Edited by Foxit PDF Editor
Copyright (c) by Foxit Software Company, 2004 - 2007
For Evaluation Only.

FUNDAMENTALS OF COMPUTING
LECTURE NOTES

PROGRAMMING WITH VISUAL BASIC 6.0

CHAPTER 5

"CONTROLLING FLOW AND DEBUGGING"

PREPARED BY

Edited by Foxit PDF Editor
Copyright (c) by Foxit Software Company, 2004 - 2007
For Evaluation Only.

CHAPTER 5: CONTROLLING FLOW AND DEBUGGING

This chapter describes writing conditional expressions that control the flow of program code, and tracking down software defects with Visual Basic's debugging tools. In this chapter, you will learn how to:

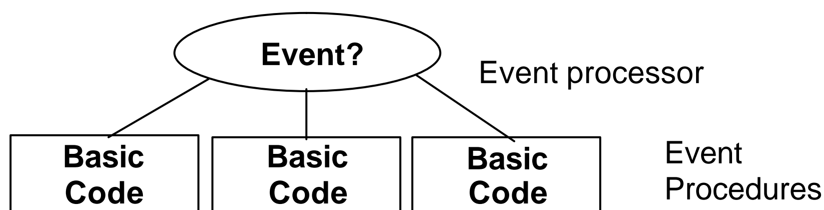
- Understand and use the principles of event-driven programming.
- Use conditional expressions, decision structures, and mathematical operators to control the order that your program executes commands.
- Find and correct errors in your programs.

5.1: Event Driven Programming

So far, the programs you have written displayed menus, objects, and dialog boxes on the screen and encouraged users to manipulate screen elements in whatever order they saw fit. These programs put the user in charge, waited patiently for a response, and then processed the input predictably.

In programming circles, this methodology is known as event-driven programming. You build a program by creating a group of intelligent objects (objects that know how to respond when the user interacts with them), and then you process the input by using event procedures associated with the objects.

Older, character-based versions of Basic such as QuickBasic or BASICA lacked visual development tools and executed code in sequence, from beginning to end. Event-driven programs start with graphical objects and then place the user in charge. This different approach requires a different development strategy. In the event-driven model, the programmer's job is to implement tasks that the user wants to accomplish.



5.2: Conditional Expressions

An event-driven program must be ready to respond to almost any operating condition. It's probably a safe bet, though, that only a few of the routines in a particular program are actually executed. For example, a word processing application may always be prepared to generate a sophisticated mailing list. However, it probably doesn't actually do so very often. The program code that accomplishes mail merge tasks sits ready, but unused, until the moment is right.

In Visual Basic programs, you can direct the course (flow) of your program by creating efficient interface objects and event procedures. However, you can also control which statements and in which order statements inside the event procedures run. In Visual Basic terminology, this process is called creating conditional expressions.

Note Expressions that can be evaluated as **True** or **False** are also known as **Boolean** expressions, in which the **True** or **False** result can be assigned to a Boolean variable or property. You can assign

Edited by Foxit PDF Editor
Copyright (c) by Foxit Software Company, 2004 - 2007
For Evaluation Only.

Boolean values to certain object properties, Variant variables, or Boolean variables that have been created by using the **Dim** statement and the **As Boolean** keywords.

5.2.1: Comparison Operators

One of the most useful tools for processing event procedure information is a conditional expression. A conditional expression is part of a complete program statement that asks a true-or-false question about:

- A property
- A variable
- another piece of data in the program code.

At the heart of every conditional expression is a comparison operator that creates a relationship between values. Here's a simple conditional expression:

```
Price < 100
```

If the Price variable contains a value that is less than 100, the expression evaluates to **True**. If Price contains a value that is greater than or equal to 100, it evaluates to **False**. You can use the comparison operators in this table to create conditional expressions:

Comparison Operator	Meaning
=	Equal to
< >	Not equal to
>	Greater than
<	Less than
> =	Greater than or equal to
< =	Less than or equal to

This table shows some conditional expressions that include some of the comparison operators. (You'll be working with these expressions later in this section.)

Conditional Expression	Result
10 < > 20	**True** (10 is not equal to 20)
Score < 20	**True** if Score is less than 20; otherwise, **False**
Score = Label1.Caption	**True** if the **Caption** property of the **Label1** object contains the same value as the Score variable; otherwise, **False**
Text1.Text = "Bill"	**True** if the word *Bill* is in the first text box; otherwise, **False**

Edited by Foxit PDF Editor
Copyright (c) by Foxit Software Company, 2004 - 2007
For Evaluation Only.

5.2.2: If – Then Decision Structures

When you use conditional expressions in a special block of statements known as a decision structure, you can control the order in which statements are executed. With an If...Then decision structure, your program can evaluate a condition in the program and take a course of action based on the result.

In Visual Basic, your If...Then decision structures can work with single or multiple conditional statements.

Single-condition Structures

In its simplest form, an If...Then decision structure is written on a single line:

```
If condition Then statement
```

where:

condition is a conditional expression.

statement is a valid Visual Basic program statement.

For example, this is an If...Then decision structure that uses a simple conditional expression:

```
If Score >= 20 Then Label1.Caption = "You win!"
```

The decision structure uses the conditional expression, Score >= 20, to determine whether the program should set the **Label1** caption to "You win!" If the Score variable contains a value greater than or equal to 20, Visual Basic sets the **Caption** property. Otherwise, it skips the assignment statement and executes the next line in the event procedure. This sort of comparison always results in a result of **True** or **False**. A conditional expression never results in "Maybe."

Multiple-condition Structures

Another Visual Basic If...Then decision structure supports several conditional expressions. This decision structure is a block of statements that can be several lines long — it contains these important keywords: **ElseIf**, **Else**, and **End If**.

```
If condition1 Then
    statements executed if condition1 is True
ElseIf condition2 Then
    statements executed if condition1 is False AND condition2 is True
[Additional ElseIf clauses and statements can be placed here]
Else
    statements executed if none of the conditions is True
End If
```

In the decision structure, Visual Basic evaluates condition1 first. If this conditional expression is **True**, the block of statements below it is executed one statement at a time. (You can include one or more program statements.) If the first condition is not **True**, Visual Basic evaluates the second conditional expression (condition2). If the second condition is **True**, the second block of statements is executed. (You can add additional **ElseIf** conditions and statements if you have more conditions to evaluate.) Finally, if none of the conditional expressions is **True**, Visual Basic evaluates the statements below the **Else** keyword. The whole structure is closed at the bottom with the **End If** keywords.

This code sample below shows how you could use a multiline If...Then structure to determine the amount of tax due on a hypothetical progressive tax return. In this decision structure, Visual Basic performs these steps:

Edited by Foxit PDF Editor
Copyright (c) by Foxit Software Company, 2004 - 2007
For Evaluation Only.

```
If AdjustedIncome <= 24650 Then         '15% tax bracket
    TaxDue = AdjustedIncome * 0.15
ElseIf AdjustedIncome <= 59750 Then     '28% tax bracket
    TaxDue = 3697 + ((AdjustedIncome - 24650) * 0.28)
ElseIf AdjustedIncome <= 124650 Then    '31% tax bracket
    TaxDue = 13525 + ((AdjustedIncome - 59750) * 0.31)
ElseIf AdjustedIncome <= 271050 Then    '36% tax bracket
    TaxDue = 33644 + ((AdjustedIncome - 124650) * 0.36)
Else                                    '39.6% tax bracket
    TaxDue = 86348 + ((AdjustedIncome - 271050) * 0.396)
End If
```

- Tests the variable AdjustedIncome at the first income level.
- Continues to test subsequent income levels until one of the conditional expressions evaluates to **True**.
- Determines an income tax for the taxpayer.

This simple decision structure is quite useful. It could be used to compute the tax owed by any taxpayer in a progressive tax system, such as the one in the United States. (We're assuming that the tax rates are complete and up to date and that the value in the AdjustedIncome variable is correct.) If the tax rates change, it's a simple matter to update the conditional expressions.

Important! The order of the conditional expressions in your If...Then and ElseIf clauses is critical. For example, reverse the order of the conditional expressions in the tax computation example. If you listed rates in the structure from highest to lowest, this is what happens:

- Taxpayers in the 15 percent, 28 percent, and 31 percent tax brackets would be placed in the 36 percent tax bracket. (That's because they all would have an income that is less than or equal to 250,000.)
- Visual Basic would stop at the first conditional expression that is **True**, even if the others are also **True**.

All of the conditional expressions in this example test the same variable, so they need to be listed in ascending order to get the taxpayers to fall out at the right spots. Moral: When you use more than one conditional expression, watch their order carefully.

Logical Operators

In Visual Basic, if you include more than one selection criterion in your decision structure, you can test more than one conditional expression in your If...Then and ElseIf decision structures. Visual Basic links the extra conditions by using one or more of the operators shown in this table:

Logical Operator	Meaning
And	If both conditional expressions are True, then the result is True.
Or	If either conditional expression is True, then the result is True.
Not	If the conditional expression is False, then the result is True. If the conditional expression is True, then the result is False.
Xor	If one and only one of the conditional expressions is True, then the result is True. If both are True or both are False, then the result is False.

Edited by Foxit PDF Editor
Copyright (c) by Foxit Software Company, 2004 - 2007
For Evaluation Only.

Note When expressions contain mixed operator types, your program evaluates operators in this order:

- Mathematical operators.
- Comparison operators.
- Logical operators.

Logical Operators at Work

This table lists some examples of logical operators at work. In the expressions, it is assumed that the variable Vehicle contains a value of "Bike" and that the variable Price contains a value of 200.

Logical expression	Result
Vehicle = "Bike" And Price < 300	**True** (both expressions are **True**)
Vehicle = "Car" Or Price < 500	**True** (second condition is **True**)
Not Price < 100	**True** (condition is **False**)
Vehicle = "Bike" Xor Price < 300	False **(both conditions are** True**)**

5.2.3: Select Case Decision Structure

In Visual Basic, Select Case decision structure is another way to control the execution of statements in your programs. A Select Case structure is similar to an If...Then structure. However, when the branching depends on one key variable (test case), a Select Case decision structure is more efficient. This efficiency can make your program code more readable and efficient.

Creating Select Case Structures

A Select Case structure begins with the **Select Case** keywords and ends with the **End Select** keywords.

This sample code below contains the syntax for a Select Case decision structure.

```
Select Case variable
Case value1
    statements executed if value1 matches variable
Case value2
    statements executed if value2 matches variable
Case value3
    statements executed if value3 matches variable
.
.
.
End Select
```

Edited by Foxit PDF Editor
Copyright (c) by Foxit Software Company, 2004 - 2007
For Evaluation Only.

▶ **To create a Select Case decision structure**

1. Replace *variable* with the variable, property, or other expression that is to be the structure's key value (test case).

2. Replace *value1*, *value2*, and *value3* with numbers, strings, or other values related to the test case.

If one of the values matches the variable, the statements below its Case clause are executed and Visual Basic continues executing program code after the **End Select** statement.

3. Include any number of Case clauses in a Select Case. If you list multiple values after a case, separate them with commas.

This example shows how you could use a Select Case structure to print an appropriate message about a person's age in a program. If the Age variable matches one of the Case values, an appropriate message is displayed by using a label. See the code below.

```
Select Case Age
Case 16
     Label1.Caption = "You can drive now!"
Case 18
     Label1.Caption = "You can vote now!"
Case 21
     Label1.Caption = "You can drink wine with your meals."
Case 65
     Label1.Caption = "Time to retire and have fun!"
End Select
```

Using a Case Else Clause

A Select Case structure also supports a Case Else clause that displays a message if none of the earlier cases matches. This program code, which works with the Age example, illustrates the Case Else clause. See the example code below.

```
Select Case Age
Case 16
     Label1.Caption = "You can drive now!"
Case 18
     Label1.Caption = "You can vote now!"
Case 21
     Label1.Caption = "You can drink wine with your meals."
Case 65
     Label1.Caption = "Time to retire and have fun!"
Case Else
     Label1.Caption = "You're a great age! Enjoy it!"
End Select
```

Edited by Foxit PDF Editor
Copyright (c) by Foxit Software Company, 2004 - 2007
For Evaluation Only.

Using the Ranges of Test Case Values

Visual Basic lets you use comparison operators to include a range of test values in a Select Case structure. These are the Visual Basic comparison operators that you can use in your programs:

Comparison Operator	Meaning
=	Equal to
< >	Not equal to
>	Greater than
<	Less than
> =	Greater than or equal to
< =	Less than or equal to

The Is and To Keywords

To use the comparison operators, you need to include the **Is** keyword or the **To** keyword in the expression to identify the comparison you're making. The **Is** keyword instructs the compiler to compare the test variable to the expression listed after the **Is** keyword. The **To** keyword identifies a range of values.

Example: Using Is and To

This sample code shows how the decision structure uses **Is**, **To**, and several comparison operators to test the Age variable and to display one of five messages.

```
Select Case Age
Case Is < 13
    Label1.Caption = "Enjoy your youth!"
Case 13 To 19
    Label1.Caption = "Enjoy your teens!"
Case 21
    Label1.Caption = "You can drink wine with your meals."
Case Is > 100
    Label1.Caption = "Looking good!"
Case Else
    Label1.Caption = "That's a nice age to be."
End Select
```

If the value of the Age variable is less than 13, the program displays the message, “Enjoy your youth!” For the ages 13 through 19, the program displays the message, “Enjoy your teens!” and so on.

5.2: Finding and Correcting the Errors

So far, the errors you have encountered in your programs have probably been simple typing mistakes or syntax errors. But what if you discover a nastier problem in your program — one you can't find and correct by a simple review of the objects, properties, and statements in your application? The Visual Basic development environment contains several tools you can use to track down and fix errors (bugs) in your programs. These tools won't stop you from making mistakes, but they often can ease the pain when you encounter a mistake.

Edited by Foxit PDF Editor
Copyright (c) by Foxit Software Company, 2004 - 2007
For Evaluation Only.

5.3.1: Three Types of Errors

As you develop Visual Basic programs, three types of errors can produce unwanted results in your applications. These are described in the following topics:

- Logic errors
- Syntax errors
- Run-time errors

Logic Errors

A logic error is a human error — a programming mistake that makes the program code produce the wrong results. Most debugging efforts focus on tracking down programmer logic errors.

Consider the following If...Then decision structure, which evaluates two conditional expressions and then displays one of two messages based on the result:

```
If Age > 13 And Age < 20 Then
Text2.Text = "You're a teenager."
Else
Text2.Text = "You're not a teenager."
End If
```

Can you spot the problem with this decision structure? A teenager is a person who is between 13 and 19 years old, inclusive, yet the structure fails to identify the person who is exactly 13. (For this age, the structure incorrectly displays the message, “You’re not a teenager.”)

This type of mistake is not a syntax error (the statements follow the rules of Visual Basic); it is a mental mistake, or logic error. The correct decision structure contains a greater than or equal to operator (>=) in the first comparison after the **If...Then** statement:

```
If Age >= 13 And Age < 20 Then
```

Believe it or not, this type of mistake is the most common problem in Visual Basic programs. It's a matter of code that works most of the time — but not all of the time — and it's the hardest problem to track down and fix.

Syntax Errors

A syntax error (compiler error) is a programming mistake that violates the rules of Visual Basic, such as a misspelled property or keyword. As you type program statements, Visual Basic points out several types of syntax errors — and you won’t be able to run your program until you fix each one.

Runtime Errors

A run-time error is any error — usually an outside event or an undiscovered syntax error — that forces a program to stop running. Two examples of conditions that can produce run-time errors are a misspelled file name in a **LoadPicture** function and an open floppy drive.

Edited by Foxit PDF Editor
Copyright (c) by Foxit Software Company, 2004 - 2007
For Evaluation Only.

5.3.2: Fixing Errors

To fix a syntax error, edit the incorrect statement (identified by Visual Basic) in the Code window. To fix a logic or run-time error, use break mode or the **Stop** statement to isolate the mistake.

- Using Break Mode
- Using the Stop Statement

Using Break Mode

One way to identify an error is to execute your program code one line at a time and examine the content of one or more variables or properties as it changes. To do this, you can enter break mode while your program runs and view your code in the Code window.

Break mode gives you a close-up look at your program while the Visual Basic compiler runs it. It's like pulling up a chair behind the pilot and copilot and watching them fly the airplane. But in this case, you can touch the controls.

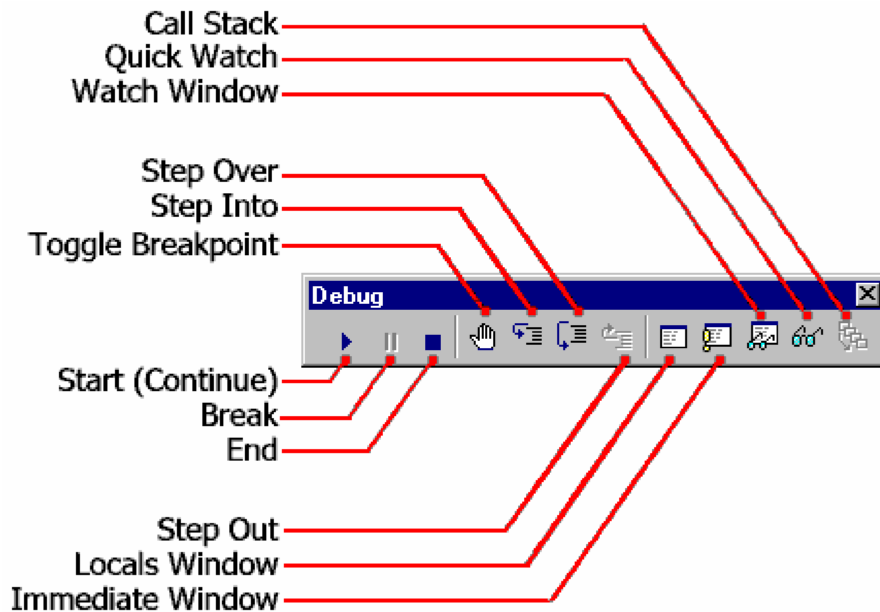
Visual Basic Resources for Debugging

While you debug your program, you may also want to open and use these Visual Basic resources:

Resource	Function
Debug toolbar	Provides tools devoted entirely to tracking down errors.
Watches window	Displays the contents of critical variables you're interested in viewing.
Immediate window	Provides a place to enter program statements and see their effect immediately.

This illustration below shows the **Debug** toolbar, which you open by clicking **Toolbars** in the **View** menu, and then clicking **Debug**.

Edited by Foxit PDF Editor
Copyright (c) by Foxit Software Company, 2004 - 2007
For Evaluation Only.



Using the Stop Statement

In Visual Basic, you can place a **Stop** statement in your code to pause the program and display the Code window. All you have to know is exactly where in the program code you want to enter break mode and start debugging.

For example, you could click the **Break** button to pause the program. Or, you could enter break mode by inserting a **Stop** statement at the beginning of the Command1_Click event procedure.

The illustration below shows how the **Stop** statement method works,

```
Private Sub Command1_Click()
    Stop    'enter break mode
    Age = Text1.Text

    If Age > 13 And Age < 20 Then
        Text2.Text = "You're a teenager."
    Else
        Text2.Text = "You're not a teenager."
    End If
End Sub
```

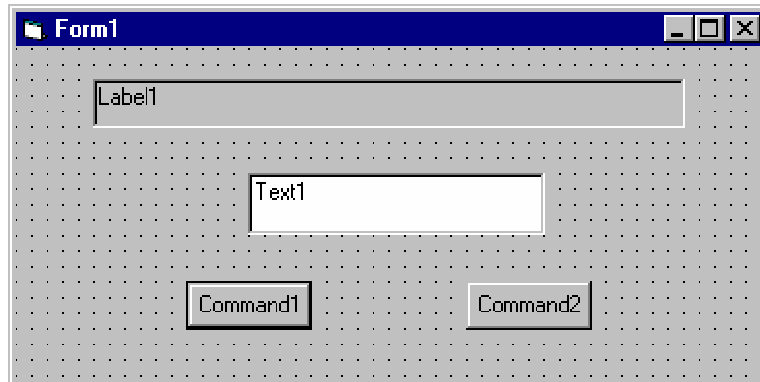
When you run a program that includes a **Stop** statement, Visual Basic enters break mode as soon as it hits the Stop statement. While Visual Basic is in break mode, you can use the Code window, the **Step Into** button, and the **Quick Watch** button just as you would if you had entered break mode manually. Finally, when you finish debugging, just remove the **Stop** statement.

Edited by Foxit PDF Editor
Copyright (c) by Foxit Software Company, 2004 - 2007
For Evaluation Only.

5.4: Exercise 1: Password Validation

1. Start a new project. The idea of this project is to ask the user to input a password. If correct, a message box appears to validate the user. If incorrect, other options are provided.

2. Place a two command buttons, a label box, and a text box on your form so it looks something like this:



3. Set the properties of the form and each object.

Form1:

BorderStyle	1-Fixed Single
Caption	Password Validation
Name	frmPassword

Label1:

Alignment	2-Center
BorderStyle	1-Fixed Single
Caption	Please Enter Your Password:
FontSize	10
FontStyle	Bold

Text1:

FontSize	14
FontStyle	Regular
Name	txtPassword
PasswordChar	*
Tag	[Whatever you choose as a password]
Text	[Blank]

Edited by Foxit PDF Editor
Copyright (c) by Foxit Software Company, 2004 - 2007
For Evaluation Only.

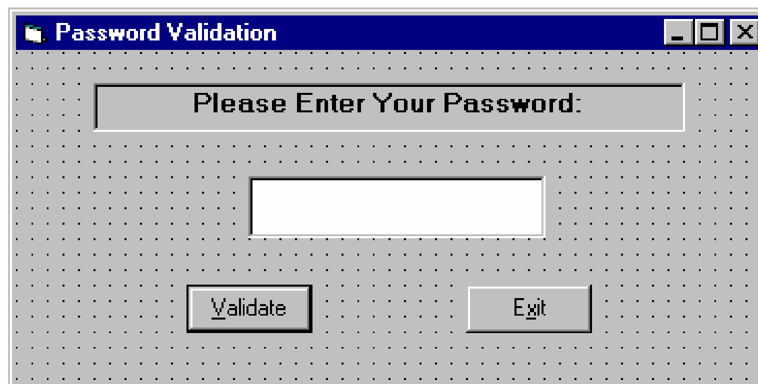
Command1:

Caption	&Validate
Default	True
Name	cmdValid

Command2:

Cancel	True
Caption	E&xit
Name	cmdExit

Your form should now look like this:



4. Attach the following code to the **cmdValid_Click** event.

```
Private Sub cmdValid_Click()
'This procedure checks the input password
Dim Response As Integer
If txtPassword.Text = txtPassword.Tag Then
'If correct, display message box
  MsgBox "You've passed security!", vbOKOnly + vbExclamation, "Access Granted"
Else
'If incorrect, give option to try again
  Response = MsgBox("Incorrect password", vbRetryCancel + vbCritical, "Access
   Denied")
  If Response = vbRetry Then
    txtPassword.SelStart = 0
    txtPassword.SelLength = Len(txtPassword.Text)
  Else
    End
  End If
End If
txtPassword.SetFocus
End Sub
```

Edited by Foxit PDF Editor
Copyright (c) by Foxit Software Company, 2004 - 2007
For Evaluation Only.

This code checks the input password to see if it matches the stored value. If so, it prints an acceptance message. If incorrect, it displays a message box to that effect and asks the user if they want to try again. If Yes (Retry), another try is granted. If No (Cancel), the program is ended. Notice the use of **SelLength** and **SelStart** to highlight an incorrect entry. This allows the user to type right over the incorrect response.

5. Attach the following code to the **Form_Activate** event.

```
Private Sub Form_Activate()
txtPassword.SetFocus
End Sub
```

6. Attach the following code to the **cmdExit_ Click** event.

```
Private Sub cmdExit_Click()
End
End Sub
```

7. Try running the program. Try both options: input correct password (note it is case sensitive) and input incorrect password. Save your project.

   If you have time, define a constant, TRYMAX = 3, and modify the code to allow the user to have just TRYMAX attempts to get the correct password. After the final try, inform the user you are logging him/her off. You'll also need a variable that counts the number of tries (make it a Static variable).

5.4: Exercise 2: International Welcome Program

1.Designing the Form

In this exercise, you will open a new project and build a form containing four label objects, a list box object, and a command button. You'll also set a few important properties.

▶ **To create objects on the form**

1. Start Visual Basic and open a new, standard Visual Basic application.
2. In the toolbox, click **Label**, and then create a large box in the top middle of the form to display a title for the program.
3. In the toolbox, click **ListBox**, and then create a list box on the left side of the form.
4. Above the list box, create a small label. To display program output, create two small labels on the right side of the screen.
5. In the toolbox, click **CommandButton**, and then create a small command button in the bottom middle of the form.
6. Open the Properties window, and then set these object properties on the form:

Object	Property	Setting
Label1	Caption	"International Welcome Program"
	Font	Times New Roman, Bold, 14-point

Edited by Foxit PDF Editor
Copyright (c) by Foxit Software Company, 2004 - 2007
For Evaluation Only.

Label2 Caption	“Choose a country”
Label3 Caption	(Empty)
Label4 Caption	(Empty)
BorderStyle	1 - Fixed Single
ForeColor	Medium red (&H00000080&)
Command1Caption	“Quit”

Note The word *(Empty)* in the table means remove all text from the indicated property setting.

7. From the **File** menu, click **Save Project As**, and then save your form and project to disk under the name MyLab5. You will be prompted for two file names — one for your form file (MyLab5.frm) and one for your project file (MyLab5.vbp).

2.Using the Select Case

In this exercise, you:

- Write an event procedure that uses the Select Case decision structure to process the items in the list box.
- Fill the list box by using the List1 object’s **AddItem** method in the Form_Load event procedure.

▶ **To write the code**

1. Double-click the form (not an object, but the form itself).

   The Form_Load event procedure appears in the Code window.

2. To initialize the list box, type the following program code:

```
List1.AddItem "England"
List1.AddItem "Germany"
List1.AddItem "Spain"
List1.AddItem "Italy"
```

These lines of code use the **AddItem** method of the list box object to add entries to the list box on your form.

3. Open the **Object** drop-down list box, and then click the **List1** object in the list box. The List1_Click event procedure appears in the Code window.
4. To process the list box selection with Select Case, type these lines of program code:

```
Label3.Caption = List1.Text
Select Case List1.ListIndex
Case 0
    Label4.Caption = "Hello, programmer"
```

Edited by Foxit PDF Editor
Copyright (c) by Foxit Software Company, 2004 - 2007
For Evaluation Only.

```
Case 1
    Label4.Caption = "Hallo, programmierer"
Case 2
    Label4.Caption = "Hola, programador"
Case 3
    Label4.Caption = "Ciao, programmatori"
End Select
```

The first statement in this block of code copies the name of the selected list box item to the caption of the third label on the form. The most important property used in the statement is **List1.Text**, which contains the exact text of the item selected in the list box. The remaining statements are part of the Select Case decision structure. The structure uses the **ListIndex** property of the list box object as a test case variable and compares it to several values. The **ListIndex** property always contains the number of the item selected in the list box; the item at the top is 0 (zero), the second item is 1, the third item is 2, and so on. Using **ListIndex**, the Select Case structure can quickly identify the user's choice and display the correct greeting on the form.

5. Open the **Object** drop-down list box.
6. In the list box, click the **Command1** object.
7. In the event procedure, type **End**, and then close the Code window.
8. To save your program to disk, click **Save Project**.

3.Testing the Program

In this exercise, you run the program and verify that each message contains accurate information. If it doesn't, you use the debugging tools to find the problem.

▶ **To test the program**

1. To run the program, click **Start** on the toolbar.

Visual Basic loads the program and displays your opening form.

2. Click each of the country names in the **Choose A Country** list box.

The program should display a greeting for each of the countries listed, and the name of the country should appear above the greeting.

3. If you notice a bug in the program, or if you want to watch Visual Basic run the program code:

- Click **Break** to enter break mode.
- Run the program one statement at a time by displaying the **Debug** toolbar and clicking **Step Into** repeatedly.

8. When you're finished testing the program, click **Quit** to exit.

-END OF CHAPTER 5-