Access VBA Made Easy

Events

04

www.AccessAllInOne.com

This guide was prepared for AccessAllInOne.com by:
Robert Austin

This is one of a series of guides pertaining to the use of Microsoft Access.

© AXLSolutions 2012
All rights reserved. No part of this work may be reproduced in any form, or by any means, without permission in writing.

Contents

Events ... 3

- Form and Report Events ... 3
- Related Objects ... 3
- How to create an event in the VBA editor ... 3
- Forms, Controls and their events ... 3
  - Mouse Events ... 5
  - OnClick ... 5
  - OnDblClick ... 7
  - OnGotFocus and OnLostFocus ... 8
- OnMouseDown, OnMouseUp ... 10
- OnMouseMove ... 11
- OnKeyDown, OnKeyUp ... 12
- OnKeyPress ... 13
- Form Events – OnOpen, OnLoad, OnResize, OnActivate, OnUnload, OnDeactivate and OnClose ... 14

Closing a Form ... 17

  - Recordset Control Events – OnCurrent, BeforeUpdate, AfterUpdate, OnChange ... 18
- OnTimer Events ... 21
- Questions ... 22
- Answers - Events ... 23

Events

Form and Report Events

An event is any interaction that a human has with the application or when parts of the application change state, which is invariably because a user has requested something; usually this will involve the user clicking a button or entering some text but can also involve touching the screen, just leaving the mouse curser over a box or form, tabbing around, cycling through records or a chain of events.

The events that we will be concentrating upon in this unit are those associated with Access forms.

Related Objects

Please open up the CodeExamplesVBAForBeginners application. The objects we will be using will be frmEvents, frmStudentsDataEntry and frmTimer.

How to create an event in the VBA editor

Modules for forms are automatically created by Access when we click on the ellipsis in the Properties | Events tab.

The form must be open in design view when you first create an event.

All events that a Form or Object can react to are in the Events tab.

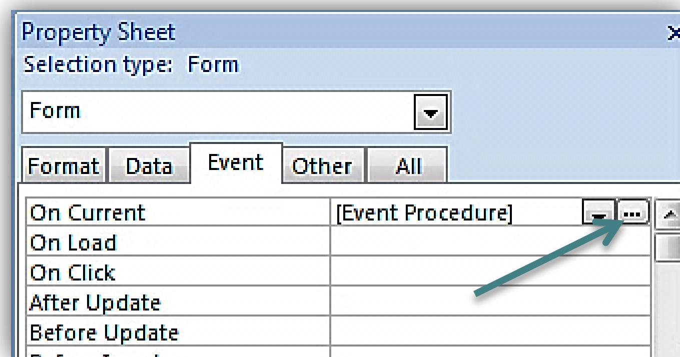


Figure 4.1

In figure 4.1 an <u>On Current</u> event already exists. We know this because [Event Procedure] is written in the <u>On Current</u> field of the property sheet.

Forms, Controls and their events

Forms are not simple objects. They are made up of a Header, a Detail, a Footer and the Form itself. Each of these parts of the form has their own set of events which you can see change as you click on them. The little square in the top left is the form itself. You can add controls to the Header, Detail and Footer areas.

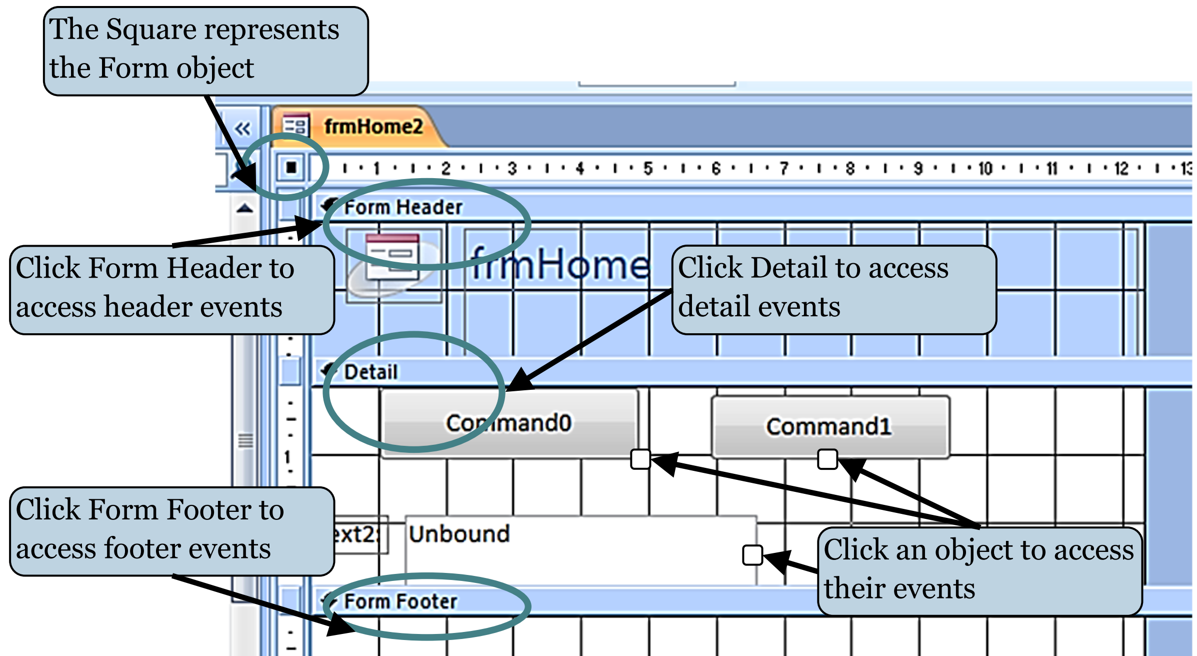


Figure 4.2

Note: *Although the form is broken down into several parts, the vast majority of the time you will be dealing with events related to opening the form, closing the form and events for different controls (combo-boxes, text-boxes, command buttons) that are usually found in the detail section of the form. This has been reflected in the material for this unit.*

Mouse Events

The main mouse events occur when you click an object such as a section of the form or a control. A click event actually consists of a MouseDown, MouseUp, MouseClick and MouseDblClick. These can then also trigger another set of events LostFocus, GetFocus, Enter, Exit .

Please open frmEvents

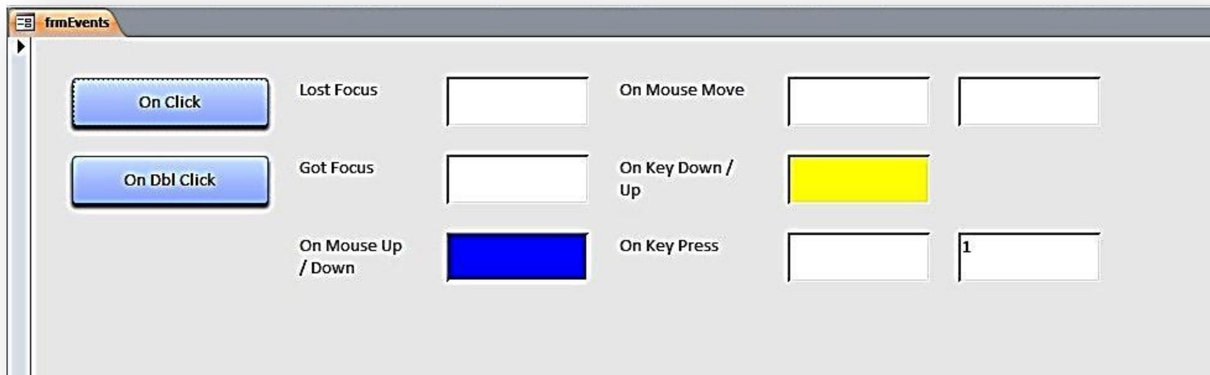


Figure 4.3

Using frmEvents we have set up several controls to demonstrate what certain events are triggered by and how they behave.

OnClick

The OnClick occurs when a Control object is clicked. This event is most commonly associated with a command button but can also be used with controls such as text-boxes and combo-boxes.

To get to the code associated with the OnClick event of cmdOnClick button, we open the form in design view, select cmdOnClick in the property sheet and click on the ellipsis on the far right hand side.

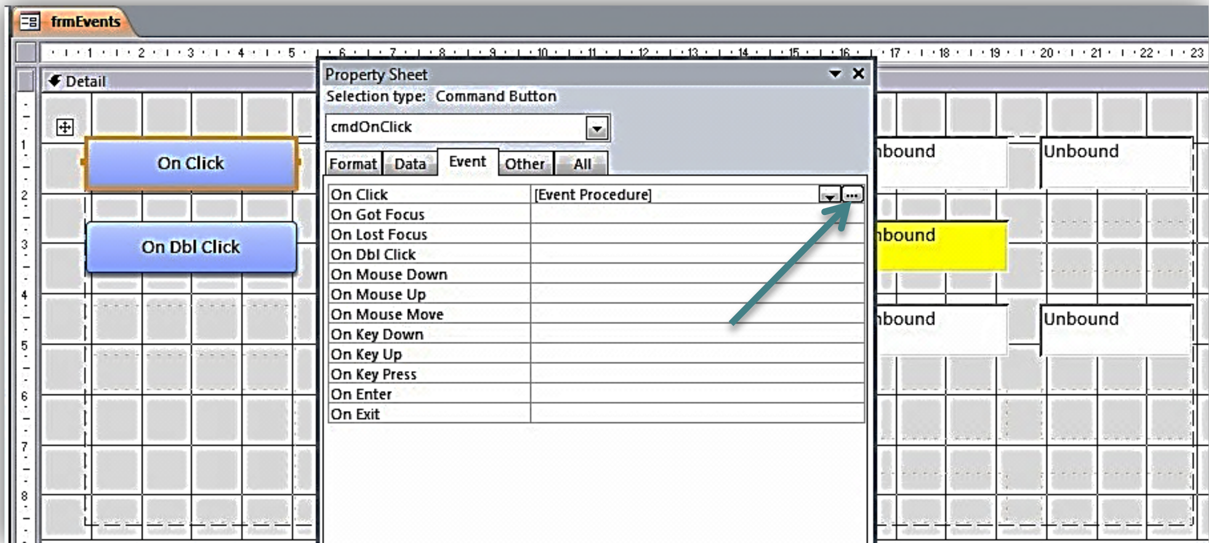


Figure 4.4

The VBA editor will open up with all the procedures related to that form on display. The curser should be flashing in *Private Sub cmdOnClick_Click()*.

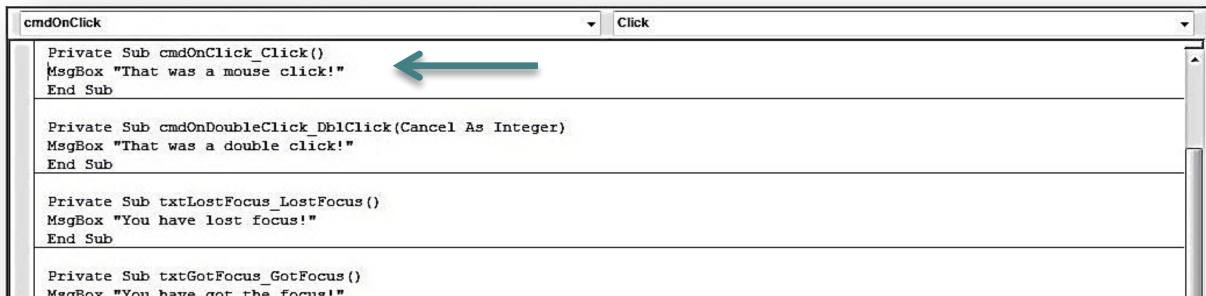


Figure 4.5

The code associated with cmdOnClick is displayed in Figure 4.6

```
1   Private Sub cmdOnClick_Click()
2   MsgBox "That was a mouse click!"
3   End Sub
```

Figure 4.6

Go back to frmEvents, change it to Form view and click the button to see what happens.

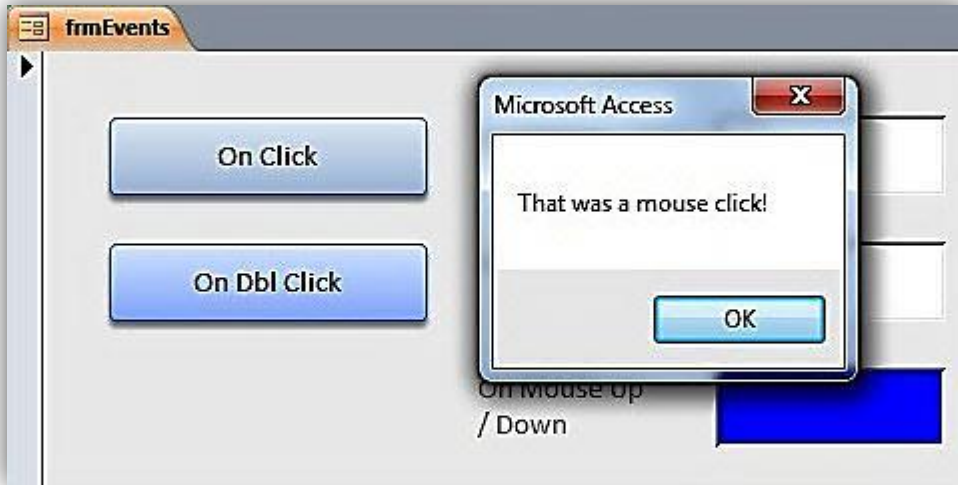


Figure 4.7

The OnClick event fired and the statement `MsgBox` *"That was a mouse click!"* `was executed.`

OnDblClick

The double click event occurs when the system identifies that the user has double-clicked an object.

Here is the code associated with the double click event for the cmdOnDoubleClick button.

```
1   Private Sub cmdOnDoubleClick_DblClick(Cancel As Integer)
2   MsgBox "That was a double click!"
3   End Sub
```

Figure 4.8

Double click cmdOnDoubleClick and this is what you should see:

Figure 4.9

OnGotFocus and OnLostFocus

The Got Focus event occurs when a control receives the focus. This can be either by clicking the control or tabbing into it. If a text-box receives the focus the curser flashes inside it whereas when a button receives the focus you can just make out a faint dotted line around the edge.

Figure 4.10

In figure 4.10 the On Dbl Click button has the focus and the dotted line is just visible.

We can trigger the OnGotFocus event of txtGotFocus by either clicking into txtGotFocus or tabbing over from cmdOnDblClick. Either way the OnGotFocus event will produce this result:

Figure 4.11

```
1   Private Sub txtGotFocus_GotFocus()
2   MsgBox "You have got the focus!"
3   End Sub
```

Figure 4.12

The code associated with the OnGotFocus event is displayed in Figure 4.12.

The OnLostFocus event triggers when a control loses the focus. If the focus is on a button (cmdOnDoubleClick) and you tab or click into txtOnGotFocus, cmdOnDoubleClick loses the focus right before txtOnGotFocus gets the focus.

To demonstrate this concept click into txtOnLostFocus (not txtOnGotFocus). The curser should be flashing within the text-box. Now click into txtOnGotFocus. You should see two messages come up one after another. The first will read:

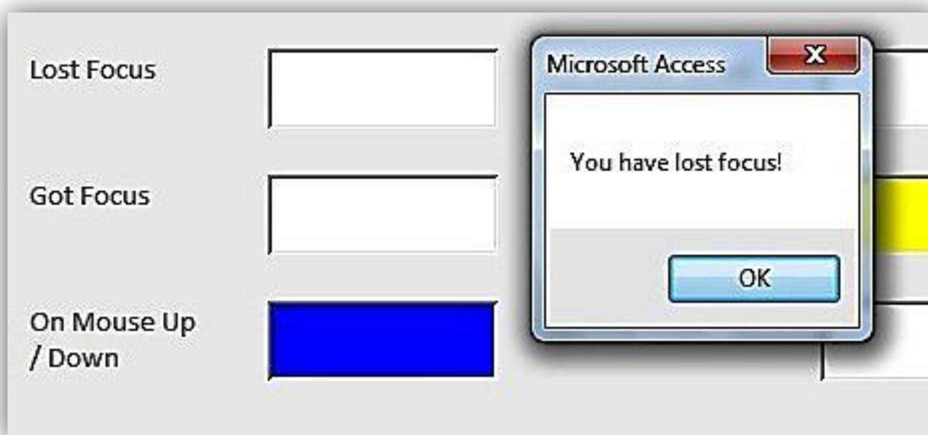


Figure 4.13

And the second will read:

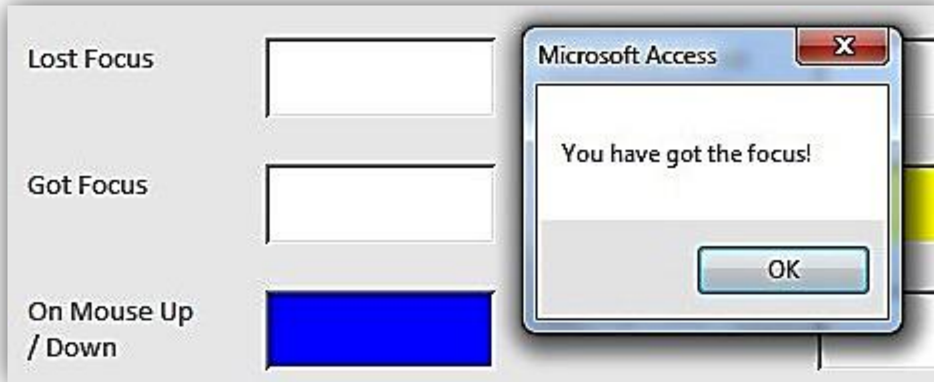


Figure 4.14

What has happened is that the first event to fire was the OnLostFocus event of txtOnLostfocus which brought up the message box in Figure 4.13 and second event to fire was the OnGotFocus event of txtOnGotFocus which brought up the message box in Figure 4.14.

OnMouseDown, OnMouseUp

Although the OnClick event represents the simple clicking of a mouse, it is actually possible to break it down into two separate events; the OnMouseDown event and the OnMouseUp event. The OnMouseDown event is fired when the mouse button is depressed and the OnMouseUp event is fired when the button is released. Before we go to frmEvents to test it out, have a look at the code associated with the two events. In this case we are using both these events on one control – txtOnMouseUpDown.

```
1   Private Sub txtOnMouseUpDown_MouseDown(Button As Integer, Shift As
2   Integer, X As Single, Y As Single)
3   Me.txtOnMouseUpDown.BackColor = vbRed
4   End Sub
5
6   Private Sub txtOnMouseUpDown_MouseUp(Button As Integer, Shift As
7   Integer, X As Single, Y As Single)
8   Me.txtOnMouseUpDown.BackColor = vbBlue
9   End Sub
```

Figure 4.15

Try and work out from the code in Figure 4.15 what is going to happen when the two events fire.

Note: *The arguments that the OnMouseDown and OnMouseUp events take may seem complicated but are anything but.*

- *Button refers to which mouse button was pressed or released to cause the event to trigger.*
- *Shift refers to whether any of the SHIFT, CTL or ALT keys were depressed at the time the event fired.*
- *X and Y refer to the mouse coordinates.*

We will be using the X and Y arguments when discussing OnMouseMove later on.

When the mouse button is depressed the BackColor property of txtOnMouseUpDown changes to VbRed:

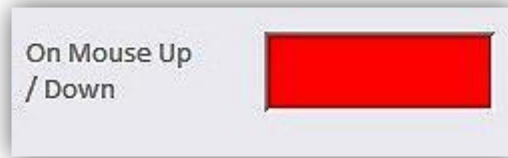


Figure 4.16

When the mouse button is released the BackColor property of txtOnMouseUpDown changes to VbBlue.

Figure 4.17

Press and release the mouse button slowly to really see the difference between the two events.

OnMouseMove

The OnMouseMove event corresponds to the mouse curser hovering over a control that contains that event procedure. The clicking of buttons makes no difference as it is merely the position of the curser that is important.

In form events there is a text-box named txtOnMouseMove. This text-box has the OnMouseMove event procedure and the code looks like this:

```
1   Private Sub txtOnMouseMove_MouseMove(Button As Integer, Shift As
2   Integer, X As Single, Y As Single)
3   Me.txtOnMouseMoveCoordinates.Value = X & " " & Y
4   End Sub
```

Figure 4.18

txtOnMouseMoveCoordinates is the text-box immediately to the right of txtOnMousemove and *X* and *Y* refer to the coordinates of the mouse. What do you think will happen when you hover the mouse curser over txtOnMouseMove?

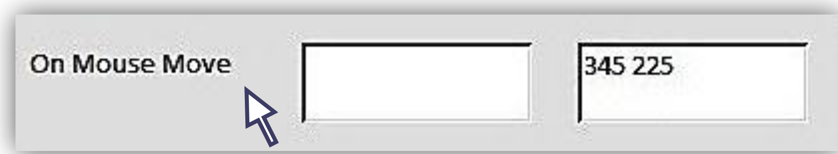


Figure 4.19

As you hover the mouse cursor over txtOnMouseMove, the *X* and *Y* coordinates are being displayed in txtOnMouseMoveCoordinates and as you move the position of the curser, the coordinates change.

OnKeyDown, OnKeyUp

The OnKeyDown and OnKeyUp events are very similar to the OnMouseDown and OnMouseUp events but are triggered by the depressing and releasing of certain keys. On frmEvents we have a text-box called txtOnKeyUpDown where we will be testing out the two events. Before testing out the events let's take a look at the code behind the text-box.

```
1   Private Sub txtOnKeyUpDown_KeyDown(KeyCode As Integer, Shift As
2   Integer)
3   Me.txtOnKeyUpDown.BackColor = vbGreen
4   End Sub
5
6   Private Sub txtOnKeyUpDown_KeyUp(KeyCode As Integer, Shift As Integer)
7   Me.txtOnKeyUpDown.BackColor = vbYellow
8   End Sub
```

Figure 4.20

What do you think will happen when you press a key within the txtOnKeyUpDown text box?

Let's say you were pressing the *ctrl* key (the key pressed doesn't matter in this example as we are merely interested in firing the event).

Figure 4.21

After pressing the *ctrl* key the BackColor property of txtOnKeyUpDown changes to vbGreen (Figure 4.21).

Figure 4.22

After releasing the *ctrl* key the BackColor property of txtOnKeyUpDown changes to vbYellow (Figure 4.22).

If you press and hold a key, it will repeatedly fire the OnKeyDown event (along with the OnKeyPress)event.

OnKeyPress

The OnKeyPress event is very similar to the OnKeyDown event with the main exception being that the key that is pressed must return a character. In the examples illustrated in Figures 4.21 and 4.22, pressing the *ctrl* key would *not* trigger the onKeyPress event.

If you click into txtOnKeyPress and start tapping keys you will notice that txtOnKeyPressCounter increments by 1 (until it reaches 100) (if the key pressed returns a charcter).

Form Events - OnOpen, OnLoad, OnResize, OnActivate, OnUnload, OnDeactivate and OnClose

Opening a Form

There are two types of form specific events, the first being those associated with the graphical user interface, and the second associated with data and recordsets.

When a form is opened or closed there are a number of stages which a form goes through in order to be capable of displaying itself. The following are the states and each has an associated event:

When the form is opening: **Open →Load →Resize →Activate →Current**

When the form is closing: **Unload →Deactivate →Close**

The Open Event is the first to be fired. In this event you can check whether data exists in the database for the form to work with, and if it doesn't you can Cancel = True to prevent the form from opening.

The Load Event is significantly different from the Open Event in that it cannot be cancelled.

The Resize Event deals with positioning of controls on the form. It is also called whenever the form is minimised, resized, moved, maximised or restored.

The Activate Event is associated with the GetFocus event except Activation is to windows (forms, reports and dialog boxes) what focus is to controls. You may want your code to refresh its view of the recordset in case any data has been updated since it was last active.

The Current Event occurs when the form is ready and retrieves data from the underlying recordset. This event is also the first step the form takes in its efforts to handle recordset data.

Please open up frmStudentsDataEntry. We will be using the immediate window to help us ascertain the correct order of events . To open the immediate window you:

- Click on the view drop-down box

Figure 4.23

- Choose Immediate Window

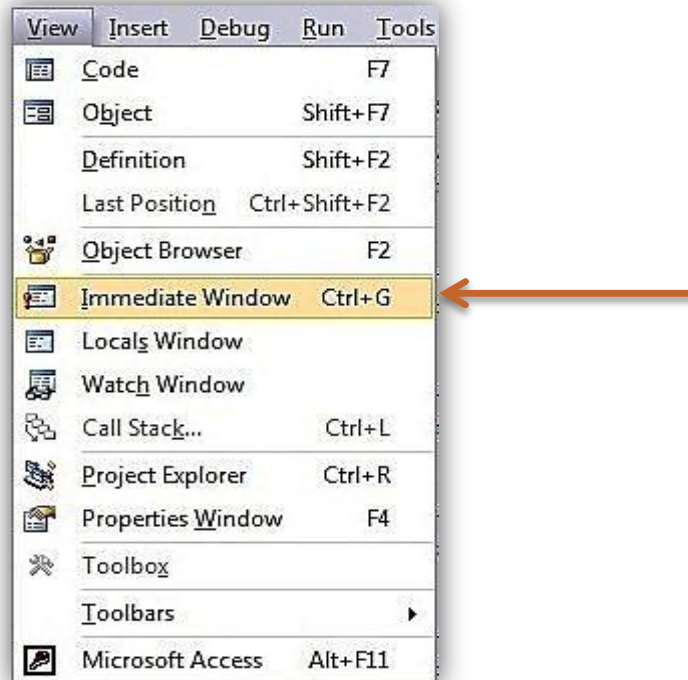


Figure 4.24

- It should be visible at the bottom of your screen (the immediate window can be docked in many different places but it is typical to have it docked below the code window)

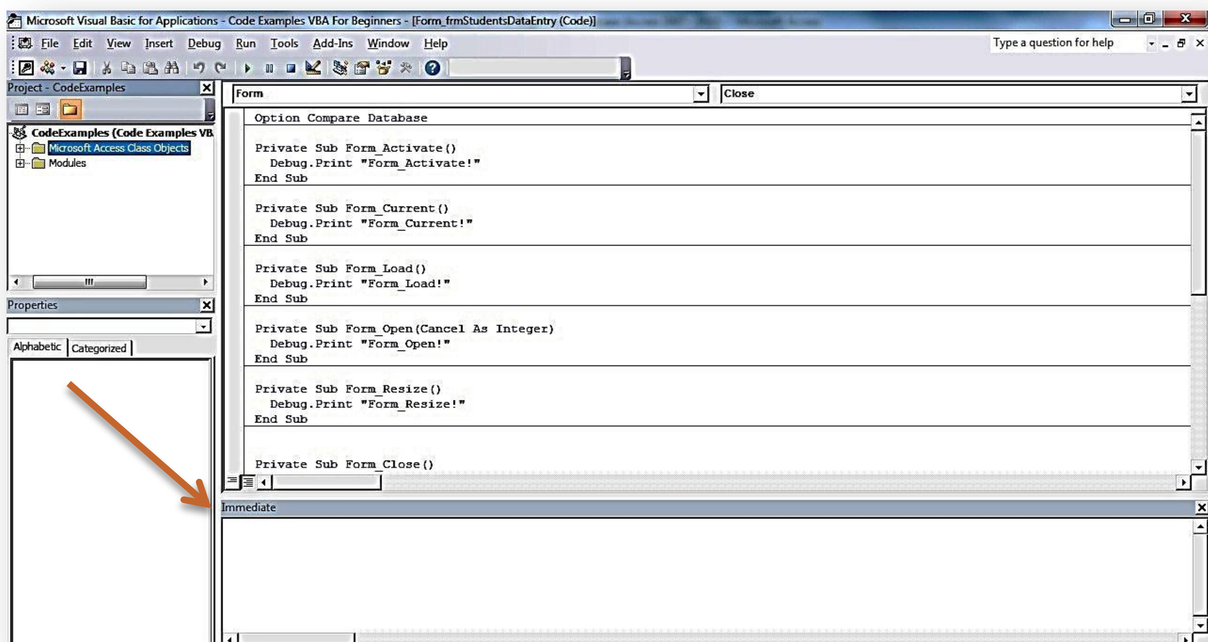


Figure 4.25

Note: *The immediate window is a tool that can be used for debugging purposes and to call sub procedures and functions. We will be discussing the immediate window in much more detail in a later unit. For now, you just need to know that when you write Debug.Print in a subprocedure or function, whatever follows will be printed to the immediate window. Ergo, Debug.Print "Form_Activate!"will print Form_Activate! In the immediate window. We will be using this technique to demonstrate the order in which form events are fired.*

Here is the code for all the events associated with the opening of a form. If you select Form_frmStudentsDataEntry from the Object Explorer (window in top right of screen) of the VBA editor you will see this code.

```
1     Option Compare Database
2
3     Private Sub Form_Activate()
4       Debug.Print "Form_Activate!"
5     End Sub
6
7     Private Sub Form_Current()
8       Debug.Print "Form_Current!"
9     End Sub
10
11    Private Sub Form_Load()
12      Debug.Print "Form_Load!"
13    End Sub
14
15    Private Sub Form_Open(Cancel As Integer)
16      Debug.Print "Form_Open!"
17    End Sub
18
19    Private Sub Form_Resize()
20      Debug.Print "Form_Resize!"
21    End Sub
22
```

Figure 4.26

Opening a form we see the order in which this series of events prints to the immediate window.

Figure 4.27

Closing a Form

Closing a form has fewer events than opening a form but is equally structured. Just to remind us: When the form is closing: **Unload →Deactivate →Close**

The Unload Event (and the load event) is Cancellable. Setting Cancel = True will prevent the form from being closed. This is very useful when users haven't saved their data and you wish for them to confirm that the changes are desired.

The Deactivate Event is the window equivalent of LostFocus. One cannot do anything about it but one could save data to the database which hasn't been committed.

The Close Event is a form and report object function. At this stage the object will be deleted once the event has finished.

```
1     Option Compare Database
2
3     Private Sub Form_Close()
4       Debug.Print "Form_Close!"
5     End Sub
6
7     Private Sub Form_Deactivate()
8       Debug.Print "Form_Deactivate!"
9     End Sub
10
11    Private Sub Form_Unload(Cancel As Integer)
12      Debug.Print "Form_Unload!"
13    End Sub
14
```

Figure 4.28

After closing the form, the immediate window will look like this (I have removed the printouts from the opening of the form):

Figure 4.29

Cancel Form_Close Event

<table>
<tr><td colspan="2">

```
1      Cancel = True
2      Debug.Print "Form_Unload!"
```

</td></tr>
<tr><td>Insert the above code into your form to test out the Cancel Unload operation</td><td>Form_Unload!</td></tr>
</table>

Figure 4.30

Recordset Control Events - OnCurrent, BeforeUpdate, AfterUpdate, OnChange

Data in a form is stored in the form's recordset property. All these events are associated with the interaction between the form and this underlying Recordset object.

The Current Event occurs when data in a form or report is refreshed. It typically fires when the active record on a bound form is changed.

The Before Update Event executes *just before the form changes are saved to the database*. This can be seen as an application implementation of update and insert triggers. Here you would carry out any final data validations, check business rules, populate hidden fields, and cancel the action altogether. As Access doesn't implement triggers (as that is a job for the Jet engine or other data source) this is probably the place where final validation checks should be done.

The After Update Event executes once the data has been committed to the database. Useful for updating other tables like audit trails, updating graphics to indicate a save, disable fields from being changed, close and open up a View type form.

The Change Event executes when data within a text object's content is changed and before the Before Update and After Update Events. This means you can validate the content of the control before it loses focus and before its data is committed to the database. If the Form is bound to a recordset, then changing focus from a *changed* text control to another control will automatically attempt to commit the *change* to the field / record in the database.

Using frmStudentsDataEntry cycle through the records and every time you change a record you will see Form_Current! being printed in the immediate window.

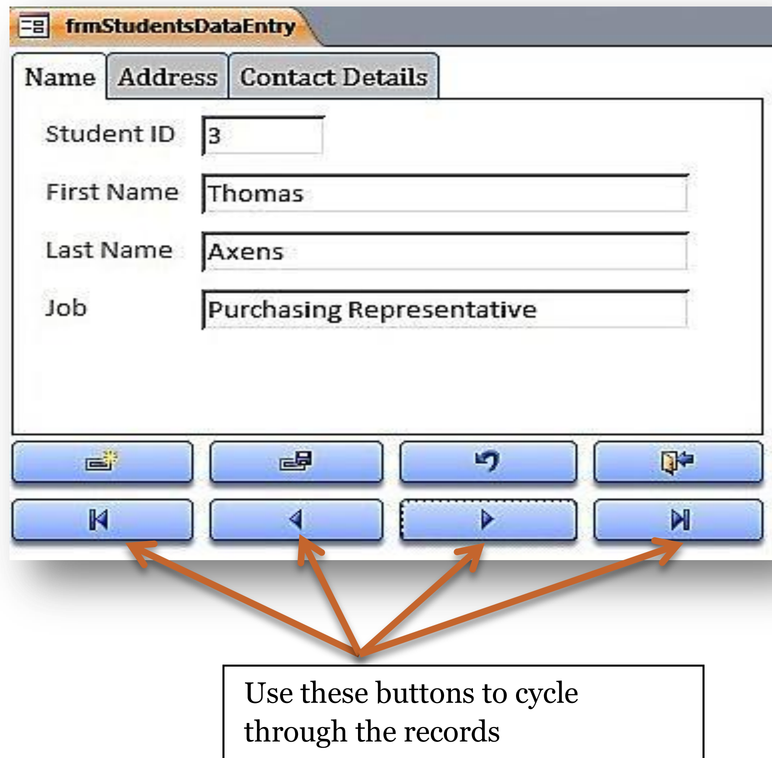


Figure 4.30

```
1    Option Compare Database
2    Private Sub Form_Current()
3      Debug.Print "Form_Current!"
4    End Sub
5
6    Private Sub Form_AfterUpdate()
7      Debug.Print "Form_AfterUpdate!"
8      Msgbox “Data change saved!”
9    End Sub
10
11   Private Sub Form_BeforeUpdate(Cancel As Integer)
12     Debug.Print "Form_BeforeUpdate"
13     If (MsgBox("are you sure", vbYesNo) = vbNo) Then
14       Cancel = True
15       Me.Undo
16     End If
17   End Sub
```

Put the form into form view and cycle back and forth. For each record movement the immediate window will have a Form_Current! Line member	Form_Current! Form_Current!

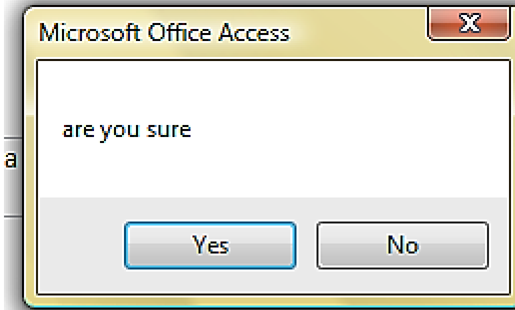
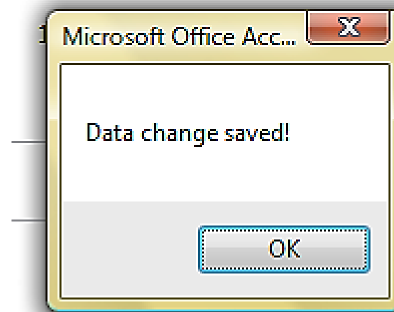
Change the value in the textbox and try to move to the next or previous record. This dialog should appear. The BeforeUpdate routine presents you with this dialog. If you press No the Cancel argument is set to True which forces the form not to update the database and not to progress to the next record. BTW, to cancel any changes press ESC and you'll be able to navigate again.	 Form_BeforeUpdate!
This time allow the changes to be saved. This will fire the After update event and display this message.	 Form_AfterUpdate!

Figure 4.31

OnTimer Events

The Timer Event is a special form event that is activated after a set period of time. The exact time of the event is at least the value of the Timer Interval property.

Open frmTimer to see the event at work. You should see a speedboat speeding across an ocean.

Figure 4.32

The code that goes behind the form is this:

```
1     Option Compare Database
2     Dim intCounter As Integer
3     Private Sub Form_Load()
4         Me.imgSpeedboat.Top = 2750
5         intCounter = 12500
6       Me.TimerInterval = 100
7     End Sub
8
9     Private Sub Form_Timer()
10    intCounter = intCounter - 500
11    If intCounter < 200 Then
12        intCounter = 12000
13    End If
14      Me.imgSpeedboat.Left = intCounter
15    End Sub
```

Figure 4.33

Although the code in figure 4.32 may look complicated it is actually fairly simple. Essentially every 1/10 of a second (`Me.TimerInterval = 100`) the Form-Timer() sub procedure is fired. And every time the the Form-Timer() subprocedure is fired the image of the speedboat is moved 500 twips to the left (a twip is a unit of measurement in Access. 1440 twips = 1 inch). And when there is no more left left (so to speak) the image is moved to 12500 twips from the left. And the whole thing repeats ad infinitum.

Questions

1. What should be written in the On Current field of the property sheet to indicate that an event procedure exists for the On Current event?
2. When is the OnMouseUp event triggered?
3. Will the OnMouseDown event fire if you right-click a mouse?
4. If you tab from txtFocus1 to txtFocus2, which event fires first? The OnLostFocus event or the OnGotFocus event.
5. Look at this code:

```
1   Private Sub txtOnMouseUpDown_MouseDown(Button As Integer, Shift As
2   Integer, X As Single, Y As Single)
3   Me.txtOnMouseUpDown.BackColor = vbRed
4   End Sub
```

Figure 4.34

True or False: The argument Button (highlighted in red) refers to the button or text-box clicked on a form.

6. In the above code snippet what do the buttons X and Y represent?
7. What causes the OnMouseMove event to fire?
8. Look at this code:

```
1   Private Sub txtOnKeyPress_KeyPress(KeyAscii As Integer)
2   MsgBox "You have pressed a key!"
3   End Sub
```

Figure 4.35

If txtOnKeyPress had the focus, what would happen if we pressed the *ctrl* key?

9. These are the 5 events associated with opening a form:

Activate
Load
Current
Resize
Open
In what order are these events executed when a form opens?

10. In what order should these will these events associated with closing a form be fired?

Close
Unload
Deactivate

11. Although similar in nature, what is the difference between the Activate event and the OnGotFocus event?
12. When does the BeforeUpdate event fire?

Answers - Events

1. [Event Procedure]
2. When a depressed mouse button is released.
3. Yes.
4. The OnLostFocus event.
5. False: it refers to which mouse button was pressed.
6. The Coordinates of the mouse curser.
7. Hovering the curser over an object that has an event procedure for OnMouseMove.
8. Nothing. The onKeyPress event is only triggered by keys that return characters.
9. **Open →Load →Resize →Activate →Current**
10. **Unload →Deactivate →Close**
11. The activate event fires when a window (such as a form, report or dialog box) receives the focus whilst the OnGotFocus event fires when a control receives the focus.
12. Just before committing changes to the server.