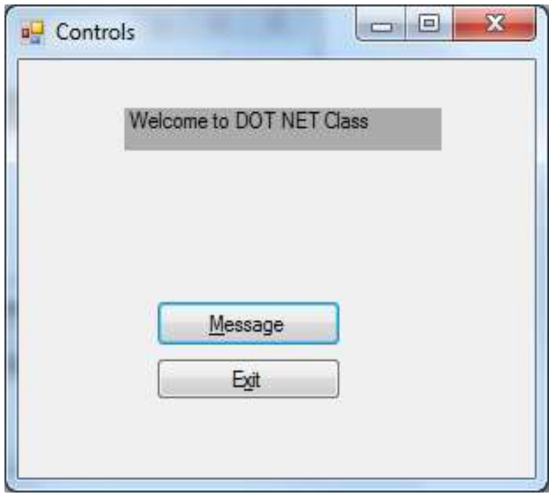


Figure 3.14: The Underlined Character Defines an Access Key

Access keys can be set for buttons, radio buttons, and check boxes when define their Text properties. Type an ampersand (&) in front of the character you want for the access key; Visual Basic underlines the character.

Examples:

&Message for <u>M</u>essage
E&xit for E<u>x</u>it

When defining the access keys, watch for several pitfalls. First, try to use the Windows-standard keys whenever possible. For example, use the <u>x</u> of Exit and the <u>S</u> of Save. Second, don't use two controls with the same access key. It confuses the user and doesn't work correctly. Only the first control is activated when the user presses the access key.

3.5.3 Default and Cancel Properties of Buttons

Once a person's fingers are on the keyboard, most people prefer to press the Enter key rather than to click the mouse. If one of the buttons on the form is the default button, pressing Enter is the same as clicking the button. Always identify the default button on a form by its darker outline.

3.5.4 Tab Order for Controls

In Windows programs, one control on the form always has the focus. See the focus change of Tab from control to control. For controls such as buttons, the focus appears as a light dotted line. For text boxes, the insertion point (also called the cursor) appears inside the box. Some controls can receive the focus; others cannot. For example, text boxes and buttons can receive the focus, but labels and picture boxes cannot. Two properties determine whether the focus stops on a control and the order in which the focus moves. Controls that are capable of receiving focus have a TabStop property, which can set to True or False. If you **do not want the focus** to stop on a control when the user presses the Tab key, set the **TabStop** property to **False**. The **TabIndex** property determines the **order** the focus moves as the Tab key is pressed. As controls created on the form, Visual Basic assigns the TabIndex property in sequence. Most of the time that order is correct, but if want to Tab in some other sequence or if add controls later, modify the TabIndex properties of your controls. When your program begins running, the focus is on the control with the lowest TabIndex (usually 0). Since you generally want the insertion point to appear in the first control on the form, its TabIndex should be set to 0. The next control should be set to 1; the next to 2; and so forth. By default, buttons and text boxes have their TabStop property set to True, but radio buttons have their TabStop property set to False. If the tab sequence are include radio buttons, set their TabStop property to True. Be aware that the behavior of radio buttons in the tab sequence is different from other controls: The Tab key takes you only to one radio button in a group, even though all buttons in the group have their TabStop and TabIndex properties set. If use the keyboard to select radio buttons, tab to the group and then use Up and Down arrow keys to select the correct button.

3.5.6 ToolTips

ToolTips are small labels that pop up when you pause your pointer over a toolbar button or control. It can be easily add ToolTips to a project by adding a ToolTips control to a form. After add the control to your form, each of the form's controls has a new property: ToolTip on ToolTip1, assuming that you keep the default name, ToolTip1, for the control. To define ToolTips, select the ToolTip control from the toolbox and click anywhere on the form. The new control appears in a new pane that opens at the bottom of the Form Designer. This pane, called the **component tray**, holds controls that do not have a visual representation at run time. After add the ToolTip control, examine the properties list for other controls on the form, such as buttons, text boxes, labels, radio buttons, check boxes, and even the form itself. Each has a new ToolTip on ToolTip1 property.

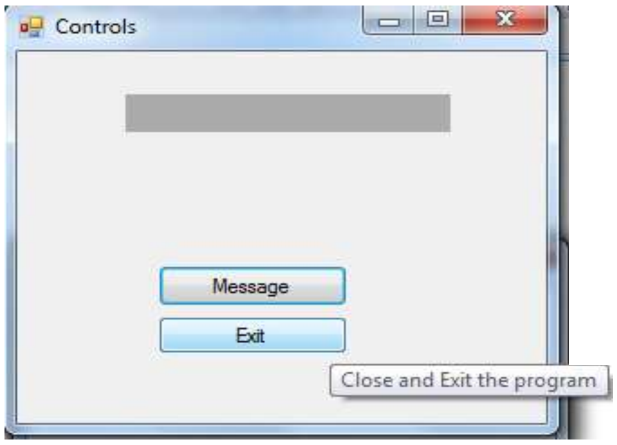


Figure 3.15: Use the ToolTipText in ToolTip1 Property to Define a Tooltip

3.4 Menus and Submenus

Menus very useful for make multiple form in an application more organized and easy to navigate.

3.5.1 Menus

This type of menus are very common to Windows Applications. Visual Basic itself has many of these drop down menus such as File, Edit, View, Project, Format, etc. Use the toolbox to add a MenuStrip control in a form (Figure 3.16).

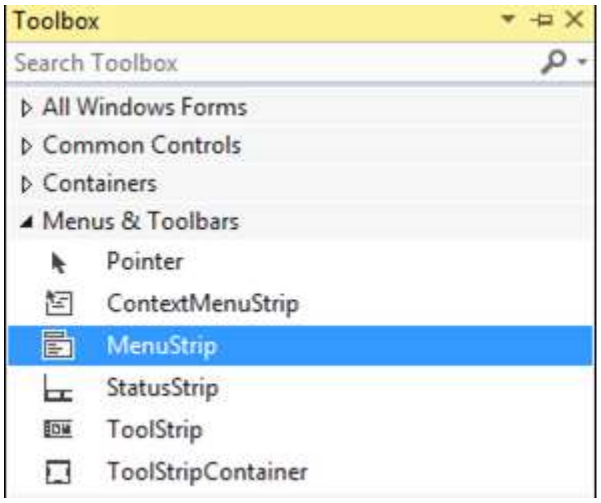


Figure 3.16: Use MenuStrip Object to add Menu in a Form

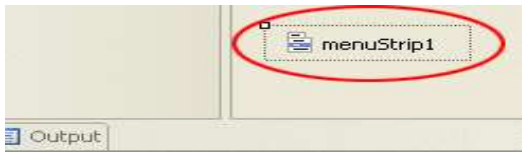


Figure 3.17: Menustrip Control Added

This is the control itself. Click on this (it's highlighted above), the Properties box on the right changes. There are many properties for the control. But there are lots of properties for the MenuItem object. The MenuItem object is the one at the top of the form - The one that says Type Here.

To start building menu list, click inside the area that says "Type Here". Type the word File as first menu. Then, press the enter key on your keyboard to add submenu list on File menu (refer Figure 3.18). To create items on your

File menu, click inside the **Type Here** box. Enter the word **New**, and press the enter key on your keyboard again. Add an "Open", a "Save" and a “Quit” item to submenu list in the same way. The submenu list will then look like at Figure 3.19.

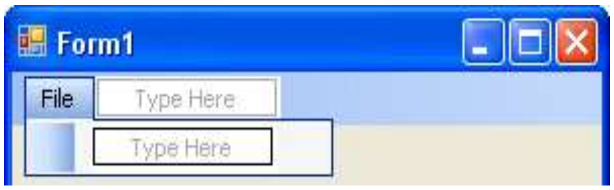


Figure 3.18: Build Menu List

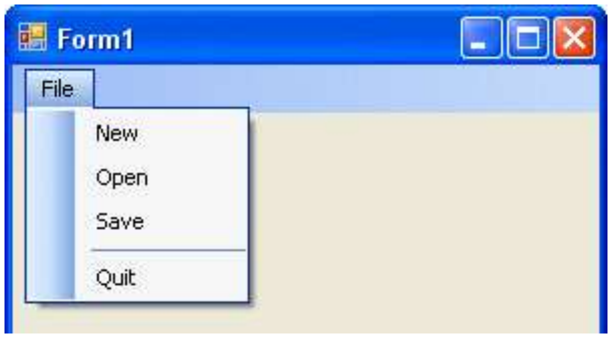


Figure 3.19: Menu Items from the First Menu

To see what the menu look like, Run the programme. Click the File menu. Since haven't added any code to the menu yet, so nothing will happen if an item on the menu is clicked. To make each menu items work, add program code. Return to the design environment. Click **File** in Design Time to see the drop down menu. Since "ExitToolStripMenuItem" is very difficult to remember. Rename the menu items so that they are more descriptive as following steps:

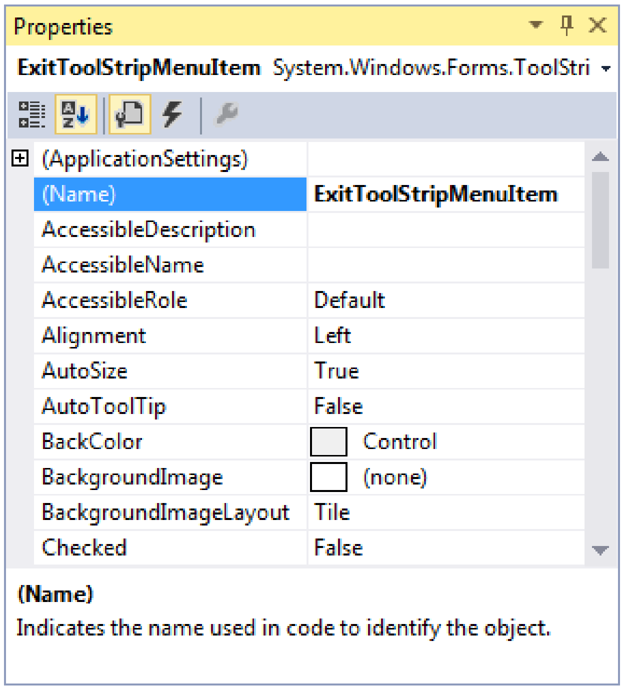
i. Get back to the form by pressing Shift + F7 on your keyboard
ii. Click the **File** menu to select it
iii. Select the **Exit** (or your Quit) item (Careful not to click in the middle as this may open the code window. Click near the left edge somewhere.)

iv. When the Exit item selected, look at the properties box (refer Figure 3.20

v. Click inside the **Name** property

vi. Change it to **mnuExit** (or mnuQuit)

vii. Press your return key on your keyboard to confirm the change


Properties

ExitToolStripMenuItem System.Windows.Forms.ToolStri

(ApplicationSettings)	
(Name)	ExitToolStripMenuItem
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
Alignment	Left
AutoSize	True
AutoToolTip	False
BackColor	Control
BackgroundImage	(none)
BackgroundImageLayout	Tile
Checked	False

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

Figure 3.20: Rename a Menu Item at Properties Window

Now double click at menu item or press F7 (or CTRL + ALT + 0) to bring the code window up. Click the drop down arrow of the General box, and the new name appear. Click on the new mnuExit item as Figure 3.21. Then, code stub would open, ready for you to type your code.

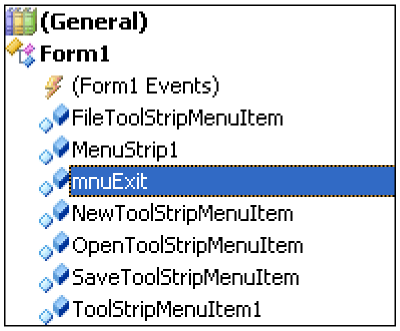


Figure 3.21: New Name of Menu Item

To jump straight to the code, to look at the drop down box opposite. It will probably say "Declarations". Click the arrow and see a new list as Figure 3.22.

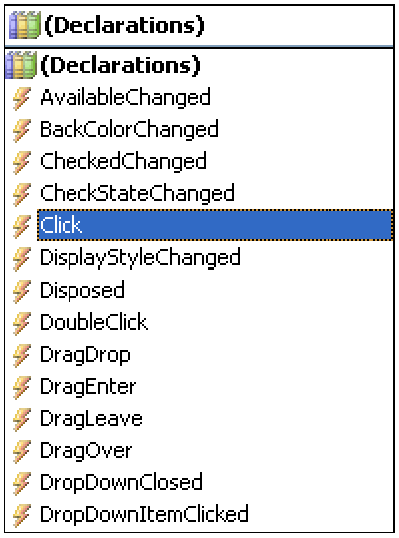


Figure 3.22: Event for the Menu Item

The items in the Declarations box are called Events. The Event you want is the **Click** event. So, select that one from the list. When select Click from the list, it taken straight into the code for that event. Add some of our own code, so that out Exit menu item actually does something. The word "Me" refers to the form. When type the word `Me`, a list items appear. Double click the word `Close`, then press return key.

```
Private Sub mnuExit_Click(ByVal sender As Object, _
                            ByVal e As System.EventArgs) _
                            Handles mnuExit.Click

    Me.Close()

End Sub
```

Figure 3.23: A Code Segment for Exit Menu Item

To test out the new code, run your program. Click your **File** menu, and then click the **Exit** item. The form should close down, and returned to the design environment.

3.5.2 Submenus

A sub menu is one that branches of a menu item. They usually have an arrow to indicate that there's an extra menu available. To create submenu, follow steps below:

i. Return to the Form view (Shift + F7 is a shortcut)
ii. Click on the File menu
iii. Select the New item (Click once on the left edge).
iv. Click on the "Type Here" just to the right of New. Type **New Project**, and then hit the return key on your keyboard.
v. Type in **New File** and then click away from the menu, somewhere on the form. See Figure 3.24.

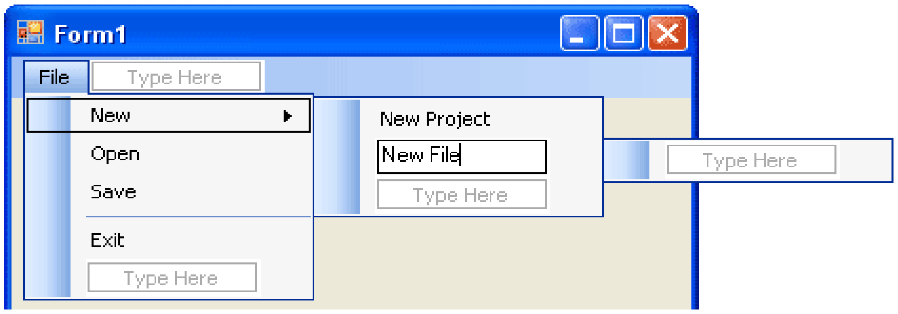


Figure 3.24: Add Submenus

Of course, none of the menu items work except the Exit menu. Write related program code to make other menu item works as plan. Now, it found that adding menus to an application is an easy matter with VB.NET.

3.6 Dialog Boxes

A dialog box (or dialogue box) is a type of window used to enable communication or "dialog" between an application and its user. It may communicate information to the user, prompt the user for a response, or both. Sometimes dialog box is called a Modal form. Built-in dialog boxes have available to create by drag the controls from toolbox and drop onto Windows forms for various tasks like: opening files, saving files, printing a page, folder browser and providing choices for colors, fonts, page set up etc. Other type of dialog boxes which created by program code are message box and input box. These built-in dialog boxes reduce the developer's time and work load. The dialog boxes are used to display information and prompt for input or response from the user.

3.6.1 Method to Show Dialog Boxes

To show dialog boxes, use the ShowDialog method. This method is used to display all the dialog box controls at run time. To convey this information from a dialog box back to the calling application, the object provides the DialogResult property. It returns a value of the type of DialogResult enumeration (Refer Table 3.1).

Table 3.1: Values of DialogResult Enumeration

Enumeration	Description
Abort	returns DialogResult.Abort value, when user clicks an Abort button
Cancel	returns DialogResult.Cancel, when user clicks a Cancel button
Ignore	returns DialogResult.Ignore, when user clicks an Ignore button
No	returns DialogResult.No, when user clicks a No button
None	returns nothing and the dialog box continues running
OK	returns DialogResult.OK, when user clicks an OK button
Retry	returns DialogResult.Retry , when user clicks an Retry button
Yes	returns DialogResult.Yes, when user clicks an Yes button

3.6.2 Add Dialog Boxes Control

To add a dialog box in an application, drag the control from toolbox and drop onto a form as Figure 3.25.

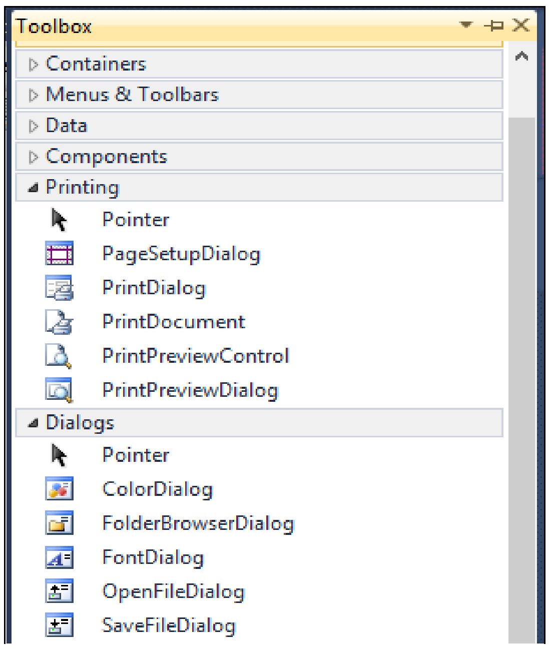


Figure 3.25: Dialog Boxes Object in a Toolbox

In this section, an example is provided to show how to display dialog boxes from menu item which created in 3.6. An Open File dialog box will be added at **File > Open** menu in Figure 3.19. In most applications, when the **File** menu is clicked, and select the **Open** item, a dialogue box is displayed. From the dialogue box, a file can be selected, then click the **Open** button. The file you is then opened up. We'll see how to do that from the menu list. (Except, the file won't open yet - only the dialogue box will display, and then name of the chosen file. Do the following steps:

1. Open up toolbox, and locate the control called "**OpenFileDialog**".
2. Double click the control to add one to your project. But notice that the control doesn't get added to a form. It gets added to the area at the bottom, next to menu control (see Figure 3.26)

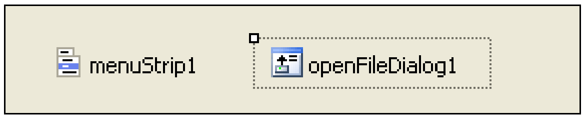
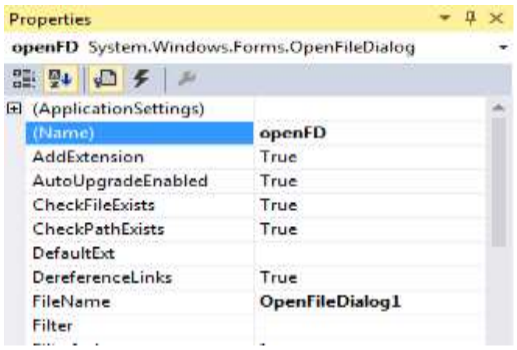


Figure 3.26: OpenFileDialog Control was added onto a Form

3. The shaded area surrounding the control means that it is selected. Look at the properties that can use with the control.
4. Click on the **Name** property and change the name to openFD. When you change the name in the properties box, the name of the control at the bottom will change (see Figure 3.27).



openFD System.Windows.Forms.OpenFileDialog

(ApplicationSettings)	
(Name)	openFD
AddExtension	True
AutoUpgradeEnabled	True
CheckFileExists	True
CheckPathExists	True
DefaultExt	
DereferenceLinks	True
FileName	OpenFileDialog1
Filter	

Figure 3.27: Property Setting for OpenFileDialog Control

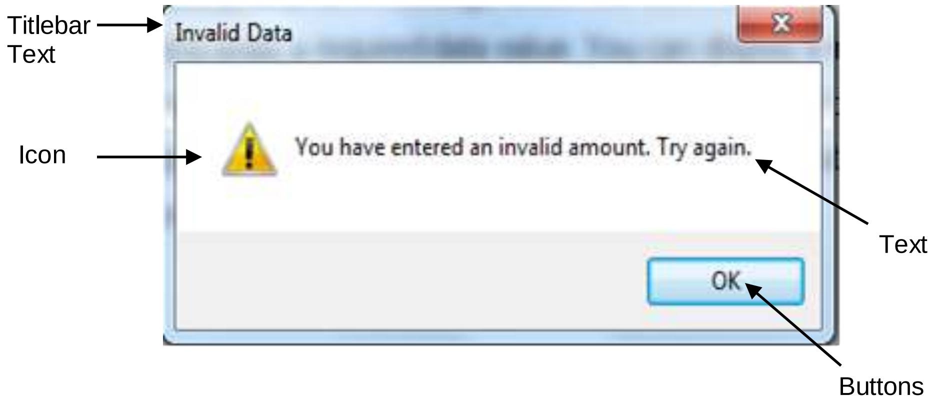
5. Now write some code to manipulate the properties of our new control. So, do the following:

   i. Access the code for your File > Open menu item. (To do this quickly, simply double click the Open item on your menu bar. Or press F7 (CTRL + ALT + 0 in version 2012) to access the Code View.)
   ii. Click the name of menu item from the left drop down box at the top of the code
   iii. Then select the Click event from the drop down box to the right
   iv. The empty code will be created
   v. In between two line of code, add a line of code for show a dialog box as below:

```
Private Sub mnuOpen_Click(ByVal sender As Object, _
ByVal e As System.EventArgs)Handles mnuOpen.Click

     openFD.ShowDialog()

End Sub
```

3.6.3 Message Boxes

VB.NET provides several built-in dialog boxes. They provide the common user interface style seen in most Windows applications. These dialogs provide properties and methods by which they can be customised to suit the needs of the particular application while still maintaining the standard Windows look and feel. The most used dialog box is MessageBox. This allows the application to display a message to the user and accept input regarding a choice that the user has made. The constituent parts of a MessageBox dialog are shown in Figure 3.28 to specify the message, an optional icon, the title bar text, and button(s) for the message box (refer).



You have entered an invalid amount. Try again.

Figure 3.28: Example of Message Box Created

`Show` method of the `MessageBox` object is used to display a message box. The `MessageBox` object is a predefined instance of the `MessageBox` class that can be used any time you need to display a message. There is more than one way to call the `Show` method as following:

```
MessageBox.Show(TextMessage)
MessageBox.Show(TextMessage, TitlebarText)
MessageBox.Show(TextMessage, TitlebarText, MessageBoxButtons)
MessageBox.Show(TextMessage, TitlebarText, MessageBoxButtons, MessageBoxIcon)
```

The `TextMessage` is the message you want to appear in the message box. The `TitlebarText` appears on the title bar of the `MessageBox` window. The `MessageBoxButtons` argument specifies the buttons to display and the `MessageBoxIcon` determines the icon to display.

Each option for your message box is separated by a comma. When type a comma in the code above, another pop-up menu will be appear. On this menu, choose the necessary buttons for the message box as Figure 3.29 then double click this item, then type a comma.

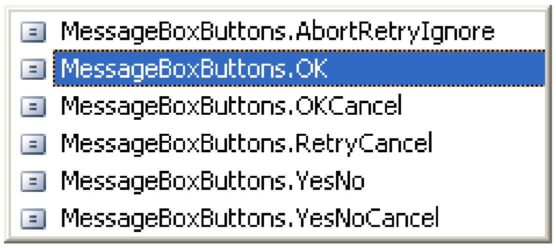

MessageBoxButtons.AbortRetryIgnore
MessageBoxButtons.OK
MessageBoxButtons.OKCancel
MessageBoxButtons.RetryCancel
MessageBoxButtons.YesNo
MessageBoxButtons.YesNoCancel

Figure 3.29: Message Box Buttons

Yet another pop-up menu will appear. On this menu, specify the symbol that appears in the message box in list at Figure 3.30.

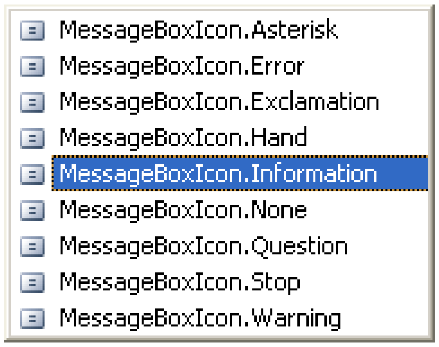


Figure 3.30: Message Box Icon

Table 3.2: List of member Name and its Icon for Design a Message Box

Member Name	Icon or Symbol
Asterisk	Information
Information	
Error	Error
Hand	
Stop	
Warning	Exclamation
Exclamation	
Question	Question mark
None	Not display any icon

Example:

A message to display when the user has entered invalid data or neglected to enter a required data value. The above message box in Figure 3.28 was created with the following method call:

```
MessageBox.Show("Enter numeric data.")
MessageBox.Show("Try again.", "Data Entry Error")
MessageBox.Show("This is a message.", "This is a title bar", MessageBoxButtons.OK)
```

3.6.4 Input Boxes

If an application need a password, name, or other info, input boxes is a very easy way of getting it. Input Box is design at run time only. The syntax for an InputBox is:

```
InputBox(prompt[,title][,default][,xpos][,ypos][,helpfile,context])
```

- The *prompt* is the message the input box. This tells the user what the program wants them to input. (ex: "Please enter your password.").
- The *title* is the title of the input box, which is the part that appears in the blue. (ex: "Password").
- *Default* is an option may want to use. It puts text in the text box part of the InputBox as a default. (ex: "Anonymous" is a good default if you're asking for the user's name).
- *Xpos* and *ypos* are the horizontal and vertical positions of the input box. If you leave these out, the default puts the input box horizontally centered and about one third of the way down the screen.
- *Helpfile* and *context* tell where in a help file the user can get help on that Input Box.

Here is an example of program code in order to use Input Box in an application:

```
Dim strUserPassword As String
strUserPassword = InputBox("Please enter your password", "Password")
If strUserPassword = "DAT20903" Then
....
```

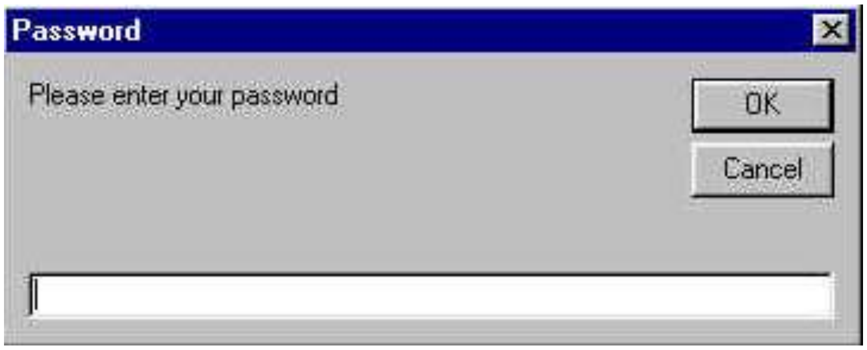


Figure 3.31: Input Box

Activities

1. You can display program output in a text box or a label. When should you use a text box? When is a label appropriate?
2. If you want two groups of radio buttons on a form, how can you make the groups operate independently?
3. What is the purpose of keyboard access keys? How can you define them in your project?
4. What is a ToolTip? How can you make a ToolTip appear?
5. Assume you are testing your project and don't like the initial position of the insertion point. Explain how to make the insertion point appear in a differet text box when the program begins.
6. What Basic statements will clear the current contents of a text box and a label?
7. How are the With and End With statements used? Give an example.
8. What is concatenation and when would it be useful?

References

1. N. Randolph and D. Gardner, 2009. ***Professional Visual Studio 2008,*** Wiley Publishing.

2. J.C. Bradley and A.C. Millspaugh, 2005. ***Programming in Visual Basic .NET***, McGraw Hill Technology Education.

3. D.I. Schneider, 2006. ***An Introduction to Programming Using Visual Basic 2005***. Prentice Hall.

4. VB .Net Programming Tutorial retrieved from https://www.tutorialspoint.com/vb.net/

TOPIC 4

PROGRAMMING FOR APPLICATION DEVELOPMENT

Learning Outcome

At the end of this topic, student should be able to:

1. Distinguish between variables, constants, and controls for different scope.
2. Apply common programming technique using correct data, operators, scope and control structures in application development
3. Create and use procedures in application
4. Display Dialog boxes in application.
5. Apply array in collection of data.

Content

4.1 Introduction

When created a professional user interface, developer have to write code to make the program do something useful. A Code Editor will be used and to be familiar with correct programming technique have to apply. Data, variables, operators, scope, procedures and control structures are all important concepts that need to understand to start writing useful and power full programs.

4.2 Programming Style

Every programming language have its own programming structure and coding conventions.

4.2.1 Statements, Keywords and Comments

Statements are VB.Net instructions that contain keywords operator, variables, constants and expressions. While, keywords are language-specific words that have special meaning in VB.Net. By default, keywords are displayed in the color blue. Any text following a single quotation mark is handled as a comment (unless the single quotation mark is placed inside double quotation marks). By default, words that make up comments are displayed in green in the Code Editor. Comments are used to describe code to make it clear to the reader what the code is supposed to do. The compiler will ignore any comments in the code at run time.

4.2.2 Continuing Long Program Lines

Basic interprets the code on one line as one statement. You can type very long lines in the Editor window; the window scrolls sideways to allow you to keep typing. However, this method is inconvenient; it isn't easy to see the ends of the long lines. When a Basic statement becomes too long for one line, use a **line-continuation character**. You can type a space and an underscore, press Enter, and continue the statement on the next line. It is OK to indent the continued lines. The only restriction is that the line-continuation character must appear between elements; you cannot place a continuation in the middle of a literal or split the name of an object or property.

Examples:

```
lblGreeting.Text = "Greetings " &txtName.Text & ": " & _
     "You have been selected to win a free prize. " & _
```

```
"Just send us RM50 for postage and handling."
```

4.2.3 Scope: Public or Private

Scope determines whether an object, method, or variable can be accessed by other objects, methods, or variables. The scope of a variable, sometimes referred to as accessibility of a variable, refers to where the variable can be read from and/or written to, and the variable's lifetime, or how long it stays in memory. The scope of a procedure or method refers to where a procedure can be called from or under what context you are allowed to call a method. There are five level of scope that can be used: Public, Private, Protected, Friend and Protected Friend. However, public and private levels of scope (access) have commonly discussed. Public scope is accessible to any object or process that can reference the current class or process. While, the private scope is accessible only to the current class.

4.3 Variables and Constants

Basically, all data used in application development have been worked with the Text property of Text Boxes and Labels. Now, without properties, variables can be used to store data. A variable is a memory locations that hold the data **can be changed** during project execution. While, when locations that hold data that **cannot change** during execution are called **constants**. For example, the customer's name will vary as the information for each individual is processed. However, the name of the company and the sales tax rate will remain the same (at least for that day). When a variable or a named constant declared, VB.Net **reserves** an **area of memory** and assigns it a name, called an **identifier**. The declaration statements establish project's variables and constants, give its names, and specify the type of data that is going to be hold.

4.3.1 Data Types

The data type of a variable or constant indicates what type of information will be stored in the allocated memory space: perhaps a name, a dollar amount, a date, or a total. Table 4.1 shows the VB.Net data types. The most common types of variables and constants you will use are String, Integer, and Decimal. When deciding which data type to use, follow this guideline: If the data will be used in a **calculation**, then it must be **numeric** (usually **Integer** or **Decimal**); if it is **not** used in a **calculation**, it will be **String**. Use **Decimal** as the data type for any decimal **fractions** in business applications; Single and Double data types are generally used in scientific applications.

Table 4.1: Visual Basic Data Types

Data Type	Use for	Storage Size
Boolean	True or False values	2
Byte	0 to 255, binary data	1
Char	Single Unicode character	2
Date	1/1/0001 through 12/31/999	8
Decimal	Decimal fractions, such as dollars and cents	16
Single	Single-precision floating point numbers with six digits of accuracy	4
Double	Double-precision floating point numbers with 14 digits of accuracy	8
Short	Small integer in the range -32,768 to 32,767	2
Integer	Whole numbers in the range -2,147,483,648 to +2,147,483,647	4
Long	Larger whole numbers	8
String	Alphanumeric data: letters, digits, and other characters	varies
Object	Any type of data	4

4.3.2 Declaration of a Variable

Although there are several ways to declare a variable, the most commonly used statement is the `Dim` statement. If you omit the optional data type, the variable's type defaults to object. It is best to always declare the type, even when you intend to use objects.

Syntax:

```
Dim Variablename as DataType
```

Examples:

```
Dim strStudentName        As String
Dim intMatricNo           As Integer
```

Another style in write the declaration statement is multiple declarations in line statement.

Examples:

```
Dim a,b As Double 'variables with same data type
Dim a As Double, b As Integer 'different variables with
different data type
Dim c As Double=2, b As Integer=5 'declaration and
initialization statement
```

4.3.3 Scope of Variable

Variables can be declared at four different locations in your programs. Notice that good programming practices dictate that constants should be declared at the module/ class level. This technique places all constant declarations at the top of the code and makes them easy to find in case need to make changes. Namespace-level variables and constants can sometimes be useful when a project has multiple forms and/or modules, but good programming practices exclude the use of namespace-level variables. The following table provides the general rules for scope. Figure 4.1 illustrates the locations for coding local variables and module-level variables.

Table 4. 2: General Scoping Rules

Location	Description	Keyword
Block	A variable declared within a block construct such as an **If** statement, that variable's scope is only until the end of the block. The lifetime is until the procedure ends.	`Dim`
Procedure (local)	A variable declared within a procedure, but outside of any **If** statement, the scope is until the **End Sub** or **End Function**. The lifetime of the variable is until the procedures ends.	`Dim` (Refer Figure 4.2)
Module/ Class	A variable declared outside of any procedure, but it must be within a **Class…End Class** or **Module…End Module** statement. The scope is any procedure within this module. The lifetime for a variable defined within a class is until the object is cleaned up by the garbage collector. The lifetime for a variable defined within a module is until the program ends.	`Dim`, `Public`, or `Private`
Project	A **Public** variable can declared within a **Module…End Module** statement, and that variable's scope will be any procedure or method within the project. The lifetime of the variable will be until the program ends.	

```
(Declaration section)

Dim ModuleLevelVariables
Const NamedConstants

Private Sub calculateButton_Click
Dim LocalVariables
…
End Sub

Private Sub summaryButton_Click
Dim LocalVariables
…
End Sub
```

Figure 4.1: Declaration of Module Level or Local Variables

```
'Module-level declarations
Const DISCOUNT_RATE As Decimal = 0.15D

Private Sub calculateButton_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs)Handles calculateButton.Click)
'Calculate the price and discount
Dim quantity As Integer
Dim price, extendedPrice, discount, discountedPrice As Decimal

'Convert input values to numeric variables
quatity = Integer.Parse(quantityTextBox.Text)
price = Decimal.Parse(priceTextBox.Text)

'Calculate values
extendedPrice = quantity + price
discount = Decimal.Round((extendedPrice * DISCOUNT_RATE), 2)
discountedPrice = extendedPrice - discount

End Sub
```

Figure 4.2: Example of Local and Module-Level Declarations

4.3.4 Constants: Named and Intrinsic

Constants provide a way to use words to describe a value that doesn't changed either **named constants** or intrinsic constants.

4.3.4.1 Named Constants

Named constants is **declared** using the keyword **Const**. Give the constant a name, a data type, and a value. Once a value is declared as a constant, its value cannot be changed during the execution of the project. The data type that declared and the data type of the value must match. For example, if an integer constant is declared, give it an integer value. The code is easier to read; for example, seeing the identifier `decMAX_PAY` is more meaningful.

Examples:

```
Const strCOMPANY    As String = "Syahmie Sdn Bhd"
Const decSALES_RATE As Decimal = .08D
```

Follow certain rules when assign values as constants:

i. Cannot include a comma, dollar sign, any other special character, or a sign at the right side.

ii. A text (string) value must be enclosed in quotation marks; String literals (also called string constants) may contain letters, digits, and special characters, such as $#@%&*.

iii. A numeric constants may contain only the digits (0-9), a decimal point, and a sign (+ or -) at the left side.

iv. Declare the data type of numeric constants by appending a type-declaration character. If you do not append a type-declaration character to a numeric constant, any whole number is assumed to be Integer and any fractional value is assumed to be Double. The type-declaration characters are Decimal as D, Double as R, Integer as I, Long as L, Short as S and Single as F.

4.3.4.2 Intrinsic Constants

Intrinsic constants are **system-defined** constants. Many sets of intrinsic constants are declared in **system class libraries** and are available for use in VB programs. For example: to use intrinsic constants, specify the class name or group name as well as the constant name. For example, `Color.Red` is the constant "Red" in the class "Color".

4.4 Operators

In programming, calculations can performed with variables, with constants, and with the properties of certain objects. The properties you will use, such as the Text property of a text box or a label, are usually strings of text characters. These characters strings, such as “Rafizah” or “12345”, should not be used directly in calculations unless you first convert them to the correct data type. Use functions to convert the property of a control to its numeric form before use the value in a calculation. The function that you use depends on the data type of the variable to which you are assigning the value. For example, to **convert text to** an **integer**, use the `CInt` function; to convert to a decimal value, use `CDec`.

Examples:

```
‘Convert input values to numeric variables
intMarks = CInt(txtMarks.Text)
decPrice = CDec(txtPrice.Text)
```

Converting from one **data type to another** is sometimes called **casting**. In the preceding example, **`txtMarks.Text`** is **cast into** an **`Integer`** data type and **`txtPrice.Text`** is **cast into** a **`Decimal`** data type. Table 4.3 shows a list of conversion examples:

Table 4.3: Conversion Examples

Contents of String Argument	`CInt` (Argument)	`CDec` (Argument)
123.45	123	123.45
$100	100	100.0
1,000.00	1000	1000.0
A123	(error)	(error)
-5	-5	-5.0
5-	-5	-5.0
(5)	-5	-5.0
1(5)	(error)	(error)
0.01	0	0.01
1.5	2	1.5
(blank)	(error)	(error)

When a conversion function encounters a value that it cannot parse to a number, such as a blank or a nonnumeric character, an error occurs.

4.4.1 Arithmetic Operators

The arithmetic operations can perform in VB.Net include addition (+), subtraction (-), multiplication (*), division (/), integer division (\), modulus (Mod), and exponentiation (^). The first four operations are self-explanatory, but may not be familiar with \, Mod or ^.

4.4.1.1 Integer Division

Use integer division to divide one integer by another giving an integer result, truncating (dropping) any remainder. For example, if `intTotalMinutes = 150`, then, `intHours = intTotalMinutes\60` returns `2` for `intHours`.

4.4.1.2 Mod

The Mod operator returns the remainder of a division operation. For example, if `intTotalMinutes = 150`, then, `intMinutes = intTotalMinutes Mod 60` returns `30` for `intMinutes`.

4.4.1.3 Exponentiation

The exponentiation operator raises a number to the power specified. The following are examples of exponentiation.

```
decSquared = decNumber ^ 2  'Square the number--Raise to the 2nd power
decCubed = decNumber ^ 3    'Cube the number--Raise to the 3rd power
```

4.4.2 Comparison Operator

A comparison operator compares operands and returns a logical value based on whether the comparison is true or not. These values yield true or false values. Table 4.4 shows the list of comparison operators and its uses.

Table 4.4: List of Comparison Operators and its Uses

Operator	Use	Value
<	Less than	True if *operand1* is less than *operand2*
<=	Less than or equal to	True if *operand1* is less than or equal to *operand2*
>	Greater than	True if *operand1* is greater than *operand2*
>=	Greater than or equal to	True if *operand1* is greater than or equal to *operand2*
=	Equal to	True if *operand1* equals *operand2*
<>	Not equal to	True if *operand1* is not equal to *operand2*
Is		True if two object references refer to the same object
Like		Performs string pattern matching

4.4.3 Assignment Operator

Calculations will perform using assignment statements. Recall that whatever appears on the right side of an "=" (assignment operator) is assigned to the item on the left. The left side may be the property of a control or a variable. It cannot be a constant. Table 4.5 shows the list of assignment operators and its uses.

Table 4.5: List of Assignment Operators

Operator	Description
=	Assignment
^=	Exponentiation followed by assignment

*=	Multiplication followed by assignment
/=	Division followed by assignment
\=	Integer division followed by assignment
+=	Addition followed by assignment
-=	Subtraction followed by assignment
&=	Concatenation followed by assignment

Examples:

```
decAverage = decSum/intCount
lblAmountDue.Text = CStr(decPrice – (decPrice * decDiscountRate))
txtCommission.Text = CStr(decSalesTotal * decCommissionRate)
```

In the preceding examples, the results of the calculations were assigned to a variable, the Text property of a label, and the Text property of a text box. In most cases you will assign calculation results to variables or to the Text properties of labels. Text boxes are usually used for input from the user rather than for program output. In addition to the equal sign (=) as an assignment operator, VB .NET has several operators that can perform a calculation and assign the result as one operation. The new assignment operators are +=, -=, *=, /=, \=, and &=. Each of these assignment operators is a **shortcut** for the standard method; you can use the standard (longer) form or the newer shortcut. The shortcuts allow you to type a variable name only once instead of requiring you to type it on both sides of the equal sign. For example, to add `decSales` to `decTotalSales`, the long version is

```
decTotalSales = decTotalSales + decSales   'Accumulate a total
```

Instead you can use the shortcut assignment operator:

```
decTotalSales += decSales     'Accumulate a total
```

4.4.4 String Concatenation Operator

Concatenation operators ('&' and '+') are used to join multiple strings into a single string. For example, "Hi "& " there " yields the string "Hi there"

4.4.5 Order of Operations

The order in which operations are performed determines the result. Consider the expression `3 + 4 * 2`. What is the result? If the addition is done first, the result is `14`. However, if the multiplication is done first, the result is `11`. The hierarchy of operations or order of precedence, in arithmetic expressions from highest to lowest are stated below:

1. Any operation inside parentheses
2. Exponentiation
3. Multiplication and division
4. Integer division
5. Modulus
6. Addition and subtraction

In the previous example, the multiplication is performed before the addition, yielding a result of 11. To change the order of evaluation, use parentheses. The expression `(3 + 4) * 2` will yield `14` as the result. One set of parentheses may be used inside another set. In that case, the parentheses are said to be nested. The following is an example of nested parentheses:

```
((intScore1 + intScore2 + intScore3) / 3) * 1.2
```

Extra parentheses can always be used for clarity.
The expressions:

```
2 * decCost * decRate and (2 * decCost) * decRate
```

are equivalent, but the second is easier to understand. Multiple operations at the same level (such as multiplication and division) are performed from left to right. The example `8 / 4 * 2` yields `4` as its result, not `1`. Although the precedence of operations in Basic is the same as in algebra, take note of one important difference: There are no implied operations in Basic. Table 4.6 show expressions would be valid in mathematics, but they are not valid in VB.

Table 4.6: Mathematical Versus VB Expression

Mathematical Notation	Equivalent VB Expression
2A	2 * A
3(X + Y)	3 * (X + Y)
(X + Y)(X – Y)	(X + Y) * (X – Y)

4.5 Procedures

So far, the code mostly been lumped together under one button. The problem with this approach is the code can get quite long and complex, making it difficult to read, and difficult to put right if something goes wrong. Another approach is to separate some of this code into its own routine. This is where functions and subs come in.

A procedure is **a group of statements that together perform a task when called**. After the procedure is executed, the control returns to the statement calling the procedure. VB.Net has 4 types of procedures:

i. **Sub- procedure or Subs:** perform specific task

ii. **Function procedure** (User Created Function and Built-in Function): return a value to the code that called them

iii. **Event-handling procedure or "event handler":** invoked in response to event having occurred to an object. Event can be triggered by the user or by the program. For example: `btnExit_Click` denotes the Click event handler for the button control named `btnExit`

iv. **Property procedure**: used when assigning values to the properties of user-created objects, and when retrieving the values of those properties.

4.5.1 Difference between Functions and Subs

The difference between a Function and a Sub: Functions return a value, and Subs do not. A function is used to get some sort of answer back, and use the answer elsewhere. Assign the answer to the function to a variable. For example, `Substring()` is a Function. Assign the answer to the `Substring()`. A Sub is some code or job that you want VB to get on with and execute. An example is closing a form down and unloading it with `Me.Close()`. A programmer don't need to return a value, here. It just want VB to close the form down.

4.5.2 Create Own Sub Procedures

Own sub procedures is created by defining the task under keyword Subs followed by Subs Name and close with End Sub. Subs is use simply use by referring to the Sub by name or using the "Call" word to makes the code easier to read, and tells own Subroutine on this line is executing. For example, the below code is definition of own Subs (Figure 4.3) and calling the subs when needed (Figure 4.4). The sub procedure's name used is `ErrorCheck()`. Then statement `Call ErrorCheck()` written under event-handling procedure of a new button where the Message box popping up, when nothing is entered into the Textbox. The point about this is that own code segment have created. This segment of code can use whenever needed, just by referring to it by name or used keyword call.

```
Private Sub ErrorCheck()
   Dim TextData As String

   TextData= Trim (txtInput.Text)

   If TextData= " " Then
         MsgBox("Please Enter your First Name")
   End If
End Sub
```

Figure 4.3 Definition of a Sub Procedure Named `ErrorCheck()`

```
Private Sub btnCheck_Click (ByVal sender As System….)
     Call ErrorCheck()
End Sub
```

Figure 4.4 Calling Sub Procedure `ErrorCheck()` from Event Procedure

4.5.3 Using Parameters in Sub Procedure

A parameter is a value that we want to pass from one code section to another. Parameters are set up in between the parentheses. Figure 4.5 below show an example created Sub to add together the two numbers from the Textboxes. The sub procedure used two parameters (`first` and `second`). Using this code, the values gathered from button code will hand them over to `AddNumbers` Sub. Variables in the `AddNumbers` Sub does not have to be the same names as the calling line. But the values in the variables get passed in the order set them up. The value in the variable **`first`** will get passed to the first variable in `AddNumbers` Sub; the value in the variable **`second`** will get passed to the next variable which set up in our `AddNumbers` Sub (refer Figure 4.6).

```
Private Sub AddNumbers(first As Integer, second As
   Integer)
   Dim answer As Integer
   answer = first + second

   MsgBox("The total is " & answer)
End Sub
```

Figure 4.5 Definition of a Sub Procedure Named `AddNUmbers()` with Parameters

```
Private Sub btnAnswer_Click (ByVal sender As System…)
   Dim first As Integer
   Dim second As Integer
   first = Val(txtFirstNumber.Text)
   second = Val(txtSecondNumber.Text

   Call AddNumbers(firstno, secondno)
End Sub
```

Figure 4.6 Calling Sub Procedure `btnAnswer()` with Parameters from Event Procedure

4.5.4 Using ByVal and ByRef in Sub Procedure

The word **ByVal** is short for "By Value". It is means that a copy of a variable are passing to Subroutine. Changes can be made to the copy and the original will not be altered. ByVal is hidden in Version 2012/13 of VB NET Express. It's hidden because ByVal is the default when the variables are passing over to a function or Sub.

Let's see a coding example at Figure 4.7 and Figure 4.8. After these code executed, Number1 didn't get incremented because that only a copy of the original variable got passed over. When the variable incremented, only the copy got 1 added to it. The original stayed the same which equal 10.

```
Private Sub IncrementValue(ByVal Number1 As Integer)
   Number1 = Number1 + 1
End Sub
```

Figure 4.7 Definition of a Sub Procedure Named `IncrementValue()` with Parameters

```
Private Sub btnIncrease_Click (ByVal sender As System…)
   Dim Number1 As Integer
   Number1=10

   Call IncrementValue(Number1)
   MsgBox(Number1)
End Sub
```

Figure 4.8 Calling a Sub Procedure `IncrementValue()` with Parameters from Event Procedure

`ByRef` is the alternative. This is short for By Reference. This means that a copy of the original variable are not handing over but pointing to the original variable. If the ByVal keyword code in Figure 4.6 and 4.7 changes to `ByRef`, Message Box will displayed 11. The code referencing the original variable. So when it add 1 to it, the original will change. The variable has now been incremented! As conclusion, the default is **ByVal** - which means a copy of the original variable. If you need to refer to the original variable, use **ByRef**.

4.5.5 Function Procedures

The Function statement is used to declare the name, parameter and the body of a function.

Syntax for the Function statement is:

```
[Modifiers] Function FunctionName [(ParameterList)] As ReturnType
    [Statements]
End Function
```

Where,

- ***Modifiers***: specify the access level of the function; possible values are: Public, Private, Protected, Friend, Protected Friend and information regarding overloading, overriding, sharing, and shadowing.
- ***FunctionName***: indicates the name of the function
- ***ParameterList***: specifies the list of the parameters
- ***ReturnType***: specifies the data type of the variable the function returns

In VB.Net, a function can return a value to the calling code in two ways:

- By using the return statement
- By assigning the value to the function name

4.5.5.1 Create Own Function in VB .NET

A **function** is more or less the same thing as a Sub - a segment of code is created, and that can be used whenever you want it. The difference is that a Function returns a value, while a Sub doesn't.

A Function is different. It is a value, will be equal to something, and have to assign a value to it. Create a Function in the same way you did a Sub, but have different syntax. First, changed the word "**Sub**" to "**Function**"; second we've added "**As**" something, in this example "**As Boolean**". The name we called our Function is **`ErrorCheck`**, and ErrorCheck is now just like a variable. And just like a variable, we use one of the **Types**. Any of the variable types can be used such as "**`As Integer`**", "**`As Long`**", "**`As Double`**", "**`As String`**" or etc.

```
Private Function ErrorCheck () As Boolean
        Dim TextBoxData As String
        TextBoxData = Trim(txtFunction.Text)
        If TextBoxData = "" Then
            MsgBox("Blank Text Box detected")
            ErrorCheck=True
        Else
            ErrorCheck=False
        End If
End Function
```

Figure 4.9 Definition of a Function Procedure Named `ErrorCheck()`

```
Private   Sub   btnCheck_Click   (ByVal   sender   As   System…)
    Dim IsError As Boolean

         Dim first As Integer
         Dim second As Integer
         Dim result As Integer
         first = Val(txtNumber1.Text)
         second = Val(txtNumber2.Text)

         result = AddTwoNumbers(first, second)


         If result = 0 Then
                 MsgBox("Please try again ")
         Else
                 MsgBox("The answer is " & result)
         End If
 End Sub
```

Figure 4.10 Calling a Function Procedure `ErrorCheck()` **from Event Procedure**

Remember that **`ErrorCheck`** is now like a variable. In this case it was a **`Boolean`** variable. So if there's nothing in the Textbox, we have set **`ErrorCheck`** to True.This time, because we've set up a Function, assign the value of **`ErrorCheck`** to the variable called "**`IsError".`**Once that code is executed we can then use the variable `IsError` and test its value. If it's true, then we know that the user did not enter anything into the Textbox; if it's False, then we know that they did. The benefit of using a Function to check for our errors is that we can halt the programme if `IsError = True`.

To sum up, then. A function will return a value. Put this value into the name of the Function. Then assign the value of the Function to a variable. Lastly, test the variable to see what's in it.

4.5.5.2 Use Parameters with Functions

Use the Parameters in function is exactly the same way as did for a Sub. From an example showed in Figure, the name of this Function is `AddTwoNumbers`, and set it up to return an Integer value. The two parameters we're passing in are also Integers. The code inside the Function simply adds up whatever is inside the variables `first` and `second`. The result is passed to another variable, `answer`. Then pass whatever is inside `answer` to the name of our Function. So, `AddTwoNumbers` will be equal to whatever is in the variable `answer`.

Instead of saying `AddTwoNumbers = answer` the `Return` keyword can be used like this:

`Return answer`

The result is the same: the value inside the variable answer is now the value of the function.

```
Private  Function  AddTwoNumbers(ByVal  first  As  Integer,
ByVal second As Integer) As Integer
      Dim answer As Integer
      answer = first + second
      'return a value by assigning the value to the function name
      AddTwoNumbers = answer

End Function
```

Figure 4.11 Definition of a Function Procedure Named `AddNUmbers()` with Parameters

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

Dim first As Integer
    Dim second As Integer
    Dim result As Integer
         first = Val(txtNumber1.Text)
         second = Val(txtNumber2.Text)
         result = AddTwoNumbers(first, second)
         If result = 0 Then
             MsgBox("Please try again ")
         Else
             MsgBox("The answer is " & result)
         End If

 End Sub
```

Figure 4.12 Calling Function Procedure `btnAnswer()` with Parameters from Event Procedure

Run this programme code and test it out. Type a number in the first text box, and one in the other. Then click the "Get Function Answer" button. Try typing two zeros into the textboxes and see what happens. Setting up and using functions can be quite tricky at first, but it's well worth your while persevering: they can vastly improve your coding skills.

4.5.5.3 Built-in Function in VB .NET

There are many built-in function provided in VB.Net. Common built-in function used in application development are Math Functions, String Functions and TimeDate Function shown in Table 4.7.

Table 4.7 List of Common Built-In Function

Type of Function	Function Name	Descriptions	Calling Syntax
Math	`Math.Abs()`	to return the absolute value of a number	`Math.Abs(Number)`
	`Math.Max()`	to return the largest of two decimal numbers.	`Math.Max( val1, val2)`

	`Math.Round()`	number to be rounded to the number of Decimal Places.	`Math.Round(Number, Decimal Places)`
String	`Lcase()`	returns a string after converting to lowercase.	`LCase(Char)or LCase(Str)`
	`LTrim()`	retruns a copy of a string without leading spaces.	`LTrim(Str)`
	`StrComp()`	to compare two strings to return values '-1', '1', '0' based on the value.	`StrComp(Str1, Str2)`
	`Val()`	return the numbers contained in a string as a numeric value of appropriate type.	`Val(Expression)`
DateTime	`IsDate()`	checks if the given expression is a valid date and returns a boolean true or false.	`IsDate(Expession)`
	`Month()`	returns the month of the year as an integer value in the range of 1-12.	`Month(Date)`

4.5.6 Event Handling Procedures

Many programmers refer to sub procedures as subprograms or subroutines. But subroutine is not acceptable because VB actually has a different statement for a subroutine, which is not the same as a sub procedure. VB automatically names your event procedures. The name consists of the object name, an underscore (_), and the name of the event. For example, the Click event for your button called `btnMessage` will be `btnMessage`. For the sample project you are writing, you will have a `btnMessage_Click` procedure and a `btnExit_Click` procedure.

4.6 Control Structures

Basically there are two types on control structures: selection and repetition structures. Both structure involved with conditions statement.

4.6.1 Conditions

The test in an `If` statement is based on a condition. To form conditions, **six relational or comparison operators** (as Table 4.3) are used to **compare values**. The **result** of the comparison is either **True** or **False**. The conditions to be tested can be formed with numeric variables and constants, string variables and constants, object properties, and arithmetic expressions. However, it is important to note that comparisons must be made on like types; that is, strings can be compared only to other strings, and numeric values can be compared only to other numeric values, whether a variable, constant, property, or arithmetic expression.

4.6.1.1 Comparing Numeric Variables and Constants

When numeric values are involved in a test, an algebraic comparison is made; that is, the sign of the number is taken into account. Therefore, negative 20 is less than 10, and negative 2 is less than negative 1. Even though an equal sign (=) means replacement in an assignment statement, in a relation test and equal sign is used to test for equality. Example show at Table 4.8.

Table 4.8: An Example a Condition and Its Description

Condition	Description
`If Decimal.Parse(txtPrice.Text) = decMaximum Then`	"Is the current numeric value stored in `txtPrice.Text` equal to the value stored in `decMaximum`?"

4.6.1.2 Comparing Strings

String variables can be compared to other string variables, string properties, or string literals enclosed in quotation marks. The comparison begins with the left-most character and proceeds one character at a time from left to right. As soon as a character in one string is not equal to the corresponding character in the second string, the comparison is terminated, and the string with the lower-ranking character is judged less than the other. The code, called the ANSI (American National Standards Institute) code (refer to Table 4.9), has an established order (called the collating sequence) for all letters, numbers, and special characters.

Table 4.9: ANSI Collating Sequence

Code	Character	Code	Character	Code	Character
32	Space (blank)	64	@	96	‘
33	!	65	A	97	a
34	“	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	,	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{

60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_	127	Del

Example 1:

txtPerson1.Text	txtPerson2.Text
JOHN	JOAN

The condition `txtPerson1.Text < txtPerson2.Text` evaluates False. The `A` in `JOAN` is lower ranking than the `H` in `JOHN`.

Example 2:

txtWord1.Text	txtWord2.Text
HOPE	HOPELESS

The condition `txtWord1.Text < txtWord2.Text` evaluates True. When one string is shorter than the other, it compares as if the shorter string is padded with blanks to the right of the string, and the blank space is compared to a character in the longer string.

Example 3:

lblCar1.Text	lblCar2.Text
300ZX	Porche

The condition `lblCar1.Text < lblCar2.Text` evaluates True. When the number `3` is compared to the letter `P`, the `3` is lower, since all numbers are lower ranking than all letters.

4.6.1.3 Comparing Uppercase and Lowercase Characters

When comparing strings, the case of the characters is important. An uppercase `Y` is not equal to a lowercase `y`. Because the user may type a name or word in uppercase, lowercase, or as a combination of cases, we must check all possibilities. The best way is to use `ToUpper` and `ToLower` methods of the String.class, which return the uppercase or lowercase equivalent of a string, respectively. Refer Table 4.10.

Table 4.10: Examples Use of `ToUpper` and `ToLower` Methods

`txtOne.Text Value`	`txtOne.Text.ToUpper`	`txtOne.Text.ToLower`
Basic	BASIC	basic
DOT NET	DOT NET	dot net
Rafizah Mohd Hanifa	RAFIZAH MOHD HANIFA	rafizah mohd hanifa
assalammualaikum	ASSALAMMUALAIKUM	assalammualaikum

4.6.1.4 Compound Conditions

Compound conditions can be used to test more than one condition. Create compound conditions by joining conditions with logical operators. The logical operators are `Or`, `And`, and `Not`. Refer Table 4.11.

Table 4.11: Examples Use of Logical Operator as Compound Conditions

Logical Operator	Meaning	Example	Explanation

Or	If one condition or both conditions are True, the entire condition is True.	`Integer.Parse (lblNumber.Text) = 1 Or _`
		`Integer.Parse (lblNumber.Text) = 2`
			Evaluates True when lblNumber.Text is either "1" or "2"

And	Both conditions must be True for the entire condition to be True.	`Integer.Parse (lblNumber.Text) > 0 And _`
		`Integer.Parse (lblNumber.Text) < 10`
			Evaluates True when lblNumber.Text is "1","2","3","4","5","6","7","8", or "9",

Not	Reverses the condition so that a True condition will evaluate False and vice versa.	`Not Integer.Parse (lblNumber.Text) = 0`	Evaluates True when lblNumber.Text is any value other than "0"

4.6.2 Selection Control Structure

A powerful capability of the computer is its ability to make decisions and to take alternate courses of actions based on the outcome.

4.6.2.1 `If` Statements

A decision made by the computer is formed as a question: Is a given condition true or false? If it is **true**, do **one thing**; if it is **false**, do **something else**.

Example 1 (Refer Figure 4.13):

```
If the sun is shining Then              (condition)
        go to the beach                 (action to take if condition is true)
Else
        go to DOT NET class             (action to take if condition is false)
End If
```

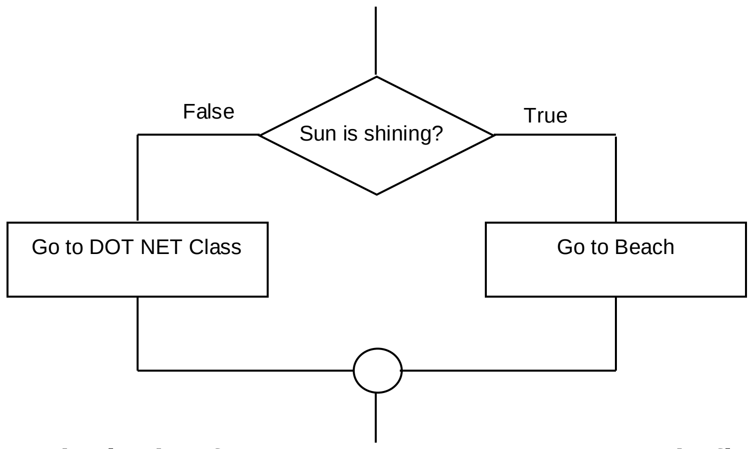
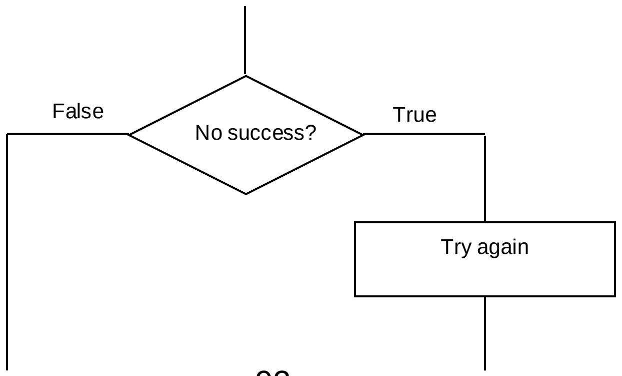


Figure 4.13: The logic of an `If`…`Then`…`Else` statement in flowchart form

Example 2 (Refer 4.14):

```
If you do not succeed Then              (condition)
        Try again                       (action)
End If
```



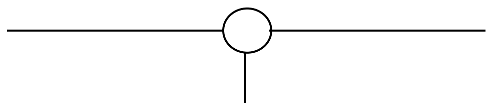


Figure 4.14: The logic of an `If` statement without an `Else` action in flowchart form

Notice in the second example that no action is specified if the condition is not true (false). In an `If` statement, when the condition is true, only the `Then` clause is executed. When the condition is false, only the `Else` clause, if present, is executed. A block `If…Then…Else` must always conclude with `End If`. The word `Then` must appear on the same line as the `If` with nothing following `Then` (except a remark). `End If` and `Else` (if used) must appear alone on a line. The statements under the `Then` and `Else` clauses are indented for readability and clarity.

Example 3:

```
decUnits = Decimal.Parse(txtUnits.Text)
If decUnits < 32D Then
     radFreshman.Checked = True
Else
     radFreshman.Checked = False
End If
```

When the number of units in **`decUnits`** is **less than 32**, **select** the **radio button** for Freshman; **otherwise**, make sure the **button** is **unselected**. Remember that when a radio button is selected, the **Checked** property has a Boolean value of **True**.

4.6.2.2 Nested `If` Statements

In many programs another `If` statement is one of the statements to be executed when a condition tests True or False. `If` statements that contain additional `If` statements are to be nested `If` statements. The following example shows a nested `If` statement in which the second `If` occurs in the `Then` portion of the first `If`.

Example of Code Segment for Nested If (Refer Figure 4.15):

```
If intTemp > 32 Then
     If intTemp > 80 Then
          lblComment.Text = "Hot"
     Else
          lblComment.Text = "Moderate"
     End If
Else
     lblComment.Text = "Freezing"
End If
```

To nest `If` statements in the `Else` portion, you may also use `ElseIf…Then` which is simpler.

Example:

```
If intTemp <= 32 Then
     lblComment.Text = "Freezing"
Else
     If  intTemp > 80 Then
     lblComment.Text = "Hot"
     Else
     lblComment.Text = "Moderate"
     End If
End If
```

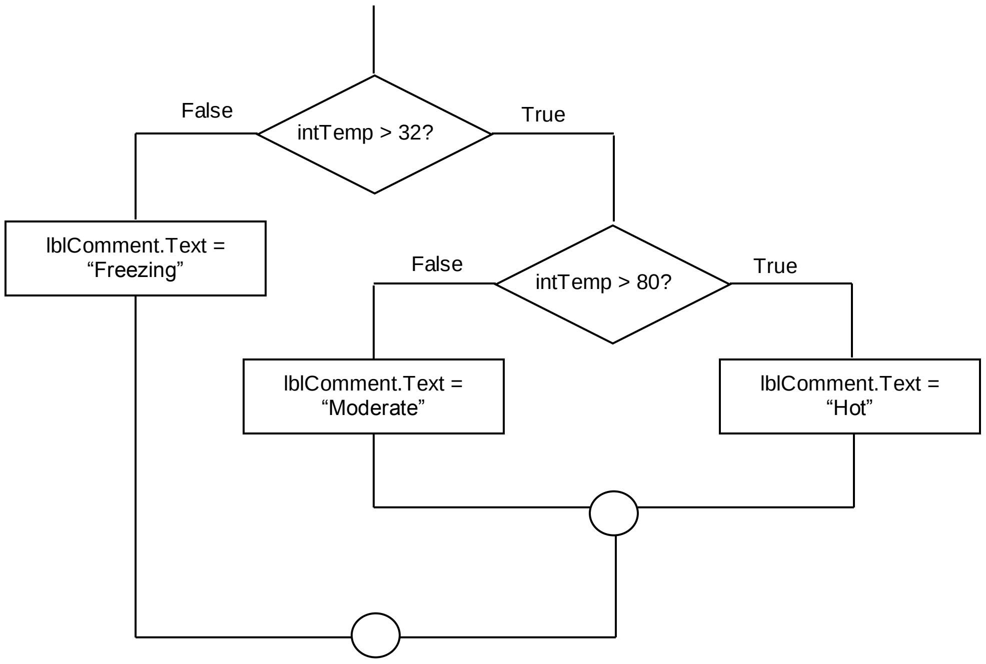


Figure 4.15: Flowcharting a nested `If` statement

Nest `If`s can be done in both the `Then` and `Else`. In fact, it may continue to nest `If`s within `If`s as long as each `If` has an `End If`. However, projects become very difficult to follow (and may not perform as intended) when `If`s become too deeply nested.

An example use Nested if for radio button control (Figure 4.16):

```
If radMale.Checked = True Then
     If Integer.Parse(txtAge.Text) < 21 Then
          mintMinorMaleCount += 1
     Else
          mintMaleCount += 1
     End If
Else
     If Integer.Parse(txtAge.Text) < 21 Then
          mintMinorFemaleCount += 1
     Else
          mintFemaleCount += 1
     End If
End If
```

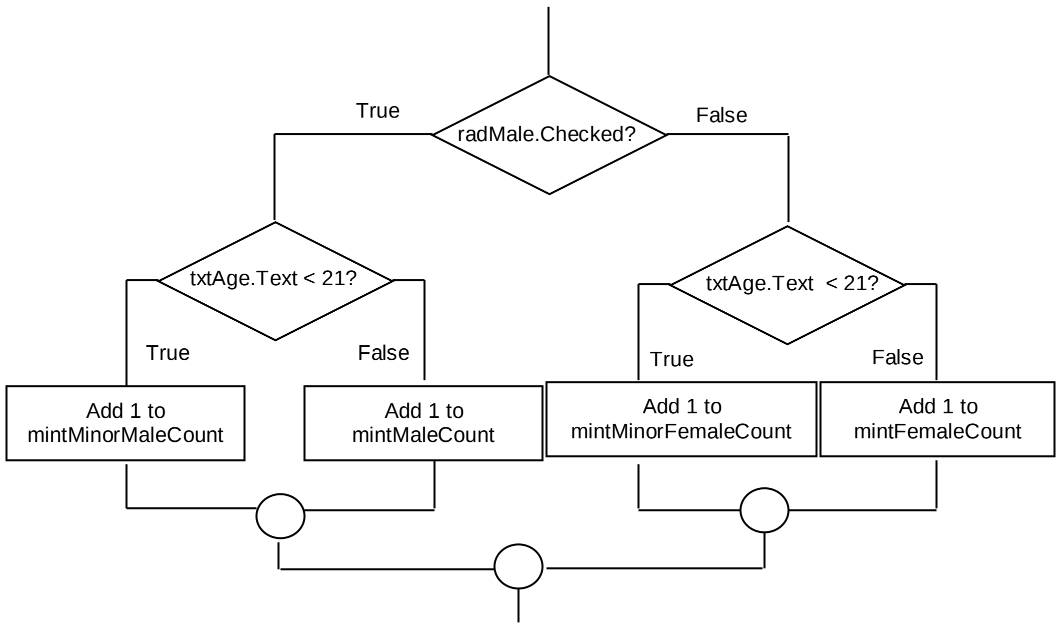


Figure 4.16: A Flowchart of a Nested `If` Statement with `If`s nested on both sides of the original `If`

4.6.2.3 Select Case Structure

To test a single variable for multiple values, the `Case` structure provides a **flexible** and **powerful** solution. Any decisions that you can code with a `Case` structure also can be coded with nested `If` statements, but usually the `Case` structure is simpler and clearer. The expression in a `Case` structure is usually a variable or property that you wish to test. The constant list is the value that you want to match; it may be a numeric or string constant or variable, a range of values, a relational condition, or a combination of these. Refer examples in Table 4.12.

Table 4.12: Examples Use of Case Statement

Case statement with string	Case statement with numbers

`Dim creamcake As String`
`Dim DietState As String`
`creamcake = txtChoose.Text`
 	
` Select Case creamcake`
` Case "Eaten"`
`     DietState = "Diet Ruined"`
` Case "Not Eaten"`
`     DietState = "Diet Not Ruined"`
` Case Else`
`     DietState = "Didn't check"End`
` End Select`
 	
` MsgBox DietState`
	`Dim agerange As Integer`
	
	`agerange = txtage.Text`
	
	`Select Case agerange`
	`       Case 16 To 21`
	`       MsgBox "Still Young"`
	`       Case 50 To 64`
	`       MsgBox "Start Lying"`
	`End Select`

Checking text in the variable	Checking a variable to see if the number that was in the variable fell within a certain age-range

4.6.2.4 Use Selection Structure for Check Box to a VB .NET form

Checkbox is one of control which provide user options to select. There are three options of properties for **CheckState** Property: Unchecked, Checked and Indeterminate (Figure 4.17a and Figure 4.17b).

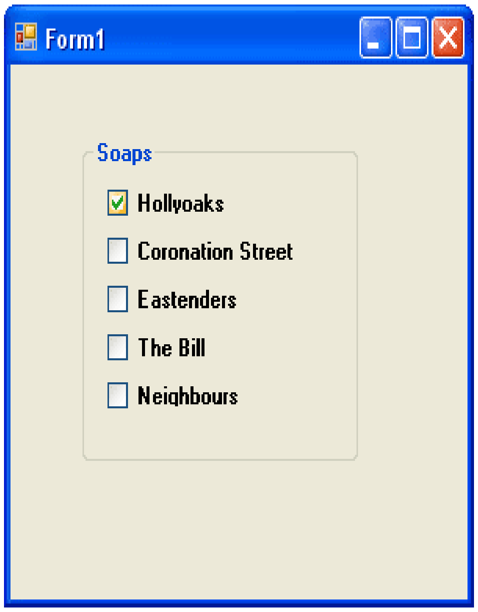
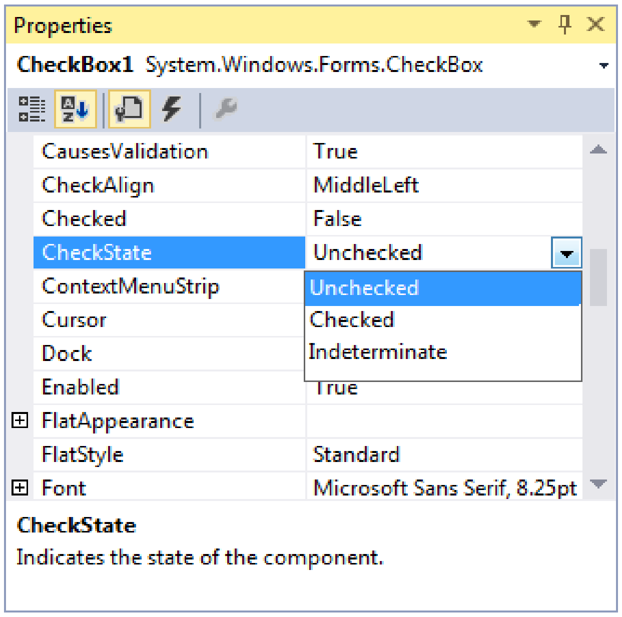


Figure 4.17a Interface Design of Checkbox


Properties

CheckBox1 System.Windows.Forms.CheckBox

CausesValidation	True
CheckAlign	MiddleLeft
Checked	False
CheckState	Unchecked
ContextMenuStrip	Unchecked
Cursor	Checked
Dock	Indeterminate
Enabled	True
FlatAppearance	
FlatStyle	Standard
Font	Microsoft Sans Serif, 8.25pt

CheckState
Indicates the state of the component.

Figure 4.17b Property Setting For Checkbox

If a checkbox has been selected, the value for the CheckState property will be 1; if it hasn't been selected, the value is zero. Only going to test for 0 or 1, Checked or Unchecked. The testing can be done with a simple If Statement (Refer Table 4.13).

Table 4.13: Example of If Statement for a CheckBox Control

Style 1	Style 2

`If CheckBox1.CheckState = CheckState.Checked Then`
`MsgBox("Checked")`
`Else`
`MsgBox("Not Checked")`
`End If`
	`If CheckBox1. CheckState = 1 Then`
	`MsgBox "Checked"`
	`ElseIf Checkbox1. CheckState = 0 Then`
	`MsgBox "Unchecked"`
	`End If`

4.6.3 Repetition Control Structure

Repetition Control Structure or a loop is something that goes round and round and round. In fact, they go round and round until you tell them to stop. It can programme without using loops. But it's an awful lot easier with them. Consider this: to add up the numbers 1 to 4: 1 + 2 + 3 + 4. It may do like this:

```
Dim answer As Integer
answer = 1 + 2 + 3 + 4
MsgBox answer
```

It looks fairly simple. And not much code, either. But what if trying to add up a thousand numbers? It is really going to type them all out like that? It's an awful lot of typing. A loop would make life a lot simpler.mThe computer is capable of repeating a group of instructions many times without calling the procedure for each new set of data. The **process** of **repeating** a **series** of **instructions** is called **looping**. The **group** of **repeated instructions** is called a **loop**. Iteration is a single looping execution of the statement(s) in the loop.

4.6.3.1 For- Next Loops

To repeat the statements for specific number of times, the `For/Next` loop is ideal. This loop uses the `For` and `Next` statements and a counter variable, called the **loop index**. The loop index is tested to determine the number of times the statements inside the loop will executed. A counter-controlled loop generally has three elements: initialize the counter, increment the counter, and test the counter to determine when the loop will terminate (refer to Figure 4.18).

Syntax:

```
For LoopIndex = InitialValue to TestValue [Step Increment]
      `
      ` Statements in loop
      `
Next [LoopIndex]
```

`LoopIndex` must be a numeric variable; `InitialValue` and `TestValue` may be constants, variables, numeric property values, or numeric expressions. The optional word `Step` may be included, along with the value to be added to the loop index for each iteration of the loop. When the `Step` is omitted, the increment is assumed to be 1.

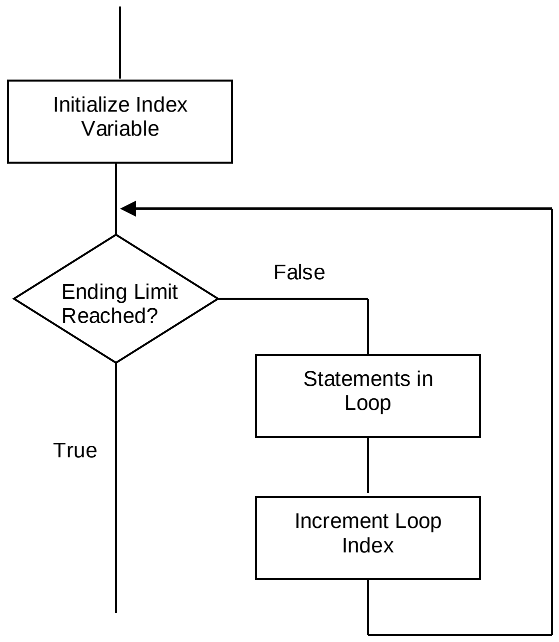


Figure 4.18: A flowchart of the logic of a `For/Next` loop

Examples:

i. `For intIndex = 2 To 100 Step 2`

ii. `For intCount = intStart to intend Step intIncrement`

iii. `For intCountDown = 10 To 0 Step -1`

iv.
```
Dim answer As Integer
Dim startNumber As Integer
answer = 0
For startNumber = 1 To 4
    answer = answer + startNumber
Next startNumber
MsgBox answer
```

As Conclusion, A For loop needs a start position and an end position, and all on the same line and also needs a way to get the next number in the loop.

4.6.3.2 Do Loops

A `Do/Loop` terminates based on a condition that you specify. Execution of a `Do/Loop` continues while a condition is True or until a condition is True. You can choose to place the condition at the top or the bottom of the loop. Use a `Do/Loop` when the exact number of iterations is unknown. The first form of the `Do/Loop` tests for completion at the top of the loop. With this type of loop, also called a **pretest**, the statements inside the loop may never be executed if the terminating condition is True the first time it is tested.

Example:

```
intTotal = 0
Do Until intTotal = 0
     'Statements in loop
Loop
```

Because `intTotal` is 0 the first time the condition is tested, the condition is True and the statements inside the loop will not execute. Control will pass to the statement following the `Loop` statement.

The second form of the `Do/Loop` tests for completion at the bottom of the loop, which means that the statements inside the loop will always be executed at least once. This form of loop is sometimes called a **posttest**. Changing the example to a posttest, you can see the difference.

Example:

```
intTotal = 0
Do
     'Statements in loop
```

```
Loop Until intTotal = 0
```

In this case, the statements inside the loop will be executed at least once. Assuming the value for intTotal does not change inside the loop, the condition (`intTotal = 0`) will be True the first time it is tested and control will pass to the first statement following the `Loop` statement. Figure 4.19 shows flowcharts of pretest and Figure 4.20 shows flowcharts of posttest loops using both `While` and `Until`.

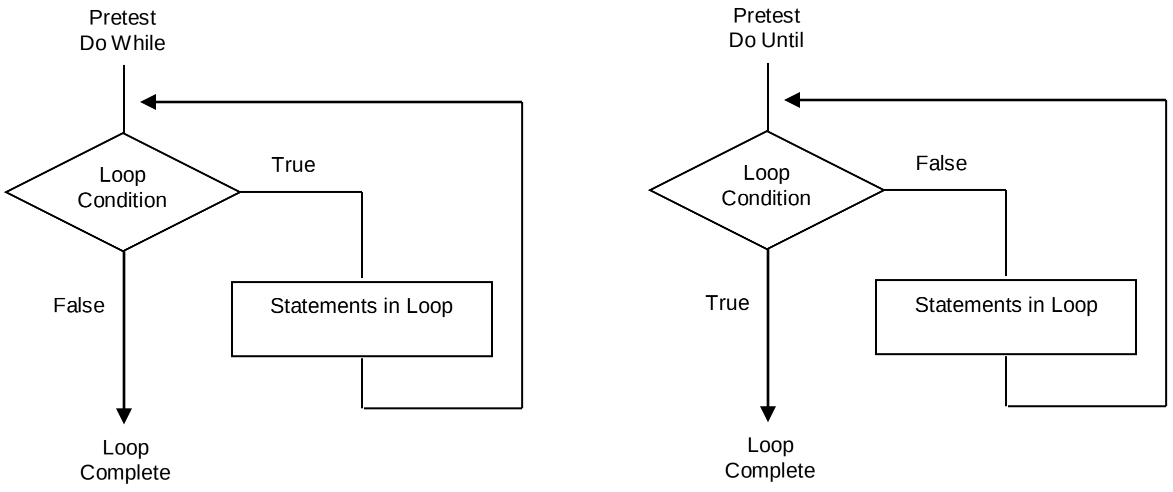


Figure 4.19: Flowcharts of pretest for `Do While` and `Do Until`

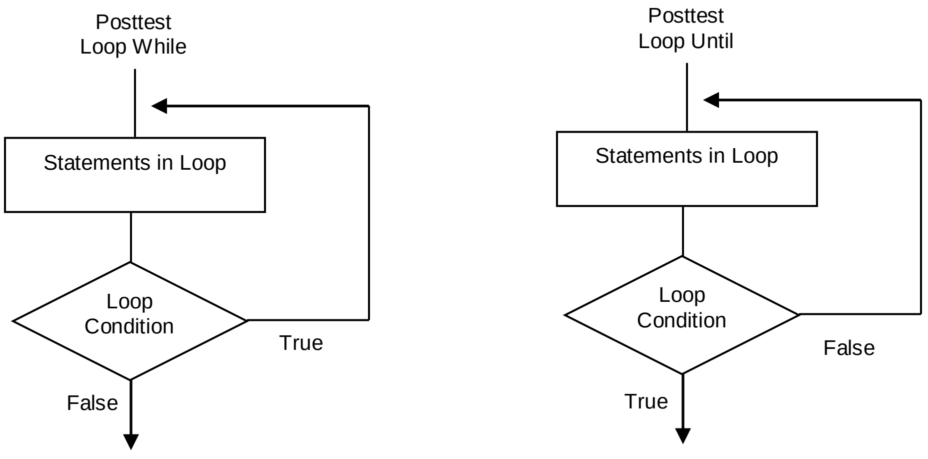
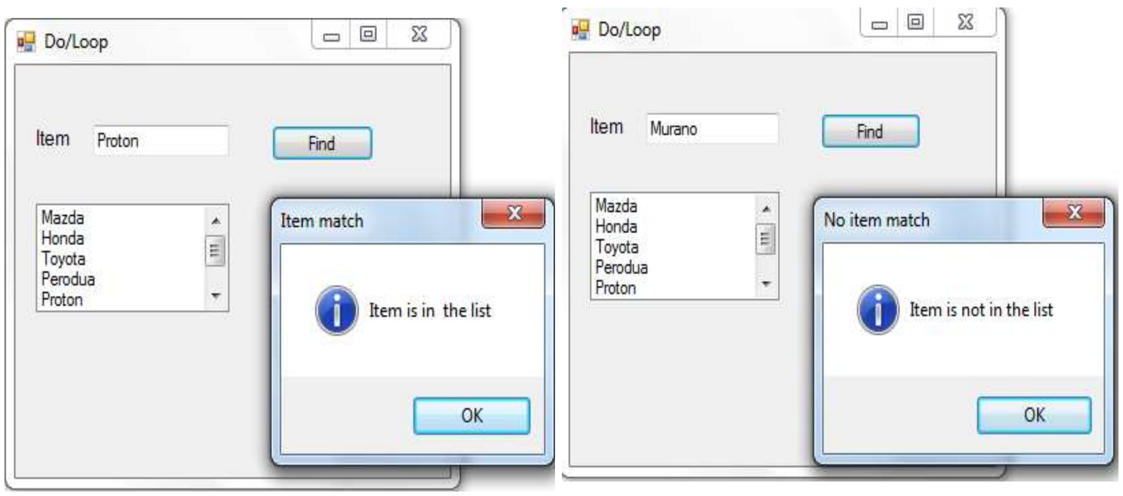


Figure 4.20: Flowcharts of posttest for `Do While` and `Do Until`

Boolean data type, which holds only the values True or False. Boolean variables useful when setting and testing conditions for a loop. This can be done by setting a Boolean variable to True when a specific circumstance occurs and then write a loop condition until the variable is True. An example of using a Boolean variable is when you want to search through a list for a specific value. The item may be found or not found, and you want to quit looking when a match is found. Using a Boolean variable is usually a three-step process. First, you must dimension a variable and set its initial value (or use the default VB setting of **False**). Then, when a particular situation occurs, you set the variable to **True**. A loop condition can then check for **True**.

Example:

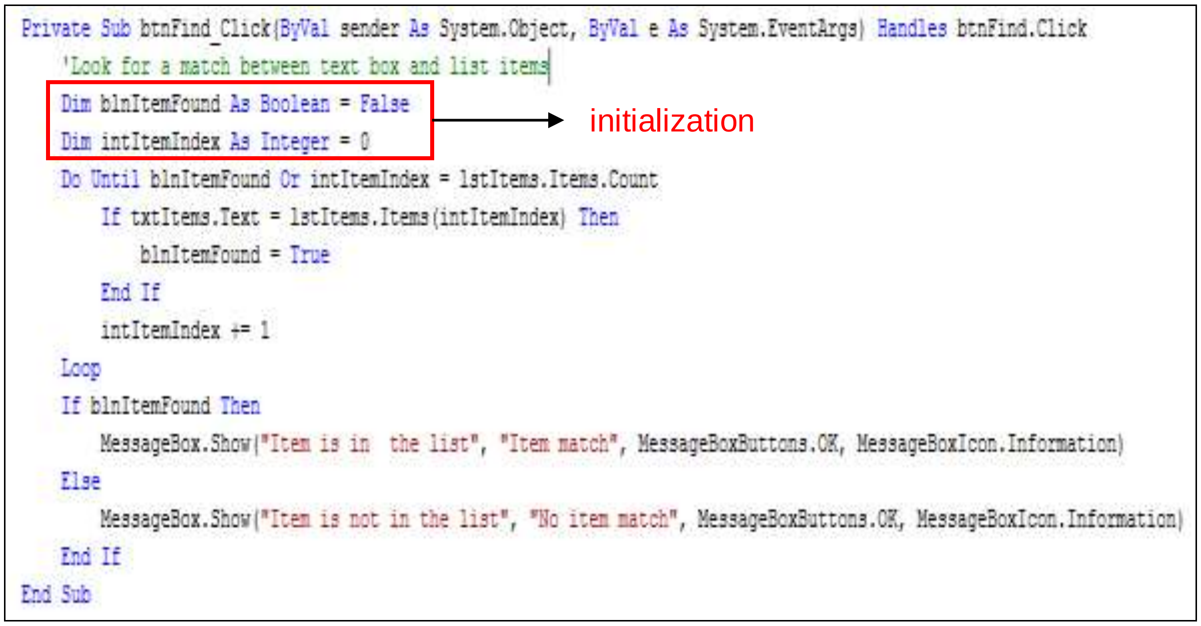


Item is in the list

Item is not in the list

Figure 4.21: Example use Loop for Searching Specific Value from a List

In the example, each element of the list is compared to `newItemTextBox.Text` (Proton) for a match. The loop will terminate when a match is found or when all elements have been tested. If the matches failed, the dialog message will pop up as shown in the diagram below.



```
Private Sub btnFind_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnFind.Click
    'Look for a match between text box and list items
    Dim blnItemFound As Boolean = False
    Dim intItemIndex As Integer = 0
    Do Until blnItemFound Or intItemIndex = lstItems.Items.Count
        If txtItems.Text = lstItems.Items(intItemIndex) Then
            blnItemFound = True
        End If
        intItemIndex += 1
    Loop
    If blnItemFound Then
        MessageBox.Show("Item is in  the list", "Item match", MessageBoxButtons.OK, MessageBoxIcon.Information)
    Else
        MessageBox.Show("Item is not in the list", "No item match", MessageBoxButtons.OK, MessageBoxIcon.Information)
    End If
End Sub
```

Figure 4.22: Example VB code to implement Do Loop

4.7 Array and Collections

So far, variables have been used quite a lot. Numbers or text have been put into variables. But, it was only done this one at a time: one number or one string of text have been put into a variable. This code have been done:

```
Dim MyNumber As Integer
MyNumber = 4
```

Or this

```
Dim MyText As String
MyText = "A String is really just text"
```

Or even this:

```
Dim MyNumber As Integer = 4
```

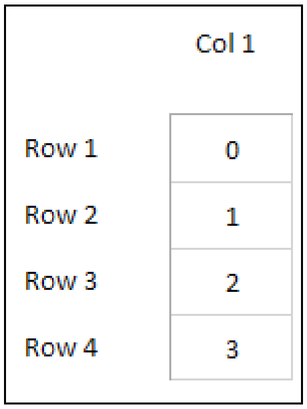
An array is a variable that can hold more than one piece of information at a time. The `MyNumber` variable above held one number 4. While, an array variable called `MyNumbers` could hold more than one number at a time.

```
Dim MyNumbers(4) As Integer

MyNumbers(0) = 0
MyNumbers(1) = 1
MyNumbers(2) = 2
MyNumbers(3) = 3
```

When set up an array with the **Dim** word, put the name of your array variable, and tell VB how many items stored in the array. Then, assign data to each position in the array. So, that's what an array is - **a variable that can hold more than one piece of data at a time**

4.7.1 Single-Dimension

One-dimensional (1-D) arrays are by far the most useful, however there are some uses for multi-dimensional arrays. A 1-D array holds one column of data. Visually, it looks like this (four rows with one value in each column) for `MyNumbers(4)` as above example.



4.7.1.2 Assigning Values to an Array

There are a number of way you can put data into each position of an array. Values can be assign straight from a Textbox into the position of your array. Like this:

```
MyNumbers(0) = Val(Textbox1.Text)
MyNumbers(1) = Val(Textbox2.Text)
etc
```

With that code, whatever you typed into the Textboxes on your Form would be stored into the positions of your array. The same would be true of a String Array:

```
MyNumbers(0) = Textbox1.Text
MyNumbers(1) = Textbox2.Text
etc
```

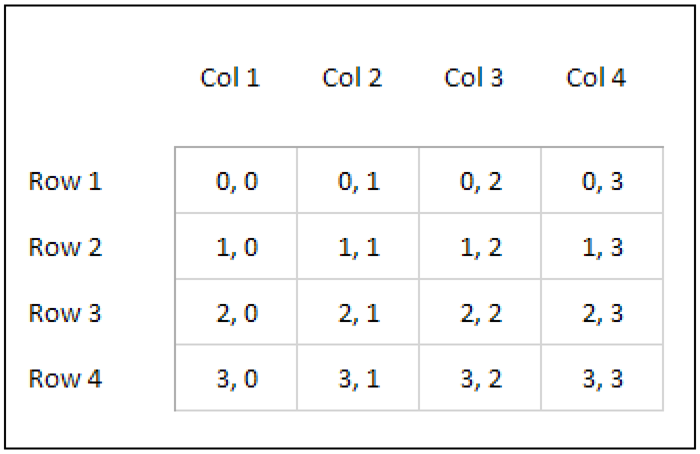
But do a programmer have to keep typing out a value for each and every position of our array. What if had an array with a hundred items in it, **MyText(99)**? Would have to type out text for all one hundred positions of the array?

Well, obviously not. VB code can be used to assign values to array. Here is an example where we don't type out values for all positions of an array. It's the times table. This time we'll use an array. And write a line of code to assign values to each position of the array and use looping statement.

```
Dim numbers(10) As Integer
Dim times As Integer
Dim StoreAnswer As Integer
Dim i As Integer
ListBox1.Items.Clear()
times = Val(TextBox1.Text)
For i = 1 To 10
StoreAnswer = i * times
       numbers(i) = StoreAnswer
       ListBox1.Items.Add(times & " times " & i & "
       = " & numbers(i))
Next i
```

4.7.2 Two-Dimension Array

Another type of array is a Two-Dimensional (2-D) array. This holds data in a grid pattern, rather like a spreadsheet. They look like this (4 rows, and 4 columns):



	Col 1	Col 2	Col 3	Col 4
Row 1	0, 0	0, 1	0, 2	0, 3
Row 2	1, 0	1, 1	1, 2	1, 3
Row 3	2, 0	2, 1	2, 2	2, 3
Row 4	3, 0	3, 1	3, 2	3, 3

To set up a 2-D array in Visual Basic .NET you just add a second number between the round brackets:

```
Dim grid(3, 3) As Integer
```

This means fill row 0, column 0 with a value of 1. You could type out all your positions and values like this: (The left number between the round brackets is always the row number; the right number is always the columns.)

```
grid(0, 0) = 1
grid(1, 0) = 2
grid(2, 0) = 3
grid(3, 0) = 4

grid(0, 1) = 5
grid(1, 1) = 6
grid(2, 1) = 7
grid(3, 1) = 8

grid(0, 2) = 9
grid(1, 2) = 10
grid(2, 2) = 11
grid(2, 2) = 12
```

```
        grid(0, 3) = 13
        grid(1, 3) = 14
        grid(2, 3) = 15
        grid(3, 3) = 16
```

Typically, though, 2-D arrays are filled using a double for loop. The following code just places the numbers 1 to 16 in the same positions as in the example above:

```
Dim grid(3, 3) As Integer
Dim row As Integer
Dim col As Integer
Dim counter As Integer = 1
For row = 0 To 3
    For col = 0 To 3
        grid(row, col) = counter
        counter = counter + 1
    Next
Next
```

4.7.3 Applications of Array in List Boxes and Combo Boxes

List boxes and combo boxes allow a list of items from which the user can make a selection. Figure 4.23 shows the toolbox tools for creating the controls.

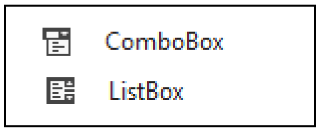


Figure 4.23: ListBox and ComboBox Tools to Create list boxes and combo boxes in forms

ListBox controls and ComboBox controls have most of the **same properties** and **operate** in a **similar fashion**. **One exception** is that a **ComboBox** control has a **DropDownStyle** property, which determines whether or not the list box also has a text box for user entry and whether or not the list will drop down.

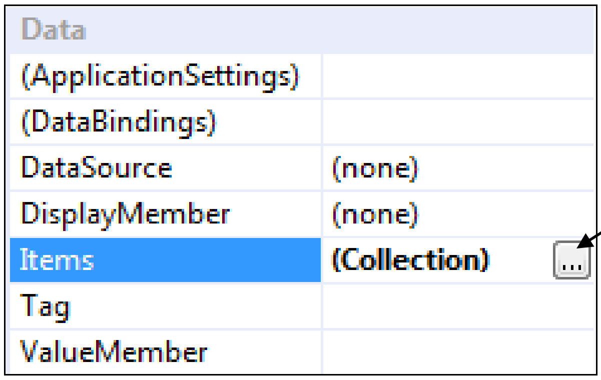
Both list boxes and combo boxes have a great feature. If the box is too small to display all the items in the list at one time, VB automatically adds a scroll bar. When you add a list control to a form, choose the style according to the space you have available and how you want the box to operate. At design time, the behavior of list boxes and combo boxes differs. For list boxes, VB displays the Name property in the control; for combo boxes, the Text property displays.

4.7.3.1 Items Collection

The list of items that displays in a list box or combo box is a collection. VB collections are objects that have properties and methods to allow programmer to add items, remove items, refer to individual elements, count the items, and clear the collection. The items in a collection can refer by an index, which is zero based. For example, if a collection holds 10 items, the indexes to refer to the items range from 0 to 9. To refer to the first item in the Items collection, use `Items(0)`.

Several methods can be used to fill the Items collection of a list box and combo box. If the list contents known at design time and the list never changes, Items collection can be define in the Properties window. If the items must added to the list during program execution, use the `Items.Add` or `Items.Insert` method in an event procedure.

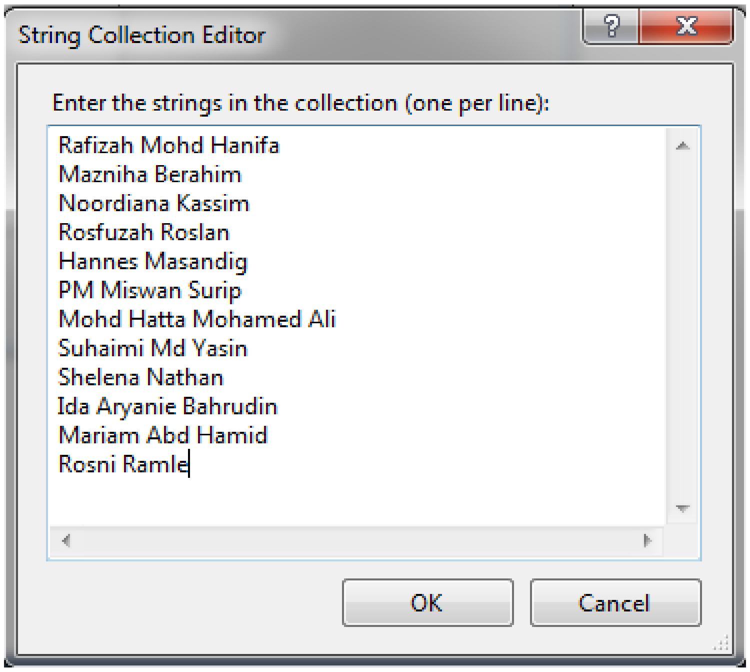
The Items property, which is a collection, holds the list of items for a list box or combo box. To define the Items collection at design time, select control and scroll the Properties window to the Items property (refer to Figure 4.24).



Data	
(ApplicationSettings)	
(DataBindings)	
DataSource	(none)
DisplayMember	(none)
Items	(Collection)
Tag	
ValueMember	

Figure 4.24: Select the Items Property of a List Box to Enter the List Items

Click on the ellipses button to open the String Collection Editor (refer to Figure 4.25) and type your list items, ending each line with the Enter key. Click OK when finished.



```
Rafizah Mohd Hanifa
Mazniha Berahim
Noordiana Kassim
Rosfuzah Roslan
Hannes Masandig
PM Miswan Surip
Mohd Hatta Mohamed Ali
Suhaimi Md Yasin
Shelena Nathan
Ida Aryanie Bahrudin
Mariam Abd Hamid
Rosni Ramle
```

Figure 4.25: Type each list item and press Enter to go to the next line

To add an item to a list at run time, use the `Items.Add` method. A variable, a constant, the contents of the text box can be choosen to add at the top of a combo box, or the Text property of another control. The new item generally goes at the end of the list. However, you can alter the placement by setting the control’s **Sorted property** to True. Then the new item will be placed alphabetically in the list.

Examples:

```
lstUniversity.Items.Add(“UTHM”)
lstUniversity.Items.Add(“USM”)
lstUniversity.Items.Add(txtUniversity.Text)
cboMajor.Items.Add(cboMajor.Text)
```

When the user types a new value in the text box portion of a combo box, that item is not automatically added to the list. If you want to add the newly entered text to the list, use the `Items.Add` method.

The location can be choosen for a new item added to the list. In the `Item.Insert` method, you specify the index position for the new item. The index position is zero based. To insert a new item in the first position, use index position = 0. If you choose the index position of an item using the Insert method, do not set the list control’s Sorted property to True. A sorted list is always sorted into alphabetic order, regardless of any other order.

Examples:

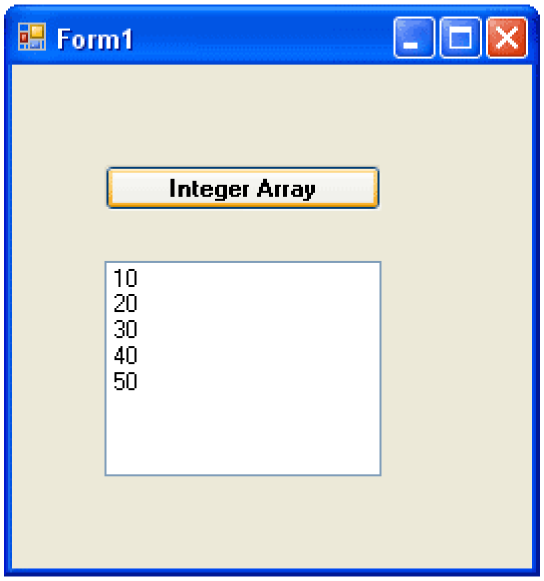
```
lstUniversity.Items.Insert(0, “UUM”)
lstUniversity.Items.Insert(1, “UTM”)
```

Another code segment as example which use an array to add items in a list of number and display in a List Box after executed.

```
Dim MyNumbers(4) As Integer

MyNumbers(0) = 10
MyNumbers(1) = 20
MyNumbers(2) = 30
MyNumbers(3) = 40
MyNumbers(4) = 50
```

```
        For i = 0 To 4
            listNumbers.Items.Add(MyNumbers(i))
        Next i
```

After executed above code. A list box will list a collection of numbers in the form where it designed and should look something like Figure 4.26:



```
10
20
30
40
50
```

Figure 4.26: A List Box List an Array of Numbers

4.7.3.2 `SelectedIndex` Property

When a project is running and the user selects (highlights) an item from the list, the index number of that item is stored in the **SelectedIndex property** of the list box. Recall that the index of the first item in the list is 0. If no list item is selected, the SelectedIndex property is set to negative 1 (-1). The SelectedIndex property can be used to select an item in the list or deselect all items in code.

Examples:

```
'Select the fourth item in list
lstCoffeeType.SelectedIndex = 3
'Deselect all items in list
lstCoffeeType.SelectedIndex = -1
```

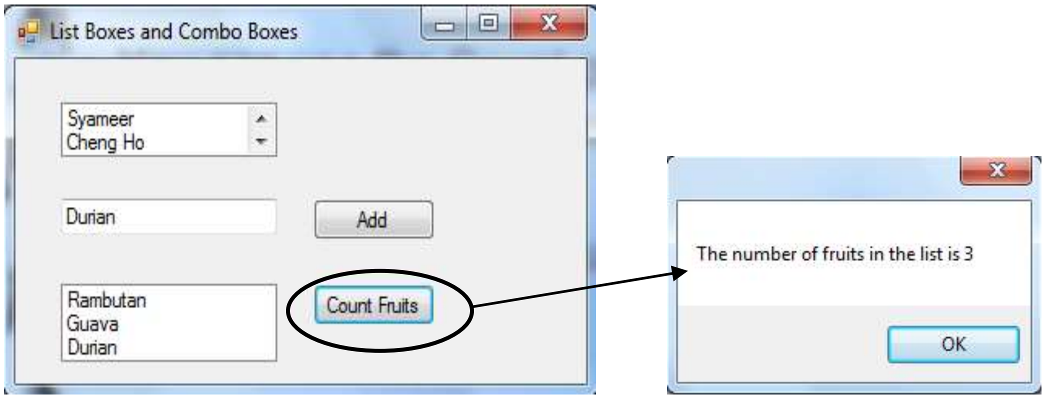
4.7.3.3 `Items.Count` Property

The Count property of the Items collection can be used to determine the number of items in the list. This can be done by using `Items.Count` property.

Example:

```
Private Sub btnCount_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnCount.Click
    Dim intTotal As Integer
    intTotal = lstFruit.Items.Count
    MessageBox.Show("The number of fruits in the list is " & intTotal)
End Sub
```

By clicking the Count Fruits button, the Message Dialog showing the value will be displayed (Refer 4.27).



The number of fruits in the list is 3

Figure 4.27: Message Dialog Appear Total Items in a List

4.7.3.4 Removing Item from a List

Individual items can remove from a list, by specifying either the index of the item or the text of the item. Use the `Items.RemoveAt` method to remove an item by index and the `Items.Remove` method to remove by specifying the text.

Examples:

```
lstStudent.Items.RemoveAt(0)
lstStudent.Items.Remove("Syazana")
```

4.7.3.5 Clearing a List

In addition to removing individual items at run time, you can also clear all items from a list. Use the `Items.Clear` method to empty a combo box or list box.

Example:

```
lstFruit.Items.Clear()
```

Table 4.14 summarizes the purpose of each of the following methods or properties for a ListBox or ComboBox control.

Table 4.14: Purpose of Methods

Methods	**Explanation**
`Items.Count`	Stores the number of element in a list box or combo box.
`Items.Add`	Adds an element to a list at run time.
`Items.Insert`	Adds an element to a list and inserts the element in the chosen position (index).
`Items.Clear`	Clear all elements from a list box or combo box.
`Items.RemoveAt`	Removes an element from the list by referring to its index.
`Items.Remove`	Removes an element from a list by looking for a given string.

Activities

1. Indicate whether each of the following identifiers conforms to the rules of Basic and to the naming conventions. If the identifier is invalid, give the reason.

(a) Omitted	(d) strSub
(b) Int#Sold	(e) Text
(c) Int Number Sold	(f) decMaximumCheck

2. Write a declaration for the following situations. In your declaration, make up an appropriate variable identifier.

   (a) You need variables for payroll processing to store the following:
   - Number of hours, which can hold a decimal point.
   - String employee's name.
   - Department number (not used in calculations).

   (b) You need variable for inventory control to store the following:
   - Integer quantity.
   - Description of the item.
   - Part number.
   - Cost.
   - Selling price.

3. Write the declarations (`Dim` or `Const` statements) for each of the following situations and indicate where each statement will appear.

   (a) The total of the payroll that will be needed in a `Calculate` event procedure and in a `Summary` event procedure.

   (b) The sales tax rate that cannot be changed during execution of the program but will be used by multiple procedures.

(c) The number of participants that are being counted in the `Calculate` event procedure, but not displayed until the `Summary` event procedure.

4. What will be the result of the following calculations using the order of precedence? Assume that `intX = 2`, `intY = 4`, `intZ = 3`.

   (a) intX + intY ^ 2

   (b) 8 / intY / intX

   (c) intX * (intX + 1)

   (d) intY ^ intX + intZ * 2

   (e) ((intY ^ intX) + intZ) * 2

5. Put two textboxes on your form. The first box asks users to enter a start position for a For Loop; the second textbox asks user to enter an end position for the `For` loop. When a button is clicked, the programme will add up the numbers between the start position and the end position. Display the answer in a message box. You can use this For Loop code

```
For i = startNumber To endNumber
   answer = answer + i
Next i
```

   Get the **`startNumber`** and **`endNumber`** from the textboxes.

6. When should you use `Try/Catch` blocks? Why?

References

1. N. Randolph and D. Gardner, 2009. ***Professional Visual Studio 2008,*** Wiley Publishing.
2. J.C. Bradley and A.C. Millspaugh, 2005. ***Programming in Visual Basic .NET***, McGraw Hill Technology Education.
3. D.I. Schneider, 2006. ***An Introduction to Programming Using Visual Basic 2005***. Prentice Hall.
4. VB.Net Programming Tutorial available at https://www.tutorialspoint.com/vb.net/
5. Variable and Method Scope in Microsoft .NET. retrieved from https://msdn.microsoft.com/en-us/library/ms973875.aspx

TOPIC 5

WEB APPLICATION DEVELOPMENT

Learning Outcome

At the end of this topic, student should be able to:

1. Describe ASP.NET Web Forms with its features, and
2. Design a web page and develop a web application

Content

5.1 Introduction

A dynamic web application consists of either or both of the following two types of programs:

i. Server-side scripting - these are programs executed on a web server, written using server-side scripting languages like ASP (Active Server Pages) or JSP (Java Server Pages).

ii. Client-side scripting - these are programs executed on the browser, written using scripting languages like JavaScript, VBScript, etc.

ASP.Net is the .Net version of ASP, introduced by Microsoft, for creating dynamic web pages by using server-side scripts. ASP.Net applications are compiled codes written using the extensible and reusable components or objects present in .Net framework. These codes can use the entire hierarchy of classes in .Net framework. The ASP.Net application codes could be written in either of the following languages: Visual Basic .Net, C#, Jscript or J#.

In this chapter, we will give a very brief introduction to writing ASP.Net applications using VB.NET.

5.2 ASP.NET Web Forms

ASP.NET Web Forms is a part of the ASP.NET web application framework and is included with Visual Studio. ASP.Net provides **Web Forms** programming models enables to create the user interface and the application logic that would be applied to various components of the user interface The Visual Studio Integrated Development Environment (IDE) is almost same as you have already used for creating the Windows Applications. It let drag and drop server controls to lay out a Web Forms page. The properties, methods, and events can easily set for controls on the page or for the page itself. These properties, methods, and events are used to define the web page's behavior, look and feel, and so on. To write server code to handle the logic for the page, VB. Net language cab be used.

Web Forms are pages that your users request using their browser. These pages can be written using a combination of HTML, client-script, server controls, and server code. When users request a page, it is compiled and executed on the server by the framework, and then the framework generates the HTML markup that the browser can render. An ASP.NET Web Forms page presents information to the user in any browser or client device.

Web forms consists of user interface and application logic. User interface consists of static HTML or XML elements and ASP.Net server controls. When you create a web application, HTML or XML elements and server controls are stored in a file with **.aspx** extension. This file is also called the page file. While, the application logic consists of code applied to the user interface elements in the page. You write this code in any of .Net language like, VB.Net, or C#.

5.2.1 Features of ASP.NET Web Forms

Basics features of ASP.Net Web Forms as below:

i. **Server Controls** - ASP.NET Web server controls are objects on ASP.NET Web pages that run when the page is requested and that render markup to the browser. Many Web server controls are similar to familiar HTML elements, such as buttons and text boxes. Other controls encompass complex behavior, such as a calendar controls, and controls that you can use to connect to data sources and display data.

ii. **Master Pages** - ASP.NET master pages allow you to create a consistent layout for the pages in your application. A single master page defines the look and feel and standard behavior that you want for all of the pages (or a group of pages) in your application. You can then create individual content pages that contain the content you want to display. When users request the content pages, they merge with the master page to produce output that combines the layout of the master page with the content from the content page.

iii. **Working with Data** - ASP.NET provides many options for storing, retrieving, and displaying data. In an ASP.NET Web Forms application, you use data-bound controls to automate the presentation or input of data in web page UI elements such as tables and text boxes and drop-down lists.

iv. **Membership** - ASP.NET Identity stores your users' credentials in a database created by the application. When your users log in, the application validates their credentials by reading the database. Your project's *Account* folder contains the files that implement the various parts of membership: registering, logging in, changing a password, and authorizing access. Additionally, ASP.NET Web Forms supports OAuth and OpenID. These authentication enhancements allow users to log into your site using existing credentials, from such accounts as

Facebook, Twitter, Windows Live, and Google. By default, the template creates a membership database using a default database name on an instance of SQL Server Express LocalDB, the development database server that comes with Visual Studio Express 2013 for Web.

v. **Client Script and Client Frameworks** - You can enhance the server-based features of ASP.NET by including client-script functionality in ASP.NET Web Form pages. You can use client script to provide a richer, more responsive user interface to users. You can also use client script to make asynchronous calls to the Web server while a page is running in the browser.

vi. **State Management** - ASP.NET Web Forms includes several options that help you preserve data on both a per-page basis and an application-wide basis.

vii. **Debugging and Error Handling** - ASP.NET includes features to help you diagnose problems that might arise in your Web Forms application. Debugging and error handling are well supported within ASP.NET Web Forms so that your applications compile and run effectively.

viii. **Deployment and Hosting** - Visual Studio, ASP.NET, Azure, and IIS provide tools that help you with the process of deploying and hosting your Web Forms application.

5.2.2 Advantages of a Web Forms-Based Web Application

The Web Forms-based framework offers the following advantages:

i. Support event model on the Hypertext Transfer Protocol (HTTP) which is provides hundreds of events that compatible in a lot of server control

ii. It has a mechanism to control pages (Page Controller Pattern) which is provide the functionality to single pages.

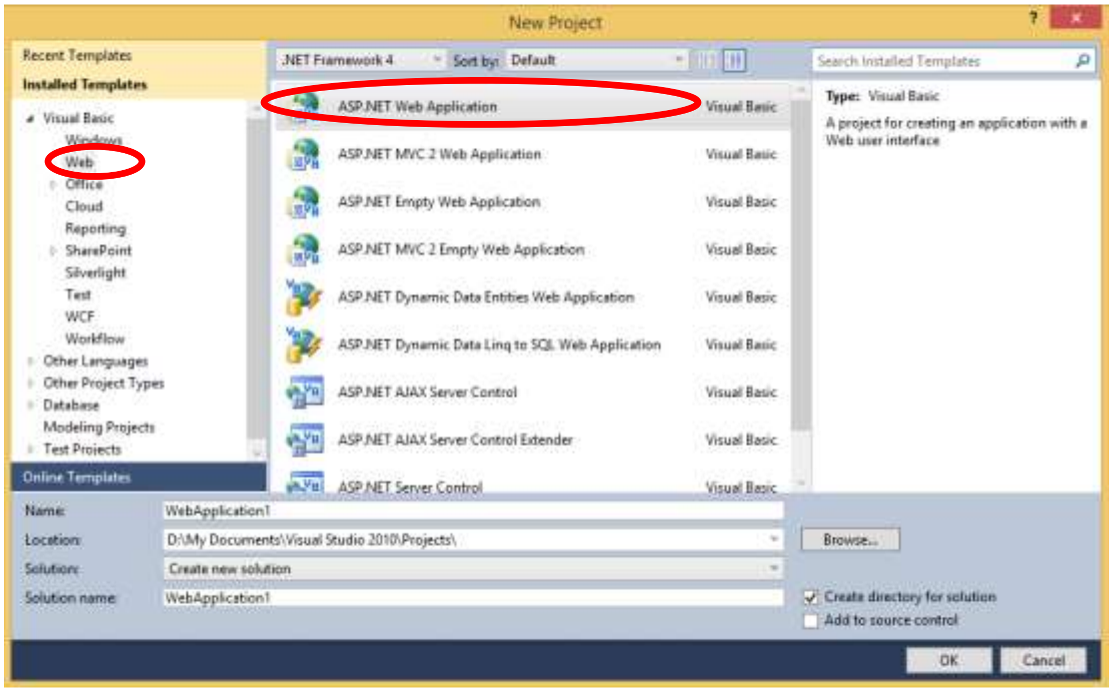
iii. It make easier to manage state information based on server based form or view state

iv. Not requires many programmer and designer to develop web project because it has the advantage of rapid application development

5.3 Creating a Web Application Project and a Page

An ASP.NET application, you must have a Web browser: Microsoft Internet Explorer or another browser, such as Chrome and a Web server: That is, a computer that is running IIS. This can be the computer you are developing on or any remote computer that you can connect to. A means to connect to the Web server.

To create a web application project, open Visual Studio. On the File Menu, select New Project. AT the New Project Dialog as Figure 5.1, Select ASP.NET Web Application.


New Project
Recent Templates
Installed Templates
Visual Basic
Windows
Web
Office
Cloud
Reporting
SharePoint
Silverlight
Test
WCF
Workflow
Other Languages
Other Project Types
Database
Modeling Projects
Test Projects
Online Templates
.NET Framework 4
Sort by: Default
Search Installed Templates
ASP.NET Web Application Visual Basic
ASP.NET MVC 2 Web Application Visual Basic
ASP.NET Empty Web Application Visual Basic
ASP.NET MVC 2 Empty Web Application Visual Basic
ASP.NET Dynamic Data Entities Web Application Visual Basic
ASP.NET Dynamic Data Linq to SQL Web Application Visual Basic
ASP.NET AJAX Server Control Visual Basic
ASP.NET AJAX Server Control Extender Visual Basic
ASP.NET Server Control Visual Basic
Browse...
OK
Cancel

Type: Visual Basic

A project for creating an application with a Web user interface

Name: WebApplication1

Location: D:\My Documents\Visual Studio 2010\Projects\

Solution: Create new solution

Solution name: WebApplication1

☑ Create directory for solution

☐ Add to source control

Figure 5.1: New Project Dialog Box to Create Web Application

Visual Studio creates a new project that includes prebuilt functionality based on the Web Forms template. It not only provides you with a *Default.aspx* page, an *About.aspx* page, but also includes membership functionality that registers users and saves their credentials so that they can log in to your website. When a new page is created, by default Visual Studio displays the page in Design view (Figure 5.2). Unlike a Windows form, a web page can have text added directly to it when it is in the Web Page Designer. In Design mode, developer can add any controls like design in windows application such as, label, text box and button to appear at the web page. Program code also can be written for the controls (Figure 5.3) as windows application.

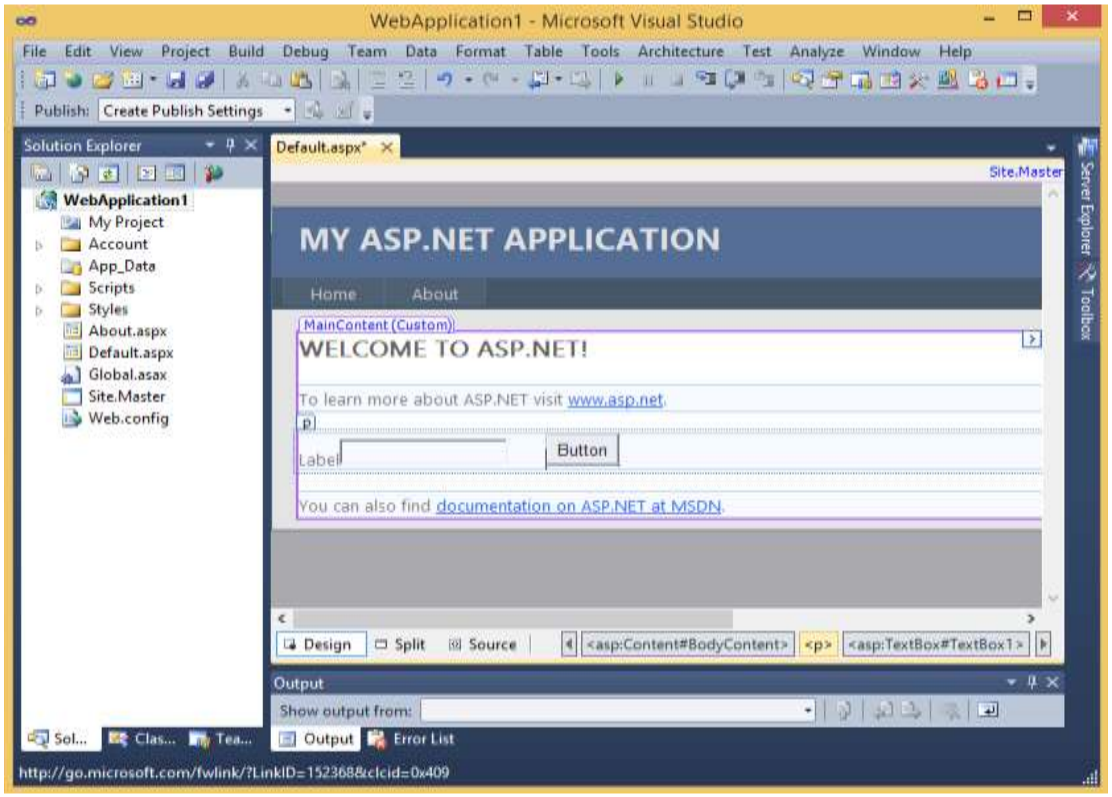
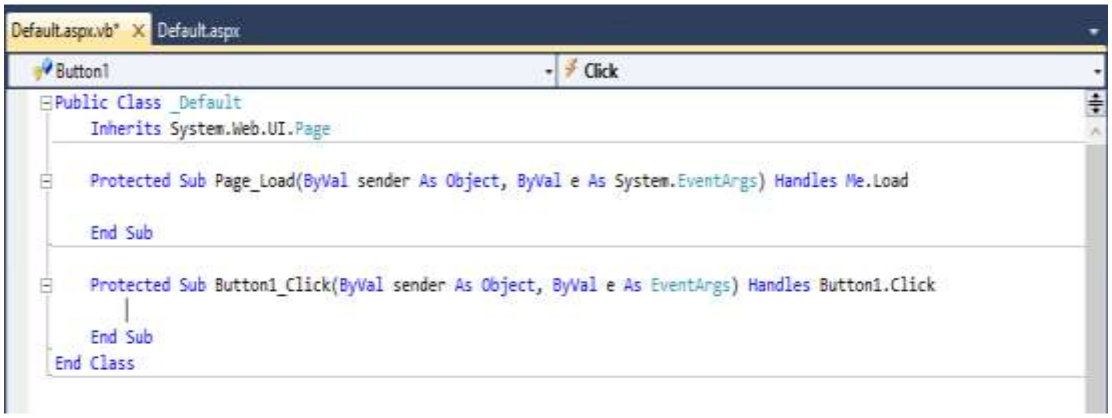

WebApplication1 - Microsoft Visual Studio
File Edit View Project Build Debug Team Data Format Table Tools Architecture Test Analyze Window Help
Publish: Create Publish Settings
Solution Explorer
WebApplication1
My Project
Account
App_Data
Scripts
Styles
About.aspx
Default.aspx
Global.asax
Site.Master
Web.config
Default.aspx*
Site.Master
MY ASP.NET APPLICATION
Home
About
MainContent (Custom)
WELCOME TO ASP.NET!
To learn more about ASP.NET visit www.asp.net.
p
Label
Button
You can also find documentation on ASP.NET at MSDN.
Design
Split
Source
<asp:Content#BodyContent> <p> <asp:TextBox#TextBox1>
Output
Show output from:
Sol... Clas... Tea...
Output Error List
Server Explorer
Toolbox
http://go.microsoft.com/fwlink/?LinkID=152368&clcid=0x409

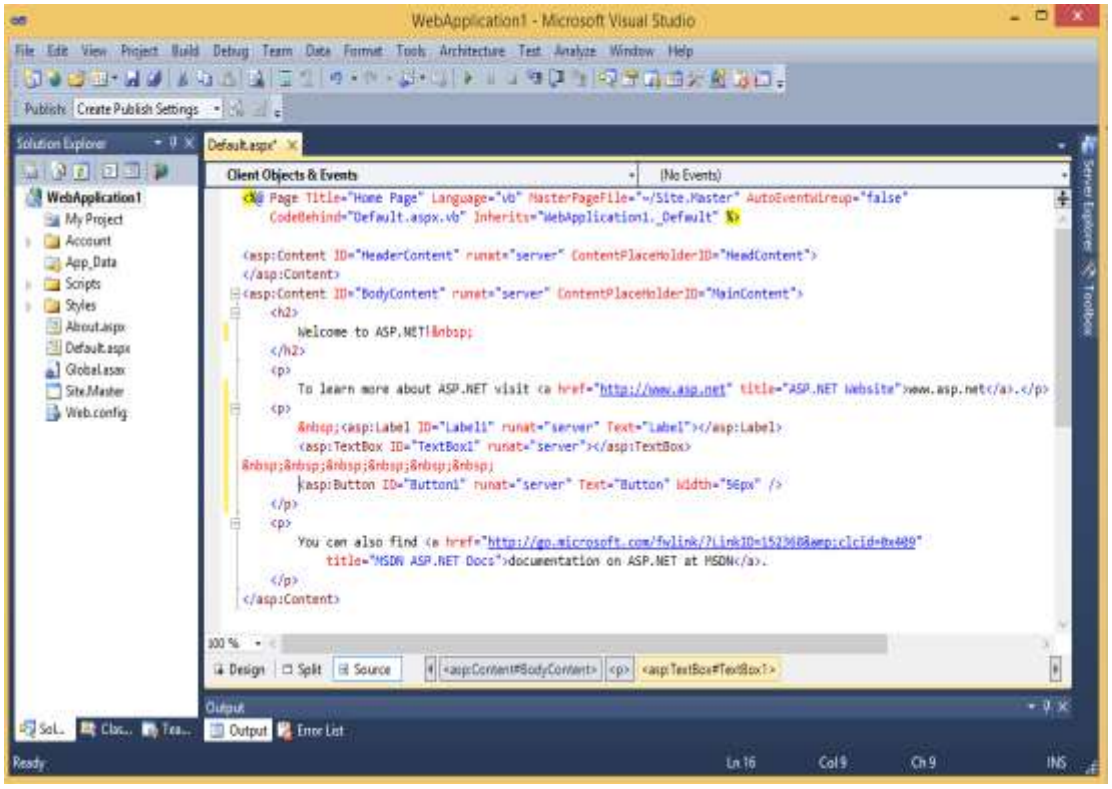
Figure 5.2: Design View of Web Page



```
Public Class _Default
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load

    End Sub

    Protected Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs) Handles Button1.Click

    End Sub
End Class
```

Figure 5.3: VB Code for Web Application

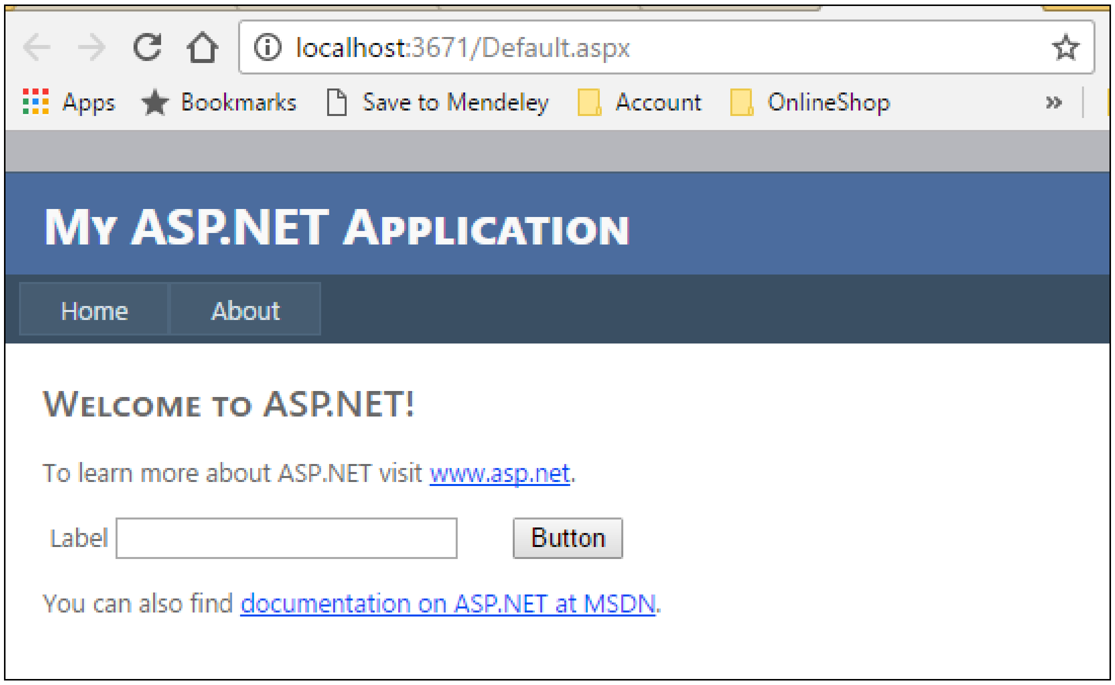
The HTML in **Source** view (**Figure 5.4**), where you can see the page's HTML elements. The illustration shows what you would see in **Source** view if you created a new Web page named *Default.aspx*.



```
<%@ Page Title="Home Page" Language="vb" MasterPageFile="~/Site.Master" AutoEventWireup="false"
    CodeBehind="Default.aspx.vb" Inherits="WebApplication1._Default" %>

<asp:Content ID="HeaderContent" runat="server" ContentPlaceHolderID="HeadContent">
</asp:Content>
<asp:Content ID="BodyContent" runat="server" ContentPlaceHolderID="MainContent">
    <h2>
        Welcome to ASP.NET!&nbsp;
    </h2>
    <p>
        To learn more about ASP.NET visit <a href="http://www.asp.net" title="ASP.NET Website">www.asp.net</a>.</p>
    <p>
        &nbsp;<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
        <asp:Button ID="Button1" runat="server" Text="Button" Width="56px" />
    </p>
    <p>
        You can also find <a href="http://go.microsoft.com/fwlink/?LinkID=152368&amp;clcid=0x409"
            title="MSDN ASP.NET Docs">documentation on ASP.NET at MSDN</a>.
    </p>
</asp:Content>
```

Figure 5.4: Source View of Web Page

To run the page, in **Solution Explorer**, right-click *Default.aspx* and select **Set as Start Page**. Then, press **CTRL+F5** to run the page. The page is displayed in the browser (Figure 5.5). Although the page you created has a file-name extension of *.aspx*, it currently runs like any HTML page. To display a page in the browser you can also right-click the page in **Solution Explorer** and select **View in Browser**. Close the browser to stop the Web application.



MY ASP.NET APPLICATION

WELCOME TO ASP.NET!

To learn more about ASP.NET visit www.asp.net.

You can also find documentation on ASP.NET at MSDN.

Figure 5.5: Page Is Displayed In the Browser

Activities

1. What is Web Form?
2. List Advantages of using Web forms?

References

1. Erik Reitan, 2014. **Creating a Basic ASP.NET 4.5 Web Forms Page in Visual Studio 2013**. Retrieved from https://www.asp.net/web-forms/overview/getting-started/creating-a-basic-web-forms-page

TOPIC 6

ACCESSING DATABASE USING ADO.NET

Learning Outcome

At the end of this topic, student can:

1. Describe database terminologies
2. Describe ADO.NET Object Model
3. Perform connection between VB application to Database
4. Manipulate Database records using insert, delete, edit and search operations.

Content

6.1 Introduction

Applications communicate with a database, firstly, to retrieve the data stored there and present it in a user-friendly way, and secondly, to update the database by inserting, modifying and deleting data.

The technology used to interact with a database or data source is called ADO.NET. The Microsoft ActiveX Data Objects.Net (ADO. NET) is a model, a part of the .Net framework that is used by the .Net applications for retrieving, accessing and updating data.

In ADO.NET the databases are connected first, then a copy of the database is stored in the memory immediately the connection to the database is disconnected. Database is connected only if any changes made to the copy of the database need to be updated to the database itself.

ADO.net classes are contained in the 'System.Date' namespace. Following are some of the important classes used in ADO.NET which are Connection, Command, DataAdapter, DataReader and DataSet.

6.2 Database Terminologies

A relational database is a collection of one or more tables of information that relate to each other in some way. Basic database terminologies describe in Table 6.1.

Table 6.1: Basic Database Terminologies

Terminology	Definition
Database	A special repository - consists of one or more physical files - used to store and retrieve data
Table	An object consists of rows and columns. Its looks like a spreadsheet
Row/ record	A record(details of a single entity
Column/field	An attribute or character of entity
Key Column (Field)	Uniquely identifies a row in a table eliminates the occurrence of duplicate rows.
Primary key	Column as key to ensure that each row in the table is unique
Composite key	Two or more columns to be joined together to make up the primary key for a table.
Foreign key	Used to relate two tables.

6.3 ADO.Net Object Model

The ADO.NET is about a software component provided by Microsoft in order to make easier for developers to make an access the data services from database. It was constructed based on class library by Microsoft .NET

Framework. Figure 6.1 has illustrated .NET Framework which is included the component of ADO.NET and it serve to the components in .NET Framework.

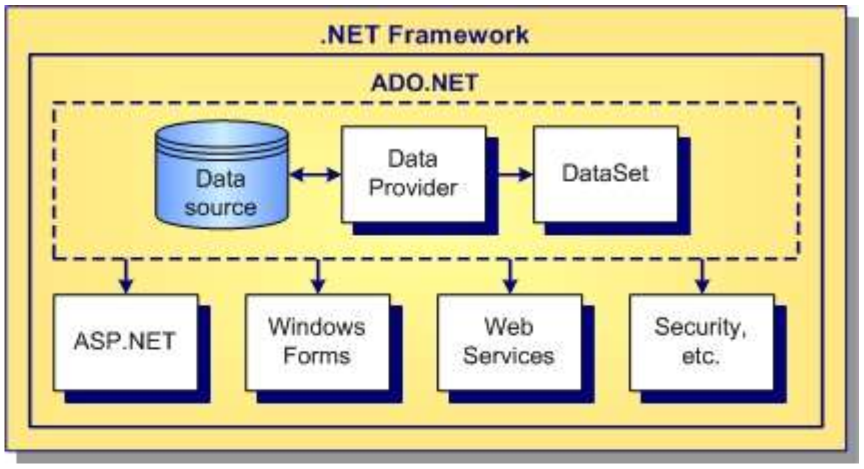


Figure 6.1: ADO.NET Architecture

The objective of ADO.NET is to bring a connection between objects in ASP.NET the back end database. ADO.NET offers an object-oriented view into the database, encapsulating many of the database properties and relationships within ADO.NET objects. Furthermore, and in many ways most important, the ADO.NET objects encapsulate and hide the details of database access; your objects can interact with ADO.NET objects without knowing or worrying about the details of how the data is moved to and from the database.

ADO.Net object model is nothing but the structured process flow through various components. The object model can be pictorially described as Figure 6.2. The data residing in a data store or database is retrieved through the **data provider**. Various components of the data provider retrieve data for the application and update data. An application accesses data either through a dataset or a data reader. Table 6.1 show descriptions for each object model.

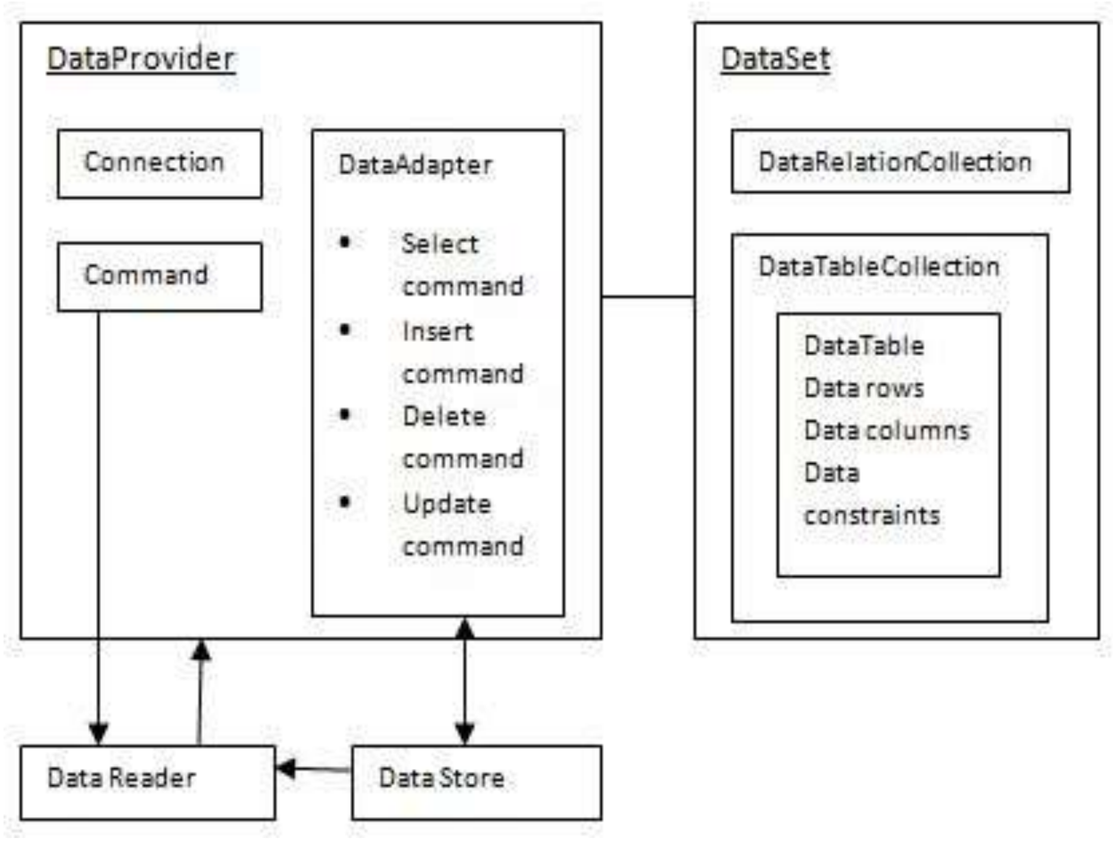


Figure 6.2: ADO.NET Object Model

Table 6.1: ADO.NET Object Model

ADO.NET Objects Model	**Description**	**Created at**
DataAdapter (brains behind the *DataSet)* (For example: *SQLDataAdapter* and *OleDbDataAdapter)*	This is integral to the working of ADO.NET since data is transferred to and from a database through a data adapter. It retrieves data from a database into a dataset and updates the database. When changes are made to the dataset, the changes in the database are actually done by the data adapter. It is used to translate your requests into the language the database uses.	Design time or at run time
Command object (For example: SQLCommand and OleDbCommand)	It is a SQL statement or a stored procedure used to retrieve, insert, delete or modify data in a data source.	Design time or at run time

DataReader	It uses a *Command* object to retrieve data one row at a time. It is very efficient because there is only one buffered row at a time in memory. But, cannot be used to update data in a database because it is read-only.	Run time.
DataSet	It is a memory-resident representation of the data that is passed to it by the *DataAdapter*. It can represent a complete set of data, including tables, constraints, and relationships between the tables.	Design time (*typed DataSet*) or at run time (*untyped DataSet*)
The Connection Object (SQLConnection, or OleDb-Connection)	Allows you to connect to a database by setting a connection string that specifies the name of the server (Data Source) and the database (Initial Catalog).	Design time or at run time
DataAdapter Object (SQLDataAdapter, or OleDbDataAdapter,)	Used to pass data to and from the database.It has the ability to retrieve, update, insert, and delete data in the database through the use of *Command* objects. *Fill* method of the *DataAdapter* is used to populate the dataset with the selected data. *Update* method of the *DataAdapter* is used to permanently update the database with the changes made in the dataset	Design time or at run time

6.3.1 Database Provider

A data provider is used for connecting to a database, executing commands and retrieving data, storing it in a dataset, reading the retrieved data and updating the database. There are basic **types of data providers** included in ADO.NET:

i. SQL Server - provides access to Microsoft SQL Server or others.

ii. for OLE DB - provides access to data sources exposed by using OLE DB.

6.3.2 Dataset

DataSet is an **in-memory representation of data.** It is a disconnected, cached set of records that are retrieved from a database. When a connection is established with the database, the data adapter creates a dataset and stores data in it. After the data is retrieved and stored in a dataset, the connection with the database is closed. This is called the 'disconnected architecture'. The dataset works as a virtual database containing tables, rows, and columns.

6.4 Connecting to a Database

VB.Net allows you many ways to connect to a database or a data source. The .Net Framework provides two types of Connection classes:

i. **SqlConnection** - designed for connecting to Microsoft SQL Server.
ii. **OleDbConnection** - designed for connecting to a wide range of databases, like Microsoft Access and Oracle.

To create VB application connect with database can be set using connection object.

6.4.1 Use Connection Object

ADO.NET offers a number of connection objects such as **OleDbConnection, SqlConnection** and more. OleDbConnection is used to access OLEDB data such as Microsoft Access while SqlConnection is used to access data provided by Microsoft SQL server. To initialize a new Connection object and to establish a connection to the data source using the ConnectionString property consist:

i. Declaring of a variable which hold the Connection Object

ii. Calling Data Provider (technology used to do the connecting)

iii. Specifying Data Source (database file and location of database)

For example if the OLE DB objects of data provider use “Jet”. Then, the name of the Access file we want to connect to is called AddressBook.mdb. (Note that "Data Source" is two words, and not one.) and its the database is on the C drive, in the root folder.

```
Dim con As New OleDb.OleDbConnection

con.ConnectionString = _
"PROVIDER=Microsoft.Jet.OLEDB.4.0;"dbProvider &_
"Data Source = C:/AddressBook.mdb"
```

Another example if using a SQL Server database. If the SQL server name is `PC123` as well as the path to database file is `C:\Program Files\Microsoft SQL Server\ MSSQL\Data\Test.mdf` and the program code as below:

```
Dim MyConnection As New SqlConnection

MyConnection.ConnectionString = "Data Source=PC123;_
AttachDbFilename=C:\Program Files\Microsoft SQLServer\_
MSSQL\Data\Test.mdf;” &_User Instance=True;Integrated
Security= SSPI”
```

To use ADO.NET classes and to access Microsoft Acccess databases, write `System.Data.OleDb` namespace for using `OleDb Connection`, `OleDb Data Adapter, OleDb Command` and `OleDb Parameter.`

While, to access SQL Server databases, write `System.Data.SqlClient` namespace for using `Sql Connection, SQL Data Adapter, SqlCommand` and `SqlParameter.`

After set the connection and when you are returned to your form, you should notice your new Data Source has been added as Figure 6.3.

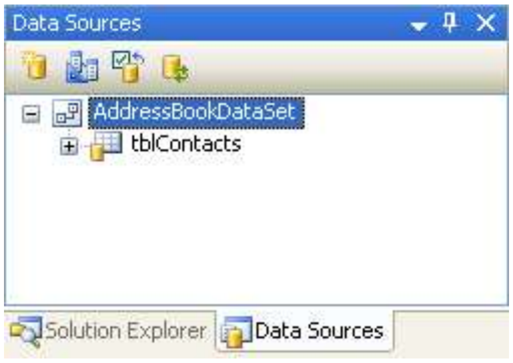


Figure 6.3: New Data Source

The Data Sources area of the Solution Explorer (or Data Sources tab on the left) now displays information about your database. Click the plus symbol (arrow symbol in version 2012/13) next to **tblContacts**. All the Fields in the Address Book database are now showing as Figure 6.4.

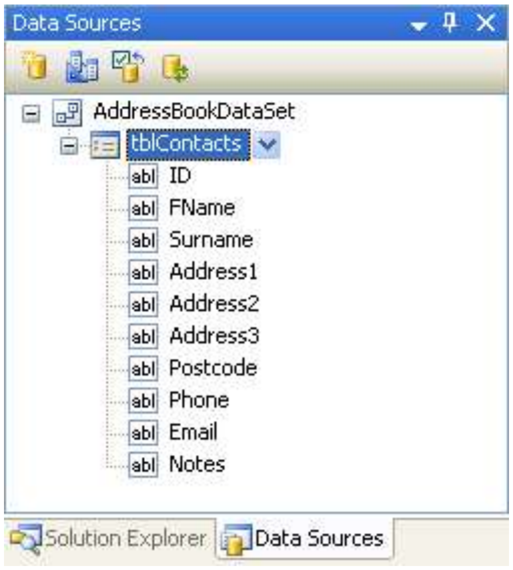
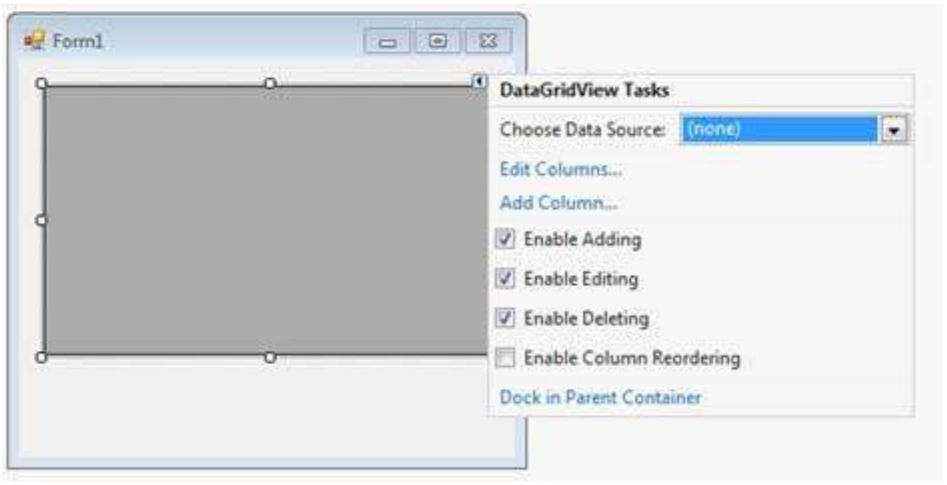
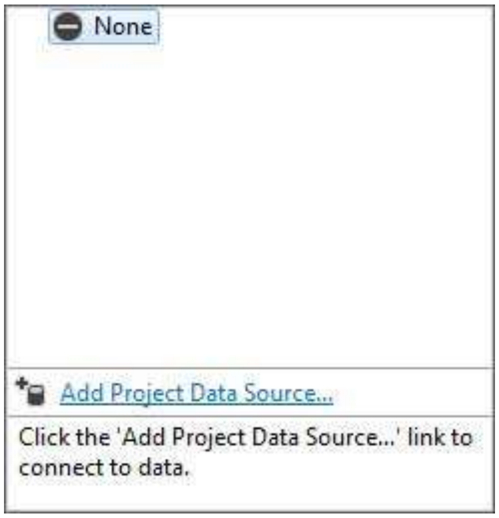


Figure 6.4: All the Fields in the Address Book database

6.5 View Data in DataGridView Control

To view data in DataGridView Control, add a DataGridView on the form. Then, click on the Choose Data Source combo box and click on the Add Project Data Source link as Figure 6.5.

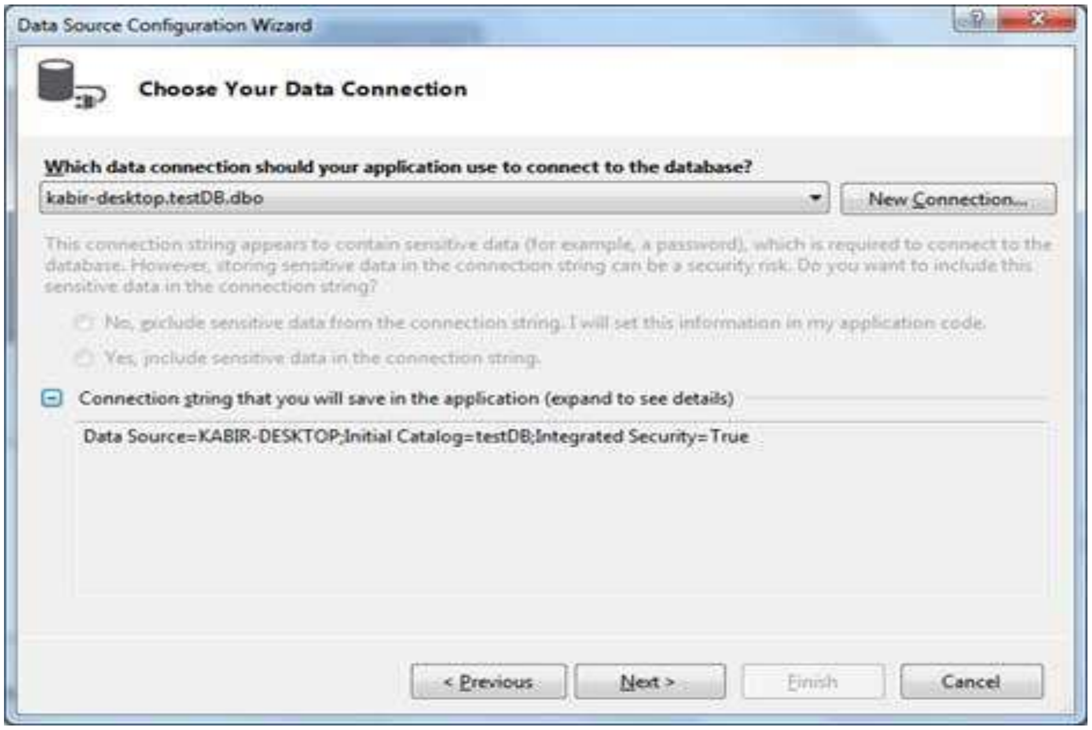




Click the 'Add Project Data Source...' link to connect to data.

Figure 6.5: Create Data Grid View onto a Form

Select Database (or Dataset) and click Next. You'll then see a screen with a Dataset item. Select this and click Next. Then, a Choose Your Data Connection screen will appear. Save the connection string.



Choose Your Data Connection

Which data connection should your application use to connect to the database?

kabir-desktop.testDB.dbo

This connection string appears to contain sensitive data (for example, a password), which is required to connect to the database. However, storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?

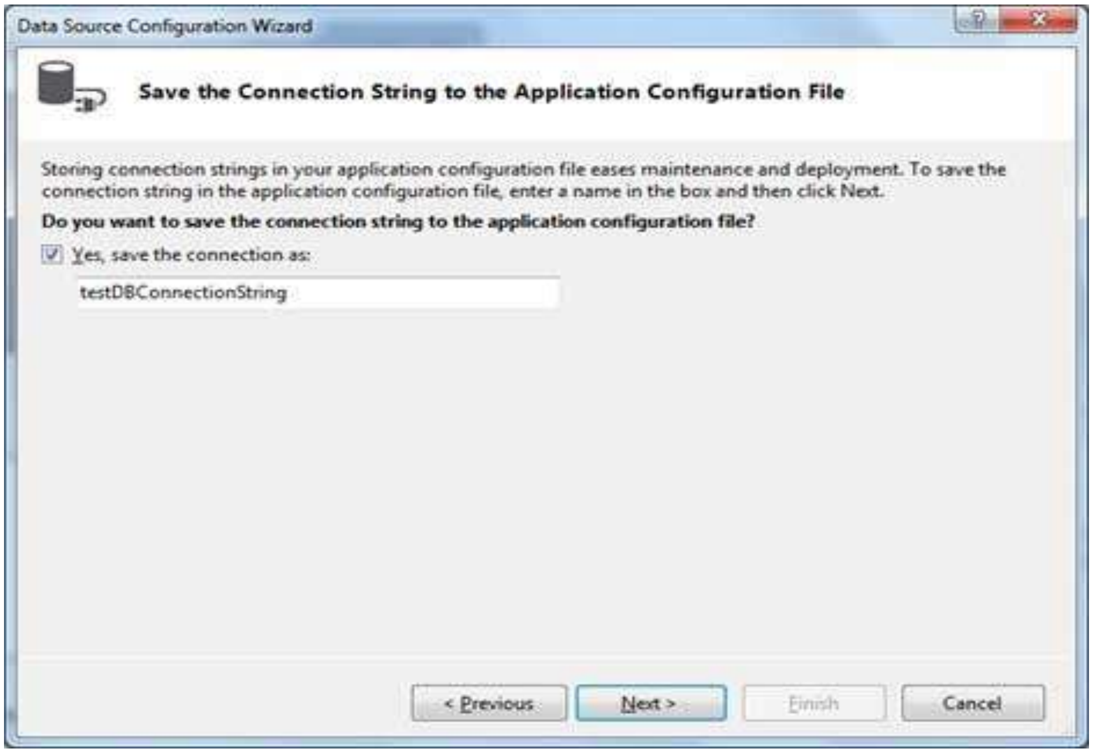
No, exclude sensitive data from the connection string. I will set this information in my application code.

Yes, include sensitive data in the connection string.

Connection string that you will save in the application (expand to see details)

```
Data Source=KABIR-DESKTOP;Initial Catalog=testDB;Integrated Security=True
```

Figure 6.6: Choose Database in order to Create a Connection



Save the Connection String to the Application Configuration File

Storing connection strings in your application configuration file eases maintenance and deployment. To save the connection string in the application configuration file, enter a name in the box and then click Next.

Do you want to save the connection string to the application configuration file?

☑ Yes, save the connection as:

testDBConnectionString

Figure 6.7: Set Name of Connection String

Choose the database object, Customers table in our example, and click the Finish button.

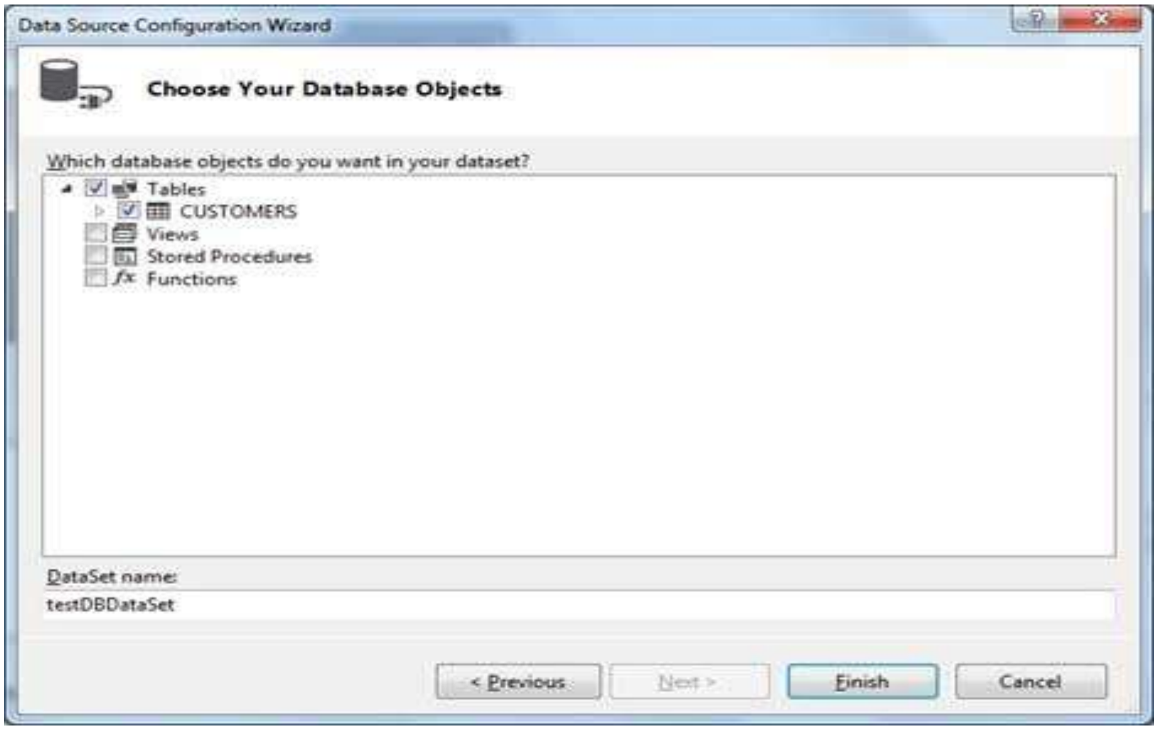
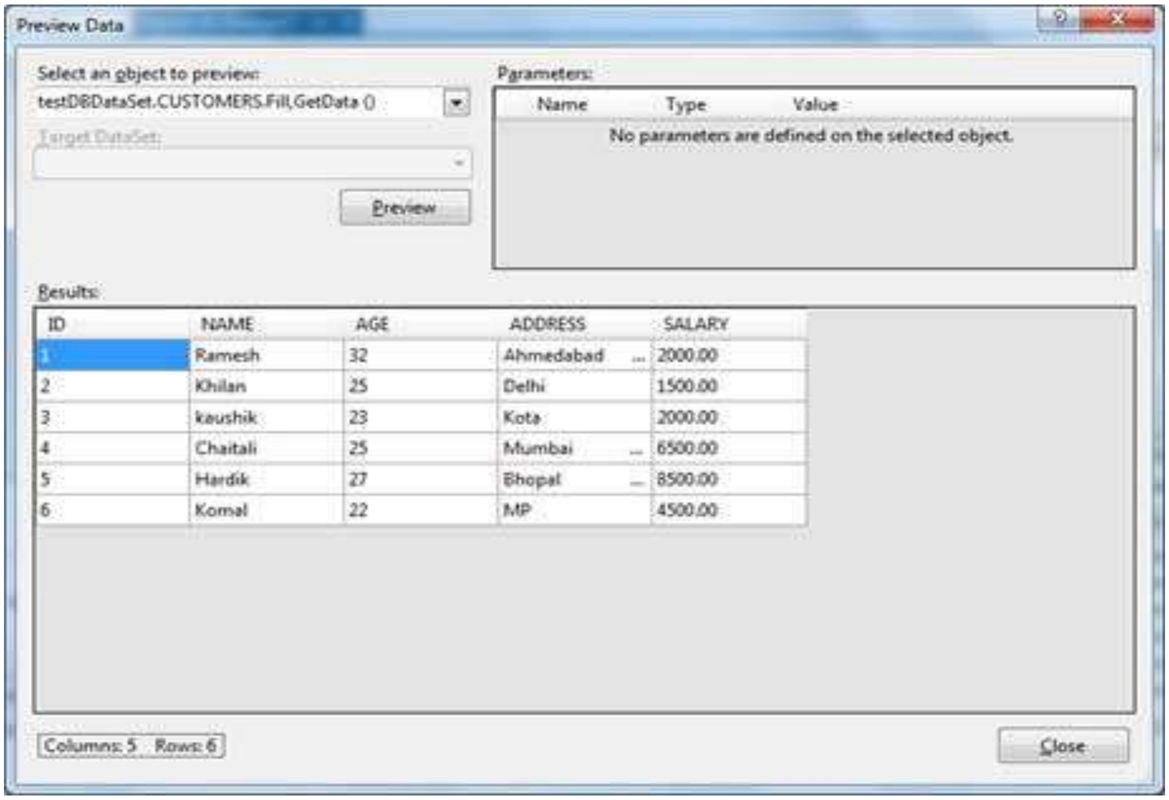

Data Source Configuration Wizard
Choose Your Database Objects
Which database objects do you want in your dataset?
Tables
CUSTOMERS
Views
Stored Procedures
Functions
DataSet name:
testDBDataSet
< Previous
Next >
Finish
Cancel

Figure 6.8: Choose Database Object

Select the Preview Data link to see the data in the Results grid:



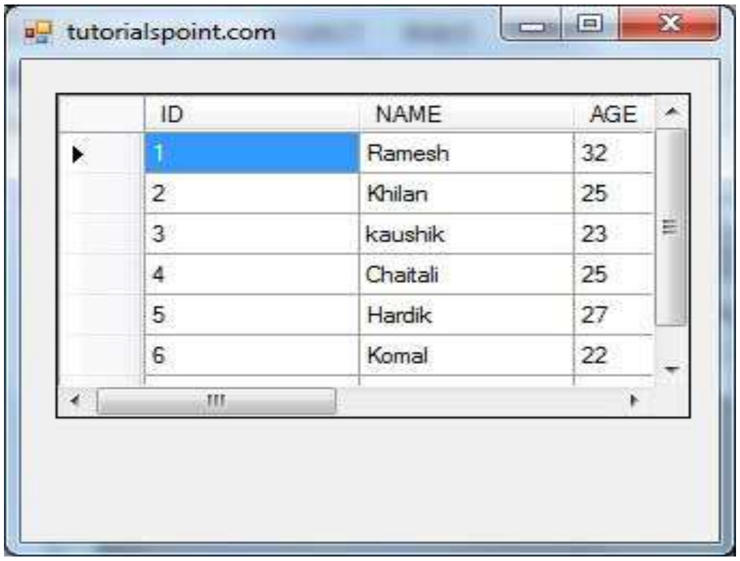
testDBDataSet.CUSTOMERS.Fill,GetData ()

No parameters are defined on the selected object.

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad ...	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai ...	6500.00
5	Hardik	27	Bhopal ...	8500.00
6	Komal	22	MP	4500.00

Figure 6.9: Preview Data link to see the Data in the Results Grid

When the application is run using **Start** button available at the Microsoft Visual Studio tool bar, it will show the window as Figure 6.10.



ID	NAME	AGE
1	Ramesh	32
2	Khilan	25
3	kaushik	23
4	Chaitali	25
5	Hardik	27
6	Komal	22

Figure 6.10: Data from Database Appear in the Form

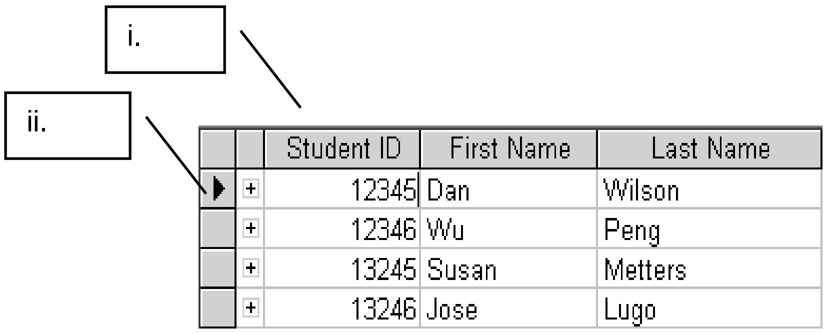
6.5 Manipulate Database Records

SQL (Structured Query Language) is used to retrieve and update data in a database table. Although SQL provides a number of powerful statements for accessing and manipulating data, we will be using the four basic statements, which are SELECT, UPDATE, INSERT, and *DELETE.* When setting up the DataAdapter, you created a SELECT statement. From this statement, the Data Adapter Configuration Wizard was able to create the other statements.

As conclusion, programmer layer – build commands (SELECT, INSERT, DELETE, and UPDATE) for execution against the database. Logical layer – uses a DataAdapter to send commands from your software (across the data adapter) to the database getting back tables to be processed. Physical layer – manages the actual (physical) execution of the commands on the database.

Activities

1. Identify basic database terminology in the table below:



Student ID	First Name	Last Name
12345	Dan	Wilson
12346	Wu	Peng
13245	Susan	Metters
13246	Jose	Lugo

2. What are the advantages of storing information in a relational database?
3. What is the purpose of defining a primary key for a table?

4. Describe a one-to-many relationship between two tables.
5. How do the connected and disconnected data access models differ?
6. List the ADO.NET Data Provider objects.
7. How is binding used in Visual Basic .NET applications?
8. What objects are used to move through the rows of the *DataTable* object and keep the data in the bound controls synchronized?

References

1. Erik Reitan, 2014. **Creating a Basic ASP.NET 4.5 Web Forms Page in Visual Studio 2013**. Available at https://www.asp.net/web-forms/overview/getting-started/creating-a-basic-web-forms-page

TOPIC 7

EFFECTIVE AND SECURE CODING

Learning Outcome

At the end of this topic, student can:

1. List types of input validation
2. Apply input validation in windows application
3. Apply web page validation for web application
4. Describe concept and purpose if exception handling
5. List types of exception and apply it in application development

Content

7.1 Input Validation

Validation is an **easy way of requiring input in a specific text box** without much hassle. Advantages for input validation will:

i. Prevent user from submitting blank values.

ii. Prevent user from circumventing your validation by pasting text

7.1.1 `TryParse` Method for Input Validation

`TryParse` Method used to set the textbox to only allow or accept specific data type (e.g: numbers) to be entered. **`TryParse`** parse a string into a value for a specific type {T} but not does that but returns a Boolean (True/False) value to indicate that parsing produce a valid value. Note that it doesn't return the value as the function return type, but passes it back via the referenced variable supplied as the argument for the result parameter. The basic pattern for .**`TryParse`** is a following, where {T} is replaced by the type as template code at Figure 7.1.

```
Dim Value As {T} = Nothing
If {T}.TryParse( Text , Value ) Then
   ' Is a valid parse and thus a valid value.
Else
  ' Not Valid
End If
```

Figure 7.1: Template code for `.TryParse` Method

A lot of the type that parses a string input into a value provides a .**`TryParse`** method such as: `Integer, Double, Int32, Date, DateTime.`

Example:

First validate if the input is actually an integer with `Integer.TryParse`:

```
Dim intValue As Integer
If Not Integer.TryParse(TxtBox.Text, intValue)_
  OrElse intValue < 1 OrElse intValue > 10 Then
```

```
    MessageBox.Show("Please Enter a Number from 1 to 10")
Else
    MessageBox.Show("Thank You, your rating was " & TxtBox.Text)
End If
```

Figure 7.2: An Example for Input Validate with `Integer.TryParse` Method

7.1.2 Use the `TextBox.TextChanged` Event

Use the TextBox.TextChanged event to catch it whether they type or paste the numbers in.

Example:

Allow the use of 1 decimal point in the number but, if only want whole numbers then you can change it a little to get that result (See Figure 7.3).

```
Public Class frmTestInput
    Dim tt As New ToolTip With {.IsBalloon = True}

    Private Sub txtNumber_TextChanged(ByVal sender As Object, _
     ByVal e As System.EventArgs) Handles txtNumber.TextChanged
        Dim dp As Integer = 0
        For Each ch As Char In TextBox1.Text
          If ch = "." Then dp += 1
            If (Not Char.IsDigit(ch) And Not ch = ".")Or dp > 1 Then
                TextBox1.Clear()
                tt.Show("Please Enter Valid Numbers Only",_
                 TextBox1, New Point(0, -40), 4000)
            End If
        Next
    End Sub
End Class
```

Figure 7.3: An Example Input Validation using `TextBox.TextChanged` Method

7.2 Exceptions Handling

Runtime errors are a lot harder than Design Time errors to track down. As their name suggests, these errors occur when the programme is running. Runtime errors are the ones that crash your programme. VB.NET has a inbuilt class that deals with errors. The Class is called **Exception**. When an exception error is found, an **Exception object** is created.

When you allow users to input numbers and use those numbers in calculations, lots of things can go wrong. The conversion functions, `CInt` and `CDec`, fail if the user enters nonnumeric data or leaves the text box blank. Or your user may enter a number that results in an attempt to divide by zero. Each of those situations causes an **exception** to occur, or as programmers like to say, **throws an exception**. You can easily **catch** program exceptions by using VB .NET's new structured exception handling. You catch the exceptions before they can cause a run-time error and handle the situation, if possible, within the program. **Catching exceptions** as they happen is generally referred to as **error trapping**, and **coding to take care** of the problems is called **error handling**. The error handling in Visual Studio .NET is standardized for all of the languages using the Common Language Runtime, which greatly improves on the old error trapping in previous versions of VB. As conclusion**, the purpose of exception handling is** to isolate specific code statements or blocks of code and catch any exception that is thrown as a result of an error.

7.2.1 Exception Keywords

VB.Net exception handling is built upon 4 keywords: Try, Catch, Finally and Throw (Refer Table 7.1).

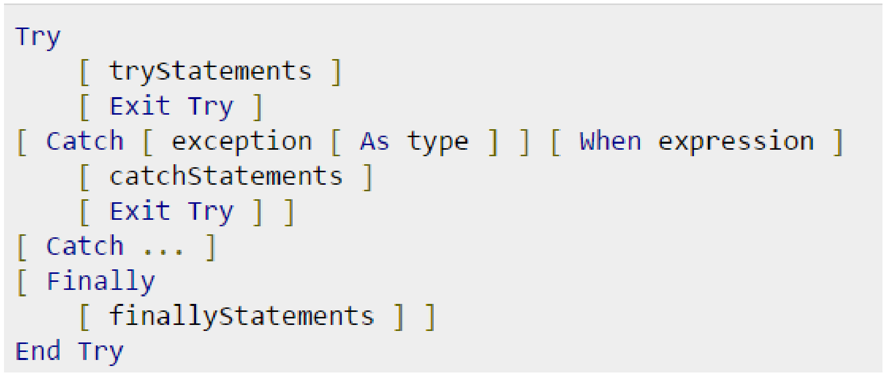
Table 7.1: Keywords Used For Handling Exception

Keywords	Description
`Try`	A Try block identifies a block of code for which particular exceptions will be activated. It's followed by one or more Catch blocks.
`Catch`	A program catches an exception with an exception handler at the place

	in a program where you want to handle the problem. The Catch keyword indicates the catching of an exception.
`Finally`	The Finally block is used to execute a given set of statements, whether an exception is thrown or not thrown. For example, if you open a file, it must be closed whether an exception is raised or not.
`Throw`	A program throws an exception when a problem shows up. This is done using a Throw keyword.

To apply exception handling, assuming a block will raise an exception, a method catches an exception using a combination of the Try and Catch keywords. Then, a Try/Catch block is placed around the code that might generate an exception.

Syntax:



```
Try
    [ tryStatements ]
    [ Exit Try ]
[ Catch [ exception [ As type ] ] [ When expression ]
    [ catchStatements ]
    [ Exit Try ] ]
[ Catch ... ]
[ Finally
    [ finallyStatements ] ]
End Try
```

To trap or catch exceptions, enclose any statement(s) that might cause an error in a `Try/Catch` block. If an exception occurs while the statements in the `Try` block are executing, then program control transfers to the `Catch` block; if a `Finally` statement is included, the code in that section executes last, whether or not an exception occurred.

Example:

```
Try
```

```
          intQuantity = CInt(txtQuantity.Text)
          lblQuantity.Text = CStr(intQuantity)
     Catch
          lblMessage.Text = "Error in input data."
     End Try
```

The `Catch` as it appears in the preceding example will catch any exception. You can also specify the type of exception that you want to catch, and even write several `Catch` statements, each to catch a different type of exception. For example, you might want to display one message for bad input data and a different message for a calculation problem. To specify a particular type of exception to catch, you can use one of the predefined exception classes, which are all based on, or derived from, the `SystemException` class.

To catch bad input data that cannot be converted to numeric, write this `Catch` statement:

```
     Catch MyErr As InvalidCastException
          lblMessage.Text = "Error in input data."
```

7.2.2 Types of Exceptions

In the .Net Framework, exceptions are represented by classes. Table 7.2 shows some of the common exception classes. Each exception is an instance of the Exception class. The properties of this class allow you to determine the code location of the error, the type of error, and the cause. The Message property contains a text message about the error, and the Source property contains the name of the object causing the error. The `StackTrace` property can identify the location in the code where the error occurred. You can include the text message associated with the type of exception by specifying the Message property of the Exception object, as declared by the variable you named on the `Catch` statement.

Table 7.2: Common Exception Classes

Exception Class	Descriptions

`FormatException`	Failure of a numeric conversion, such as `Integer.Parse` or `Decimal.Parse`. Usually blank or nonnumeric data.
`System.IO.IOException(e.g: System.IO.FileNotFoundException)`	Handles Input Output Errors (for example: Try to open file that not exist)
`System.IndexOutOfRangeException`	Handles errors generated when a method refers to an array index out of range.
`System.ArrayTypeMismatchException`	Handles errors generated when type is mismatched with the array type
`System.ArithmeticException(e.g: System.DivideByZeroException, System.OverflowException)`	Handles errors generated from arithmetic calculation, such as division by zero or overflow of a variable.
`System.InvalidCastException`	Handles errors generated during typecasting. For example: Failure of a conversion function, such as `CInt` or `CDec`. Usually blank or nonnumeric data.
`System.OutOfMemoryException`	Handles errors generated from insufficient free memory.E.g: Not enough memory to create an object.
`Exception`	Generic exception

Be aware that the messages for exceptions are usually somewhat terse and not oriented to users, but they can sometimes be helpful.

For example:

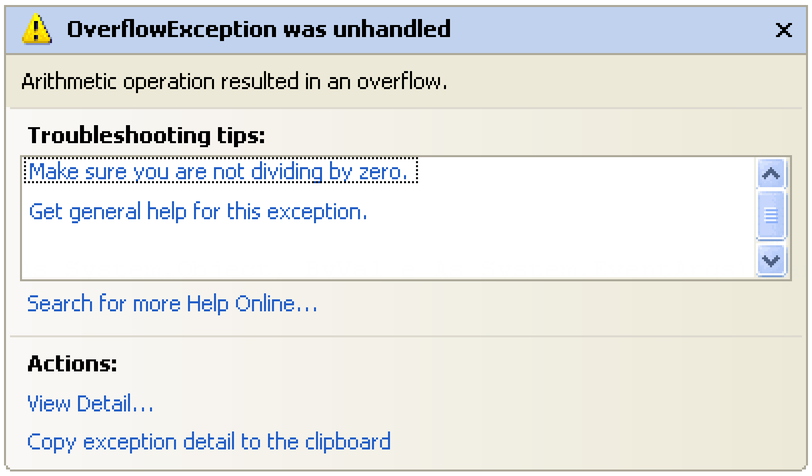
```
Catch MyErr As InvalidCastException
lblMessage.Text = "Error in input data: " & MyErr.Message
```

7.2.3.1 Arithmetic Exception

A simple way to crash a programme is divided a number by zero. Try this code for a button:

```
Dim Num1 As Integer
Dim Num2 As Integer
Num1 = 10
Num2 = 0
txtDivide.Text = CInt(Num1 / Num2)
```

But run the programme and test it out. Click the button and the error message will popping up in Visual Studio Express 2010 (Figure 7.4):


OverflowException was unhandled

Arithmetic operation resulted in an overflow.

Troubleshooting tips:

Make sure you are not dividing by zero.

Get general help for this exception.

Search for more Help Online...

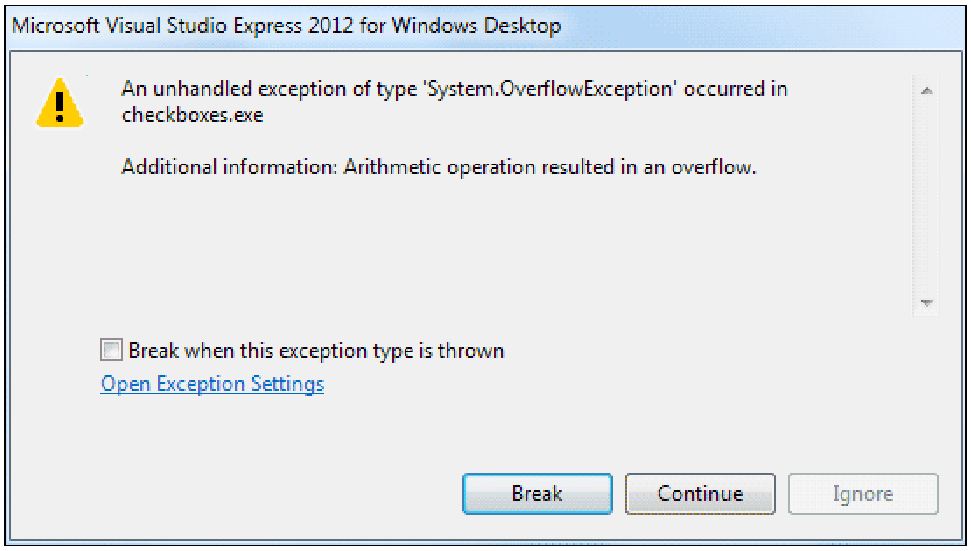
Actions:

View Detail...

Copy exception detail to the clipboard

Figure 7.4: OverflowException Message Box (VS Express 2010)

In version 2012/13, the error message (the 2010 version seems more helpful, and better designed) will appear in Figure 7.5:


Microsoft Visual Studio Express 2012 for Windows Desktop

An unhandled exception of type 'System.OverflowException' occurred in checkboxes.exe

Additional information: Arithmetic operation resulted in an overflow.

Break when this exception type is thrown

Open Exception Settings

Figure 7.5: OverflowException Message Box

Click the Break button, and then stop the programme from running.

When trying to divide by zero, VB.NET throws up the **Overflow** error message - there would be just too many zeros to go into the **Integer** variable type. Even if you change the Type into a **Single** or a **Double**, you'd still get the same error message.

7.2.3.2 Input Output Exception

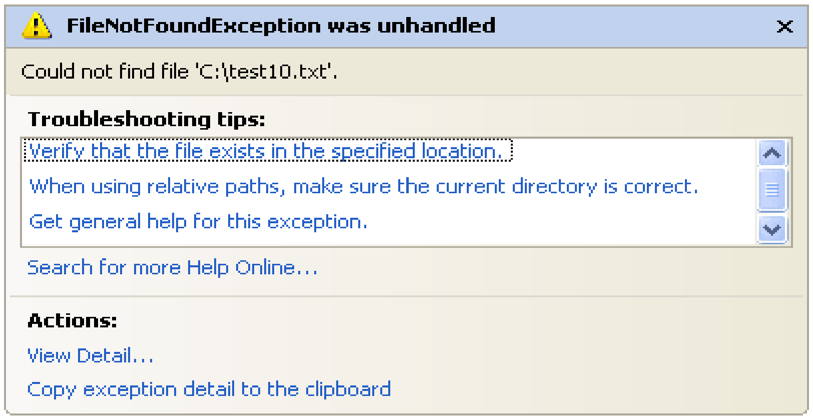
An example how input output exception will happen. From the controls toolbox, add a RichTextBox control to a form. Change the **Name** property of your**RichTextBox** to **rt1**. A RichTextBox is just like a normal textbox but with more functionality. Delete or comment out any code for a button, and add the following line:

```
rt1.LoadFile("C:\test10.txt",
RichTextBoxStreamType.PlainText)
```

However, for Windows 7 then change the file name above to this:

```
"C:\Users\Owner\Documents\test10.txt"
```

All the line does is to load (or try to) the text file called `"test10.txt"` into the RichTextBox. The second argument just specifies that the type of file we want to load is a Plain Text file. Run the programme, and then click the button. If a text file called `"test10.txt"` is not exist in the root folder of your C drive, you'll get the following Runtime error message (refer Figure 7.6 for version 2010 of Visual Studio Express). In version 2012/13, the same information can be seen in the first line: `"An unhandled exception of type 'System.IO.FileNotFoundException' occurred in mscorlib.dll"`.


FileNotFoundException was unhandled

Could not find file 'C:\test10.txt'.

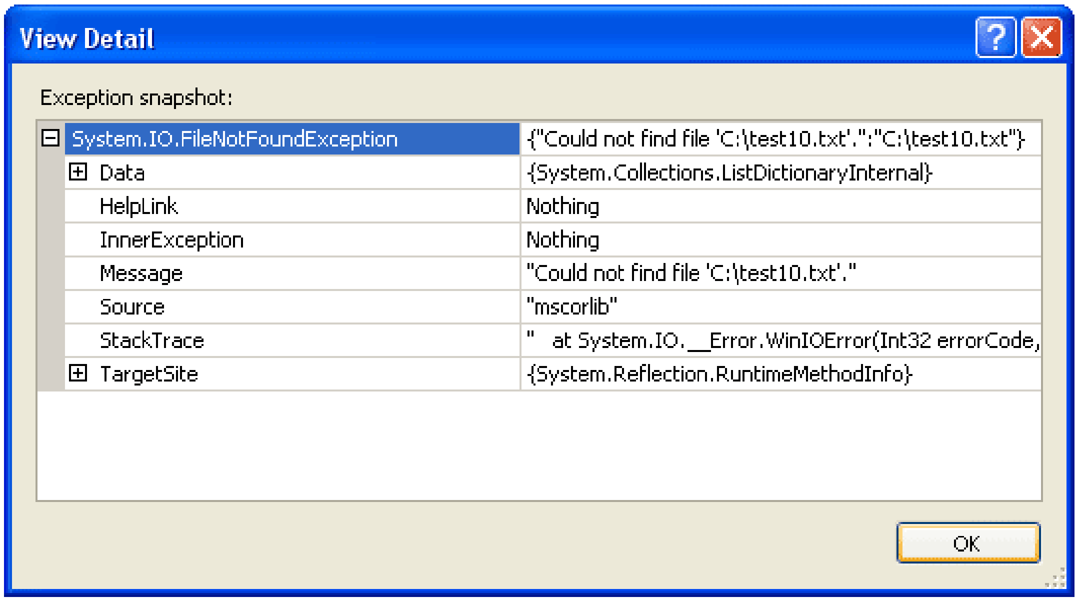
Troubleshooting tips:
Verify that the file exists in the specified location.
When using relative paths, make sure the current directory is correct.
Get general help for this exception.

Search for more Help Online...

Actions:
View Detail...
Copy exception detail to the clipboard

Figure 7.6: FileNotFoundException Message Box

Click the **View Details** links under **Actions** to see the Figure 7.7. The additional information is quite useful this time. It's saying that the file `"C:\test10.txt"` could not be found. If the error occurred in a normal programme, it would shut down.



Exception snapshot:

System.IO.FileNotFoundException	{"Could not find file 'C:\test10.txt'.":"C:\test10.txt"}
Data	{System.Collections.ListDictionaryInternal}
HelpLink	Nothing
InnerException	Nothing
Message	"Could not find file 'C:\test10.txt'."
Source	"mscorlib"
StackTrace	" at System.IO.__Error.WinIOError(Int32 errorCode,
TargetSite	{System.Reflection.RuntimeMethodInfo}

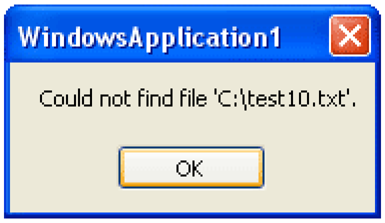
Figure 7.7: Details of Exception FileNotFoundException

The coding structure VB.NET uses to deal with such Exceptions is called the Try … Catch structure and should look like this:

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

    Try

        rt1.LoadFile("C:\test10.txt", RichTextBoxStreamType.PlainText)

    Catch ex As Exception

        MsgBox(ex.Message)

    End Try

End Sub
```

The `Try` word means "`Try to execute this code`". The `Catch` word means "`Catch any errors here`". The `ex` is a variable, and the type of variable it is an Exception object. Because `ex` is an object variable, it now has its own Properties and methods. One of these is the Message property.

When run this programme, VB will try to execute any code in the `Try` part. If everything goes well, then it skips the `Catch` part. However, if an error occurs, VB.NET jumps straight to `Catch`. Run the programme and test it out. Click your button. You should see the following error message:



Could not find file 'C:\test10.txt'.

Figure 7.8: Message Box

From Figure 7.7, the first line tells us the Type of Exception it is:

```
System.IO.FileNotFoundException
```

It can be added directly to the catch part. Previously, it just catching any error that might be thrown:

```
Catch ex As Exception
```

But if a "file not found" error might be thrown is known, add that to the Catch line, instead of Exception:

```
Catch ex As System.IO.FileNotFoundException
```

There is one last part of the `Try … Catch` Statement that VB.NET doesn't add - **`Finally`**:

```
Try
Catch ex As Exception
    Finally
End Try
```

The **`Finally`** part is always executed, whether an error occurs or not. You typically add a Finally part to perform any cleanup operations that are needed. For example, you may have opened a file before going into a **`Try`** … **`Catch`** Statement. If an error occurs, the file will still be open. Whether an error occurs or not, you still need to close the file. You can do that in the **`Finally`** part. But Microsoft advice that you always use **`Try`** … **`Catch`** Statements in your code.

7.2.3.3 Handling Multiple Exceptions

To trap for more than one type of exception, multiple `Catch` blocks (handlers) can be included. When an exception occurs, the `Catch` statements are checked in sequence. The first one with a matching exception type is used.

```
Catch MyErr As InvalidCastException
      'Statements for non numeric data
Catch MyErr As ArithmeticException
      'Statements for calculation problem
Catch MyErr As Exception
      'Statements for any other exception
```

The last `Catch` will handle any exceptions that do not match either of the first two exception types. Note that it is acceptable to use the same variable name for multiple `Catch` statements.

Activities

1. When should you use `Try/Catch` blocks? Why?
2. Give an example caused of each types of exception.

References

1. https://www.tutorialspoint.com/vb.net/vb.net_exception_handling.htm