Introduction

Objectives

- To understand the concept of an "event" as it pertains to windows and programming in Visual Basic.
- To become familiar with the syntax for a Visual Basic method.
- To learn how to redefine the properties of control objects from within the body of a method.
- To learn how to write methods in Visual Basic to respond to the Click, GotFocus, and LostFocus events.
Mouse Clicking

You may recall from the previous unit that an "event" was the term used to describe an action that takes place while a program is running, such as a mouse click. VB programs can be viewed as programs that are written to respond to "events" that may occur through the input/output interface of the program. This is the reason VB programming is called "event-driven" programming.

"Mouse clicking" is one of the most common kinds of activities that we, as users, perform when we interact with the windows interface of a program. For example, we can click on a Button, causing some action associated with that button to be performed. Or, we can click on a check box to cause a check mark to appear inside the box. As a third example, we may click on a TextBox so that a cursor prompt will appear inside the box allowing us to enter text. Try the following:

1. Place a Button, Textbox and CheckBox on a Form object anywhere in the VS IDE. Instances of the CheckBox class can be created from the tools provided on the left-hand side of the VS IDE. Look for a small square box with a check mark inside. Your Form window should look something like the following:

2. Click on the "F5" key on your keyboard: that is, "execute the program."
3. Move your cursor (the "arrow") to each of these objects in turn and click once on them.
4. In each case observe what takes place: that is, determine what event occurs.

In each case, you will observe some change to the representation of each object as a consequence of the event of clicking on the object:

- The border thickness of the button graphic changes, giving the appearance that it is pushed;
- The TextBox displays a vertical cursor line, permitting editing;
- The CheckBox displays a check mark, or clears it if one was previously present.

VB assigns a name to this type of action, not surprisingly calling it a "Click" event. The actual response depends on the type of object under the cursor. It is important to note
at this time that besides what you observe happening on your Form, there are other events occurring that are not apparent to the user.

**Click Event Methods**

So far, there are three identifiable events—Click, GotFocus, and LostFocus—that are associated with what occurs when a cursor is clicked on a specific object. Of course, there is usually some reason why a particular control object was placed on the Form in the first place, and the programmer will need to know what should happen as a consequence of clicking on that object. It is the responsibility of the programmer to write a sequence of instructions in the VB programming language that will perform the actions expected by the user. Furthermore, the programmer must write these instructions in such a way that they will be executed when the appropriate event occurs. Programmers perform this task by encapsulating a sequence of programming statements inside a single entity called a "method."

Besides properties, there are methods associated with every class of object. These methods allow the programmer to define the actions that are to occur when an event takes place. The programmer can thus define what should happen when an event occurs to a specific object belonging to the class. For example, you can click on a TextBox object, and TextBox objects can gain or lose focus. Therefore, if you want to customize what should take place when each of these three events occurs in a particular TextBox object, you will need to define three methods, one for each type of event. Of course, if you wish to ignore some events, no instructions need to be performed and no method needs to be written to handle those events. In other words, you only need to write methods to handle those events for which you wish your program to respond.

The Visual Basic IDE provides a library of templates for all events that can occur for each class of object. These templates provide a "shell" that can be used to create the source code for a method that will be associated with a particular instance of a class of objects. This shell can then be edited by the programmer to include the Visual Basic instructions that describe what action is to be taken whenever a particular event occurs for a given object. Much of Visual Basic programming consists of defining these methods.
EXAMPLE: A button object, whose "(name)" property is "Button1" and whose text property is "STOP" is shown on the following form:

A method is required for responding to the event of "STOP" being clicked.

Assuming Form1 is currently visible in your VB design window, by double-clicking on the object labelled "STOP", a new window appears with the tab "Form1.vb". The contents of this window display the template created by the Visual Basic IDE. This template provides the shell for a method called "Button1_Click":

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

To complete this method, an appropriate set of instructions must be provided between the method declaration (first statement) and the end statement. Since the button is labelled "STOP" it seems likely that its purpose is to terminate execution. An appropriate way to achieve this is as follows:

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

Focus Event Methods

When a "Click Event" occurs, there are other events occurring that are not apparent to the user. The change in the border of a button and the appearance of the text cursor in a TextBox indicates the Button object or TextBox is now the object that has the focus of attention during execution. When the event of clicking on any windows object occurs, the attention of the program is directed to the object under the cursor.

At any time, only one object on your Form at most will have the "focus." Therefore, as you move your cursor from one windows object to the next and click on the new object, the focus is shifted and we say that the new object has "got the focus" and the object
that had the focus previously is now said to have "lost the focus." At anytime all objects, except possibly one, will have "lost the focus." The change in appearance of the object that you observed indicates which object has the focus.

As a consequence, besides the "click" event, there are two more events associated with most windows objects, named appropriately the "GotFocus" event and the "LostFocus" event. It is possible to move the focus successively to each object on the Form using the Tab key on the keyboard. Try it. The sequence in which each object is given the focus is determined initially by the order that you placed the objects on the Form when you created it.

Given a collection of objects on a form, during execution a Visual Basic program can detect the occurrence of a click event and determine what object now has the focus and which object just lost the focus. Therefore, after you have created a form with objects, you can start to write a program that performs certain steps based on what you wish to have happen each time one of the control objects on your form is clicked.

The template for a method that responds to Click events on a button object, one simply has to double-click on the button object in the Form window. Obtaining a template for GotFocus and LostFocus events is somewhat more complex.

When an object is double-clicked in a form, a default template is provided. In the case of Button objects, the template provided is for a Click method. When TextBox objects are double-clicked, the default template is for a "TextChanged" method. In neither case is a template for a GotFocus or LostFocus method provided.

To obtain such a template:

1. Click on the object for which you wish to create a control method. This will produce a default template for that object and open the design window displaying the template for that object.
2. At the top left of the design window, the name of the object will be displayed. This is a pull-down list of all objects on your form, called the "Objects List." Note that you won't need to change the name shown if you performed step 1.
3. At the top right of the design window will be the name of the default template method. This is a pull-down list of all methods that are associated with the object shown at the top right of the window, called the "Methods List."
4. Click on the Methods List and select the control method whose template you wish to define. For now, this will be either GotFocus or LostFocus.
5. When you click on the name of the desired method, the template will appear in your source code. You can now edit the template by adding whatever VB statements you wish to be performed when an event occurs that activates the method.
6. If you do not require the default template that you created in step 1, you should delete this code from your design window.
Linking Events and Objects

Event-driven programming is about using objects in a Form to cause segments of source code to be executed. Therefore each control object on the Form has a set of methods associated with it to respond to events that can happen to that object. Previously, three such methods have been described: GotFocus and LostFocus are methods that are applicable to both TextBox and Button objects, whereas Click is a method applicable only to the Button method. The event that activates these methods is as follows:

<table>
<thead>
<tr>
<th>METHOD</th>
<th>EVENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Button_Click</td>
<td>Button is clicked</td>
</tr>
<tr>
<td>Object_GotFocus</td>
<td>Object (Button or TextBox) is clicked</td>
</tr>
<tr>
<td>Object_LostFocus</td>
<td>Object other than Object (Button or TextBox) is clicked</td>
</tr>
</tbody>
</table>

Consider the following Form, created to provide a Button that when clicked will clear whatever text is shown in the TextBox:

When the Button object is added to the form the VB IDE gives the object variable name "Button1." Similarly, when the TextBox object is added to the form, the VB IDE gives the object the variable name "TextBox1." You can verify this by examining the "(Name)" property of each object. The label shown on the Button is the value of the "Text" property of Button1. It is initially the same value as the (Name) property, but can be changed by assigning a different value to the "Text" property, such as "CLEAR." The value of the Text property of the TextBox is initially spaces but can be assigned a different value as well, such as the string "Click button to Clear".
Assuming these values are assigned to the Text properties of the Button and TextBox objects, the Form becomes:

![Form with Button and TextBox](image)

What is now required is that the CLEAR button be linked with the TextBox so that when the event of clicking the button occurs, the value showing in the TextBox is replaced by spaces. This requires that the Button1_Click method be created to respond to the Click event by placing the String "" in Text property of TextBox1. This action is defined by the following assignment statement:

```csharp
TextBox1.Text = ""
```

To create the necessary source code to define the method Button1_Click:

1. Double-click on the Button object on the form. This will open a new window showing the source code that accompanies your Form. In particular, a template for the Button1_Click method is provided that consists of a "method declaration" and an "end" statement.
2. Place the source code you wish to execute in the body of the template; that is, between the method declaration and end statements.

The result should be the following source code:

```csharp
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        TextBox1.Text = ""
    End Sub
End Class
```

The link is now complete. To test the implementation, simply run the application by pressing the F5 key on the keyboard. Then click on the "CLEAR" button and observe the clearing of the message from the TextBox.
Creating a Complete Program

To write a program in Visual Basic:

1. Determine the requirements analysis for the inputs, outputs, and events;
2. Construct the User Interface Form;
3. Identify what events will need to be defined in Visual Basic;
4. Complete the encoding of the methods required to link each object on the Form with the identified events.

The following example illustrates writing a VB program to convert miles to kilometres.

1. Requirements Analysis

   INPUT REQUIREMENTS
   
   INPUT   TYPE     PURPOSE     REPRESENTATION
   Miles    Numeric     Distance in miles    TextBox

   OUTPUT REQUIREMENTS
   
   OUTPUT   TYPE     PURPOSE     REPRESENTATION
   Kilometres Numeric     Distance in kilometres    TextBox

   EVENT REQUIREMENTS
   
   EVENT   EFFECT     REPRESENTATION
   Calculate    convert distance to kms    Button
   Clear    clear input value    Button
   Stop    terminate execution    Button

2. Construct User Interface Form

   From the requirements analysis, there are two TextBoxes and three Buttons required. Labels will be used to identify the TextBoxes. A suitable Form is:
3. Identify Required Events

The events of interest are those that occur as a result of positioning the mouse cursor on one of the three buttons and clicking it. Therefore, the Click methods for each of the three buttons need to be defined so that each performs the function suggested by the Button names and defined under the "PURPOSE" column in the requirements analysis above.

The templates for the click methods for the three buttons can be generated by double-clicking each button. This results in the following code being generated:

```vbnet
Public Class Form1
    ' STOP method
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    End Sub

    ' CALC method
    Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
    End Sub

    ' CLEAR method
    Private Sub Button3_Click(sender As Object, e As EventArgs) Handles Button3.Click
    End Sub
End Class
```

Note that the three templates are encapsulated in a "Class declaration". This is because Form1 is an object that includes the three Button objects. Also, no methods are required for the TextBoxes because there is no requirement for the program to do anything when text is typed into them.
4. Complete the Templates

The following program completes the definition of each required method:

```vbnet
Public Class Form1
' STOP method
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
Me.Close()
End Sub

' CALC method
Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
TextBox2.Text = 1.609 * TextBox1.Text
End Sub

' CLEAR method
Private Sub Button3_Click(sender As Object, e As EventArgs) Handles Button3.Click
TextBox1.Text = ""
TextBox2.Text = ""
End Sub
End Class
```

Explanation of the statements in the program ↓

If this is your first opportunity to examine the source code of a Visual Basic program, then it will be helpful to understand the structure and purpose of each statement you see in the program.

All programs are organized into contiguous lines of programming language source code called "blocks." Each block defines a logical entity of the program. In the example above, the statements of the program are grouped into two types of blocks:

1. "Class" blocks begin with a statement such as "Public Class Form1" and end with the statement "End Class." These two statements enclose program elements that describe kinds of objects. In the example, the purpose of the Class block is to identify methods to handle the events associated with the objects on the Form of our example.

2. "Sub" blocks begin with a statement such as "Private Sub . . ." and end with the statement "End Sub." In the example source code, there are four Sub blocks and each defines a method associated with a particular event. By inspecting the name of each method that follows the keywords "Private Sub", we can infer what event is being handled by each method and for which object. For example, "Private Sub Button1_Click" identifies the method that will be executed if the Button whose (Name) property is Button1 is "clicked." The first method, called "Form1_Load" describes what happens when the program is initial run and Form1 is first displayed.

Further inspection of the statements that introduce each method reveals a lengthy sequence of text enclosed in parentheses. This text is called the "parameter list of the method." This text may continue for several lines, and except for the last line, each line ends with a space followed by an underscore character. In VB, a single statement that
The first line of every method is called a "method declaration statement." Every method declaration statement is introduced by the keyword "Private" (or "Public"), followed by the keyword "Sub," followed by a method name and a parameter list. The purpose of the parameter list will be examined in more detail later. For now, it is sufficient to understand that the parameter list provides a way for other parts of a program to communicate information with the method being defined.

After the method declaration statement, all programming lines that follow until the next "End Sub" statement form what is called the "body" of the method. The body provides the instructions that are to be performed when an event occurs that causes the method to be executed.

In our example, the event of clicking on Button3 (labelled CLR) should cause the text fields of both TextBoxes to be cleared. This is achieved by the two statements:

```vbnet
TextBox1.Text = ""
TextBox2.Text = ""
```

Informally, each of these statements says "Assign no characters to the Text properties of TextBox1 and TextBox2." These statements constitute the body of the Button1_Click method, performed when Button1, "CLR", is clicked. Note that here we have an example of setting the properties of an object using assignment statements, as discussed in Unit 2.

When the event of clicking on Button2 (labelled CALC) occurs, we require a calculation to be performed on the value entered in TextBox1. This value must then be displayed in TextBox2. This effect is achieved by the statement:

```vbnet
TextBox2.Text = 1.609 * TextBox1.Text
```

This statement is obtained from the formula for converting miles to kilometres:

Distance in kilometres = \( 1.609 \times \text{distance in miles} \)

and defines the body of the Button2_Click method. This method is executed when someone clicks Button2, the "CALC" button.

Button1, labelled "STOP", should terminate the program when clicked. To see how this is achieved, look at the method body of the Button1_Click method. There is but one statement in the body:

```vbnet
Me.Close()
```

This statement is inside a method that itself is inside the Class block "Form1." The term "Me" refers to the object "Form1" and as the rest of the expression suggests, this statement simply "closes" Form1. As you should know from your experience with windows environments, when a window is closed, the program associated with that
window terminates execution and the window disappears from the screen. The statement "Me.Close()" has exactly this effect, and so clicking on the "STOP" button terminates the program as would clicking on the red box labelled with an "X" in the upper right corner of the Form window.

This simple example employed four types of object: Form, Label, TextBox, and Button. With just these four classes of object, simple windows user interfaces can be designed for many programming problems. Nevertheless, the VS IDE provides many other types of objects for use in creating a window by providing other means to supply input and display output. With the application of such tools it is possible to construct elaborate yet attractive, "user-friendly" windows for any application.

Revising

The "Distance Converter" program used only the "Click" event method for each Button. In this example, use will be made of the "LostFocus" event method to write a program that can not only convert from miles to kilometres but also from kilometres to miles.

When a solution to a similar problem exists it is common practice to try and use the existing solution to develop a solution to the new problem. The first example seems similar to the previous one, so it may be advantageous to modify that solution rather than to formulate a new one since it will save some of the design time.

One possible solution is simply to add another CALC button that converts kilometres to miles when it is clicked. This approach will require revision of the "EVENTS REQUIREMENTS" and redesign of the form to add the second CALC button. Before reading further, you should attempt to modify the previous solution to solve the problem in this way.

A different solution is possible if we take advantage of another type of event, specifically the "LostFocus" event described earlier. If you have tried the previous solution, you will have noticed that you needed to click on a TextBox before you could enter text into its field. The reason this is the case is that it was necessary to give the TextBox the focus first. We can use this observation to redesign the previous solution so that it meets the needs of this problem without introducing another button. In effect we do not need to change the design of the Form. Rather, we simply need to provide suitable method bodies for the `TextBox1_LostFocus` and the `TextBox2_LostFocus` methods that will be needed if our program is to respond to "LostFocus" events on each of the two TextBoxes. This saves the need to revise either the Requirements or the Form.

Two "LostFocus" methods will be required, one for each TextBox. To obtain the templates for these methods, double-click each of the two TextBoxes on Form1 of the solution to example 1. There will now be five methods in the VB program: three "Button_Click" methods (one for each Button) and two "TextBox_TextChanged" methods. Since "LostFocus" methods are desired, all occurrences of the names can be edited from "TextChanged" to "LostFocus".
An algorithm for this solution is helpful in understanding the Visual Basic source code that is used. **Pseudocode** is used to describe the algorithm:

1. If `TextBox1` has focus then calculate the conversion from miles to kilometres.
2. If `TextBox2` has focus then calculate the conversion from kilometres to miles.
3. When `CALC` is clicked, it gains the focus and either `TextBox1` or `TextBox2` loses the focus.
4. When either `TextBox1` or `TextBox2` lose the focus, the values in `TextBox1` and `TextBox2` are displayed.
5. When `CLEAR` is clicked, clear both `TextBoxes`.
6. When `STOP` is clicked, terminate execution.

The following source code provides an implementation:

```vbnet
Public Class Form1
    ' STOP method
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Me.Close()
    End Sub
    
    ' CALC method
    Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
    End Sub
    
    ' CLEAR method
    Private Sub Button3_Click(sender As Object, e As EventArgs) Handles Button3.Click
        TextBox1.Text = ""
        TextBox2.Text = ""
    End Sub
    
    ' Miles to Kilometres (Miles entered)
    Private Sub TextBox1_LostFocus(sender As Object, e As EventArgs) Handles TextBox1.LostFocus
        TextBox2.Text = 1.609 * TextBox1.Text
    End Sub
    
    ' Kilometres to Miles (Kilometres entered)
    Private Sub TextBox2_LostFocus(sender As Object, e As EventArgs) Handles TextBox2.LostFocus
        TextBox1.Text = 0.622 * TextBox2.Text
    End Sub
End Class
```

Note that the only purpose of the `CALC` method, `Button2_Click`, is to take the focus away from either `TextBox`. Therefore there is nothing to do in the body of this method.