Lab 3: Event

An event is something that happens. An event in programming terminology is when something special happens. These events are so special that they are built in to the programming language. VB.NET has numerous Events that you can write code for. And we're going to explore some of them in this section.

1. Exploring the Click Event

Buttons have the ability to be clicked on. When you click a button, the event that is triggered is the Click Event. If you were to add a new button to a form, and then double clicked it, you would see the following code stub:

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

End Sub
```

This is a Private Subroutine. The name of the Sub is Button1_Click. The Event itself is at the end: Button1.Click. The Handles word means that this Subroutine can Handle the Click Event of Button1. Without the arguments in the round brackets, the code is this:

```
Private Sub Button1_Click( ) Handles Button1.Click
```

You can have this Button1_Click Sub Handle other things, too. It can handle the Click Event of other Buttons, for example. Try this.

- Start a New project
- Give it the name it Events
- When your new Form appears, add two Buttons to it
- Double click Button1 to bring up the code
- At the end of the first line for the Button, add this:

```
Handles Button1.Click, Button2.Click
```

Add a message box as the code for the Button. Your code window might then look like this:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click, Button2.Click
        MsgBox("I can handle two buttons!")
End Sub
```

Run your program, and then click both of the buttons in turn. The same message box appears, regardless of which one you clicked.

The reason it did so was because the Events that the Button1.Click Subroutine can handle are at the end: the Events for Button1.Click AND Button2.Click.

You can add as many Events as you want on the End. As long as the Subroutine can handle them, the Event will happen. For example, you could create two more buttons, and then add the Click Event on the end of the first button:

```
Handles Button1.Click, Button2.Click, Button3.Click, Button4.Click
```

When you click any of the four buttons, the code inside of the Button1_Click Subroutine will be triggered.

However, if you double clicked button2 to try to bring up its coding window, you'd find that the cursor is flashing inside of the code for Button1_Click. Because you've attached the Click Event of button2 to the Button1 Subroutine, you can't have a separate Click Event just for Button2. This Click Event is Handled By the Subroutine called Button1_Click.

Event Arguments

The arguments for a Button's click event, the ones from the round brackets, are these two:

```
sender As Object, e As EventArgs
```

This sets up two variables: one called sender and one called e. Instead of sender being an integer or string variable, the type of variable set up for sender is System.Object. This stores a reference to a control (which button was clicked, for example).

For the e variable, this is holding an object, too - information about the event. For a button, this information might be which Mouse Button was clicked or where the mouse pointer was on the screen.

But because this is the Click Event, there's not much more information available: either the button was clicked or it wasn't.

But you can use other Events available to the button. One of these is the MouseDown Event. The information for the event would be which button was clicked, where the mouse pointer was when the mouse button was held down, and something called Delta (a count of how many notches have been rotated on a mouse wheel).

2. Exploring the MouseDown Event

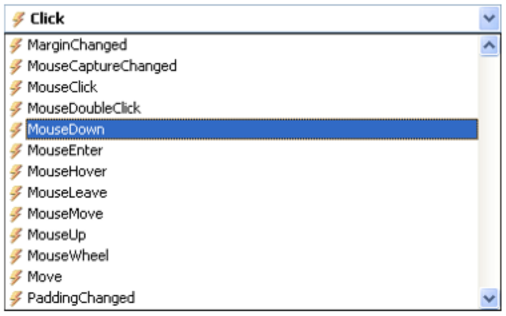
The MouseDown event is available to many controls on the form. A Form can detect when the mouse was held down on it; a textbox can detect when the mouse was held down inside of it; and a Button can detect which mouse button was held down to do the clicking.

We'll see how it all works right now. First, delete the all but leave one of the buttons on your form. Go back to your coding window, and delete any Handles code except for Handles Button1.Click. Your coding window should look something like this one:

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

        End Sub
```

Right at the top of the code window, it says Button1 and Click. The lightning bolt next to Click signifies that it is an Event. If you click the drop down box, you'll see a list of other available events:



Scroll down and find the MouseDown event, as in the image above. When you click on it, a new code stub appears:

```
        Private Sub Button1_MouseDown(sender As Object, e As MouseEventArgs)
        Handles Button1.MouseDown

        End Sub
```

This is a Private Subroutine called Button1_MouseDown. Notice that it Handles the Button1 MouseDown event, and not Button1.Click.

Event Arguments

In between the round brackets of the Subroutine, we still have sender As Object. But we have a new argument now:

e As MouseEventArgs

The name of the variable is still e. But the type of Object being stored inside of the e variable is different:

MouseEventArgs

The bit on the end of all that is what we're interested in: MouseEventArgs. This stands for Mouse Events Arguments. What is being stored inside of the e variable is information the Mouse Event: Did you click a button, if so which one?

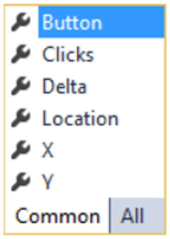
The only thing you need to do to detect which button was pressed is to access a property of the e variable. Let's see how to do that.

Which Button was clicked?

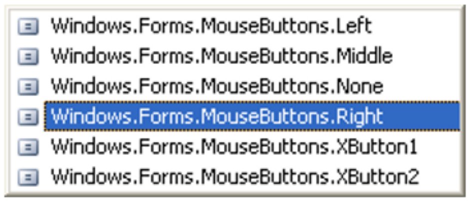
Inside of the Button1_MouseDown Subroutine, type the following code:

```
If e.Button = MouseButtons.Right Then
        MsgBox("Right Button Clicked")
End If
```

As soon as you type the letter "e", you'll see this pop up box:



To detect which button was clicked, you need the first Property on the list: Button. Double click this property to add it to your code. Then after you typed the equals sign, another pop up list appears. This one:

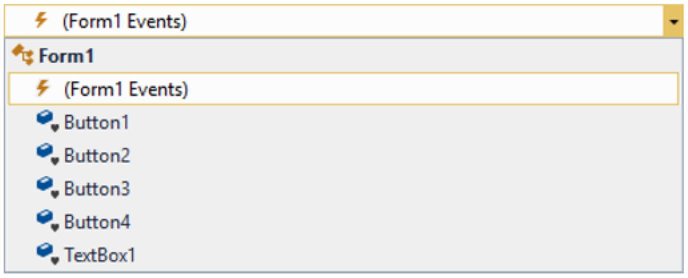

Windows.Forms.MouseButtons.Left
Windows.Forms.MouseButtons.Middle
Windows.Forms.MouseButtons.None
Windows.Forms.MouseButtons.Right
Windows.Forms.MouseButtons.XButton1
Windows.Forms.MouseButtons.XButton2

This is a list of available buttons that VB can detect. Left and Right are the ones you'll use most often. When you're finished writing your code, run your program. Click the button with your Left mouse button and nothing will happen. Click it with the Right mouse button and you should see the message box display.

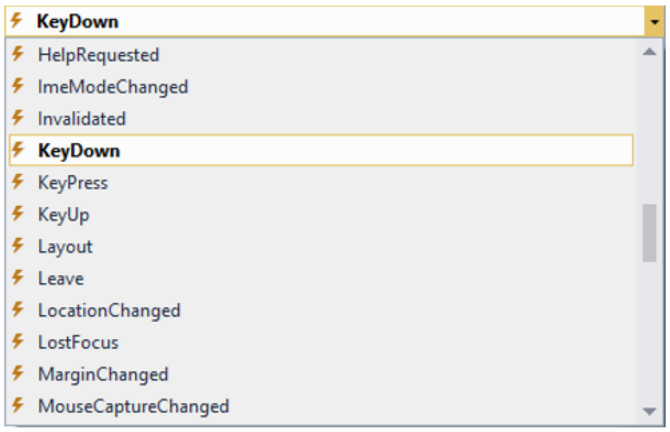
3. Exploring the KeyDown Event

Another useful event is the KeyDown event. As its name suggest, this allows you to detect when a key on the keyboard was held down. This is useful for things like validating text in a textbox.

To test it out, add a textbox to your form. Change the Text property of the textbox to "Press F1 for help." Bring up your code window and click the arrow that reveals the list of controls and objects in your project:



Click on your textbox from the list to select it, as in the image above. Then click the arrow on the Event drop down box to reveal the events available to the textbox. Scroll down and select the KeyDown event:



When you select the KeyDown event, a code stub appears:

```
Private Sub TextBox1_KeyDown(sender As Object, e As KeyEventArgs) Handles
TextBox1.KeyDown

End Sub
```

The event that is being Handled is the KeyDown event of TextBox1. Notice, though, that there is a slightly different argument in round brackets:

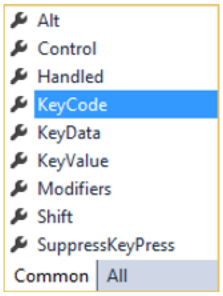
e As KeyEventArgs

Again, the variable name is still e. But now we have something called KeyEventArgs at the end. This means that the variable e will hold information about the Key on the keyboard that you're trying to detect.

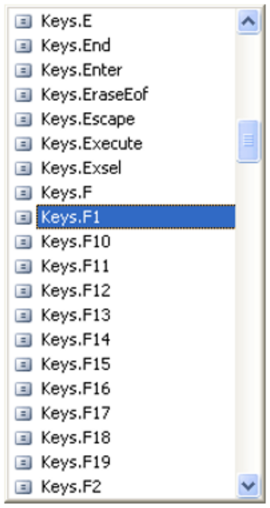
To see what properties the e variable has available to it, add the following to your TextBox1_KeyDown code:

```
        If e.KeyCode = Keys.F1 Then
                TextBox1.Clear()
                MsgBox("Help!!!")
        End If
```

As soon as you type the full stop after the letter "e", you'll see this pop up box:



Double click a property to add it to your code. After you type an equals sign, you'll get another pop up box:



The list is a list of keys on your keyboard, some of which you'll have and others that you won't. Scroll down the list until you come to Keys.F1, and double click the item to add it to your code.

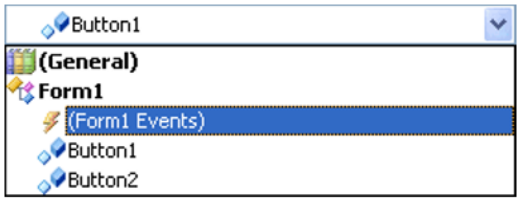
Try your program out. Press F1. When the F1 key is pressed, you should see the message box appear.

4. Exploring the Form Load Event

An important event you'll want to write code for is the Form Load event. You might want to, for example, set the Enabled property of a control to False when a form loads. Or maybe blank out an item on your menu. You can do all this from the Form Load event.

Add another button to your form fo

r this example, and we'll see how the Form Load event works. Bring up your coding window, and select the Form1 Events from the drop down box:



In the events drop down box, select Load. A code stub for the Form Load event is then added to your code. Type in the following as the code for the Load Event:

```
MsgBox("Form Load Event")
```

Run your program. You should see the message box display before the Form loads. To switch off your second Button before the Form loads, add this to your code:

```
Button2.Enabled = False
```

Run your program again. You should see that button is no longer available for clicking on.