Simply – Visual Basic® for Microsoft ® Access

Microsoft® Access Made Simple

Thank you for subscribing to the Simply-Visual Basic® for Microsoft ® Access Newsletter.

This Newsletter is a product of the www.simply-access.com website,

The Newsletters are a series of lessons on Visual Basic® for Applications development system. The lessons are easy to understand with numerous screen shots and detailed explanation of the Visual Basic® for Applications development system code.

If you have no idea about the Visual Basic® for Applications development system this is the place to start. These lessons will give you a firm grounding to expand your wings when building your Microsoft® Access Databases.

Disclaimer

Whilst every effort is made to test the code contained within these lessons, the code provided by this Simply-Visual Basic® for Microsoft® Access 2000 Newsletter is for demonstration purposes only. Using the code in your projects is entirely at your own risk.

Acknowledgements

Symbols
® = Registered trademark or service mark
TM = Trademark ownership claimed
© = Copyright ownership claimed

Microsoft ® Visual Basic® and Windows ® are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Screen shot(s) reprinted by permission from Microsoft ® Corporation.

References to Microsoft ® Corporation, its products, trademarks and screen shots are in accordance with their guidelines, www.microsoft.com/permission and http://www.microsoft.com/trademarks.

Copyright

Guarantee

If at any stage these Newsletters are not meeting your expectations, or you have simply lost interest in learning VBA, then you can cancel your subscription and the remainder of your outstanding subscription amount will be returned to you.

Table of Contents

Newsletter Number Nine

Welcome

This week is quite a lot of theory on the very important aspect of coding known as Events. The Events topic will take up a few Newsletters as there is quite a bit of information to cover. There will be very little coding. You can either read through these Newsletters on Events as you receive them, or keep them in a handy spot to refer to when you start to write some of your own code and are not sure of where the best place to put the code would be, or to refer back to them when we utilise the Events in subsequent editions of the Newsletter. These few Newsletters on Events are something you are likely to refer to frequently. Have a quick read now so you know what is enclosed in them, but do not get too stressed with remembering all the information. These series of Newsletters on Events are more of a resource to refer to when you need them.

Once again, if you need any help with any part of the tutorials – maybe I have not been as clear as I should have been, then drop me a line outlining what the problem is.

mailto:helpwithVBA@simply-access.com

Objective

This week's objective is to develop an in-depth understanding of the world of Events (at least start to), and the importance they have in Visual Basic® for Applications development system coding language. Microsoft® Access is an event driven program. This means a course of action (code) is set in motion when an Event occurs.

I will first refresh your memory with a definition of an Event, and then we will have a look at form, control, and report events in more detail.

Events

Definition.

Events are things that happen to objects, such as the clicking of a command button, or the opening and closing of a form, etc. These events can trigger an action to be carried out. This action is in the form of a Macro or an Event Procedure. In this Newsletter we will be concerning ourselves with Event Procedures only.

You will often see events written with the 'On' Keyword, such as 'On Open', 'On Current', etc.

Tip No: 20 When looking for Help with the many different Events, leave the Keyword 'On' out of the search and include the word 'Event', i.e. type in' Open Event' instead of 'On Open'. You will get a much better result. Also do these searches in the VBE window version of help.

Form Events

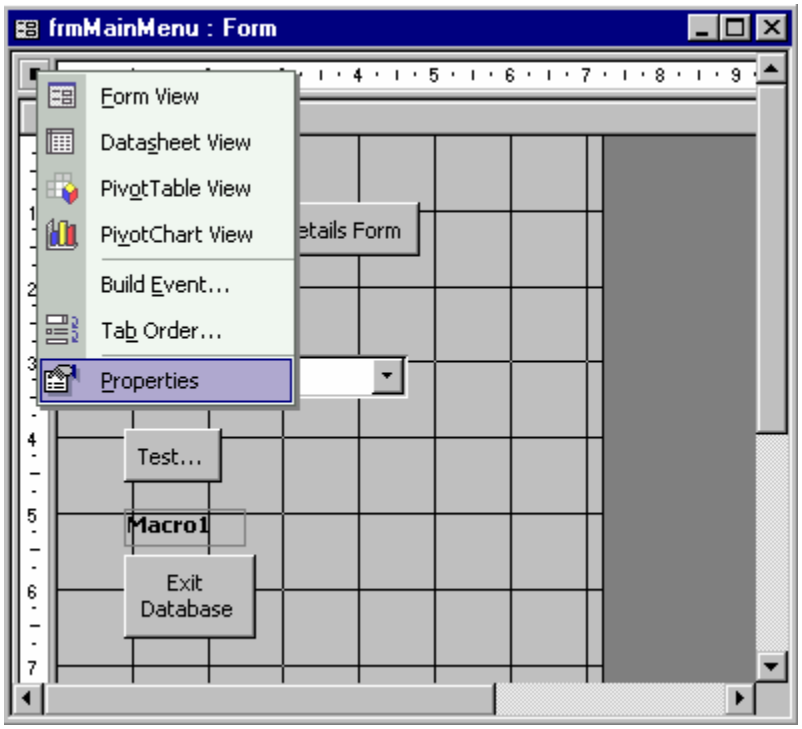
The first type of Event we are going to look at is the 'Form Event'. A 'Form Event' is an Event that can occur with a form. Events occur for forms when you open or close a form, move between forms, or work with data on a form. We will look at each of these in turn.

During past Newsletters we have looked at Events briefly. We have previously set code to run for the 'On Load' Events for both the forms 'frmMainMenu' and 'frmPersonalDetails' when we utilised the Move and SetFocus methods in Newsletter Number 8.

An easy way to view the available Events for a Form (Control or Report) is to open the properties window of the Form in question. Let's do this:

- Open the Database, VBADatabase8 (sent with the previous Newsletter);
- With the Main Window open, select Forms and open Form 'frmMainMenu', in Design View;
- Open the Properties Window of the form by right clicking in the upper left hand corner of the form, where the two rulers meet, and selecting 'Properties' (left click) from the list displayed (Figure 9.1);

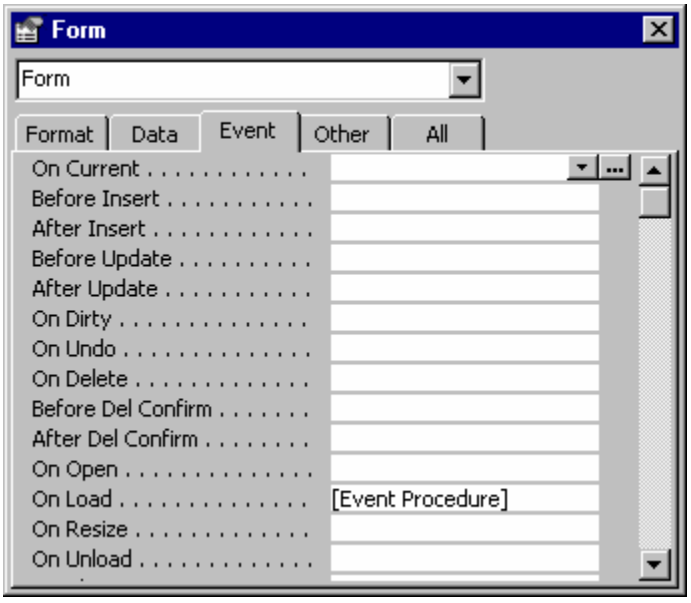
Figure 9.1
Opening the Properties Window.



- With the Properties Window open, select the Event Tab (Figure 9.2).

Note: The [Event Procedure] assigned to the 'On Load' event.

Figure 9.2
Properties Window with the Event tab selected.



This will also be the Window where you will often assign your [Event Procedures].

With Events, it is important to remember that Events occur in order and the attachment of your code to a particular event will set when that code is run. Let's explore this a bit further: If you open a Form, the following Events occur in the following order:

Open ➲ Load ➲ Resize ➲ Activate ➲ (GotFocus) ➲ Current

If there isn't an active Control on a Form, the **GotFocus** event will occur. If there is an Active Control, the **GotFocus** Event will **not** occur. This is because the Focus is on the form rather than a Control, and therefore this Event can occur.

We have used the opening of a form as an example, we will now explore this action in more detail.

Open Form Events.

As mentioned above, the Events that occur when you open forms are the following:

Open ➲ Load ➲ Resize ➲ Activate ➲ (GotFocus) ➲ Current

On Open Event

The Open event occurs when a form is opened, but before the first record is displayed. Therefore, attach code here which you wish to run as soon as the form is opened. The Open event will not occur when you activate (move to a form) a form that's already open, i.e. if you open a second form from the first form, then close the second form, the first form's On Open Event will not occur as the first form has not been closed and opened, it has just been hidden behind the second form. (If you wish an action to occur when the first form is reactivated use, the On Activate Event instead.)

If the Form is based on a Query, the Query is run, prior to the On Open Event.

A few examples of actions you may wish to code into the On Open Event are:

- Cancelling the opening of a form if there are no records to display (you can use the On Open Event to do this, but not the On Load Event);
- Opening other forms, and then hiding them, so they are displayed quicker when the command button of the form is pressed to open these other forms;

- To size a form.

The On Open Event occurs immediately before the On Load event.

On Load Event

Whereas the on Open Event occurs when the form is opened and before the first record is displayed, the On Load Event occurs when the first record is displayed.

A few examples of actions you may wish to code into the On Load Event are:

- Setting the default values of Controls;
- Displaying of calculated data that depends on the values in the record;
- Use of the 'OpenArgs' property (the 'Open Args' property will be discussed in detail in a later Newsletter);
- Set the focus to a specific control (you did this in Newsletter Number 8);
- To size a form.

On Resize Event

The Resize Event occurs when a form is opened or whenever the form's size changes. A form's size may change with the use of the Move Method (as in Newsletter Number 8), or the MoveSize method of the DoCmd.

An example of an action you may wish to code into the On Resize Event is:

- Move or size a control when the form is resized;

On Activate Event

The Activate event occurs when a form receives the focus and becomes the active window. You make a form active by:

- Opening it;
- Clicking on form with your mouse, or a clicking a control on the form;
- Or by invoking the **SetFocus** method.

The On Activate Event can only occur if the form is visible. If the form is not visible an error will occur.

A few examples of actions you may wish to code into the On Activate Event are:

- Maximising the form;
- Assigning custom toolbars;
- The Requery of a Subform to update the data in the Subform;
- Changing the attributes of a form when it becomes Active.

On GotFocus Event

This rarely used event only occurs when the form gets focus, but only if there are **no** visible, enabled controls on the form.

On Current Event

The Current event occurs when the focus moves to a record making it the current record, or when the Form is Refreshed or Requeried. (The difference between Refreshed and Requeried will be covered in a later edition of the Newsletter.) This event occurs when a form is opened, whenever the focus leaves one record and moves to another, and when the Form's underlying Table or Query is requeried. This event is one of the more commonly used Events. If you wish to run code whenever a record is displayed, this is the place to put it.

A few examples of actions you may wish to code into the On Current Event are:

- Display a message, depending on the data in the current record;
- Synchronise the data on another form to match the current record of the active form. An example of this may be that when you display a department in your organisation, the corresponding form may display all the employees who work in that department;
- Change the colour of a background of a field, depending on a condition;
- Make a field visible or invisible, depending on the value of that field or another field;

- Control whether the form can be edited or not (read only);
- Clear unbound combo boxes.

This concludes the look at the Events which occur when you open a form. Next we are going to have a look at the Events which occur when you close a form.

Close Form Events.

The order of Events which occur when a user closes a form, are the following:

Unload ➲ (LostFocus) ➲ Deactivate ➲ Close

If there isn't an active Control on a Form, the **LostFocus** event will occur. If there is an Active Control, the **LostFocus** Event will **not** occur. This is because there is a focus on the Form rather than a Control, and therefore this Event can occur.

On Unload Event

The Unload Event occurs when the form is closed, but before the form is removed from the screen. You can cancel the Unload Event, but not the Close Event.

The Unload Event can be triggered by the following actions:

- Closing the form by any method; and
- Quitting an application.

A few examples of actions you may wish to code into the On Unload Event are:

- To check if the user actually wishes to close the form;
- Displaying a Dialog Form for the user to enter information, such as the name of the person who has just accessed the form;
- To check if the user wishes to save the record prior to unloading, thus giving the user the choice not to save changes.

On LostFocus Event

This rarely used event only occurs when the form loses focus, but only if there are **no** visible, enabled controls on the form.

On Deactivate Event

The Deactivate event is triggered when a form loses focus. The focus can be lost to a Table, Query, Report, Macro, Module window, Database window, or to another Form. The Deactivate Event does not occur if a dialog or pop up form is opened, or if another application is commenced.

If you are switching between two open forms, then the Deactivate event occurs for the form with the original focus, and the Activate event occurs for the form receiving the focus. If the form has no visible enabled controls, then the LostFocus and GotFocus events will occur.

An example of an action you may wish to code into the On Deactivate Event is:

- To return the Form to its previous state if the attributes of the Form were changed when the form was opened.

On Close Event

The On Close Event occurs when the form is closed and is removed from the screen; therefore it occurs after the Unload Event. Remember: The Unload Event can be cancelled, but not the Close Event.

Two examples of actions you may wish to code into the On Unload Event are:

- Check to see if another form is open (loaded) then close this Form when the current Form is closed;
- Reopen the Main Menu Form.

Moving Between Forms Events.

When you move from one form to another, two events are triggered in the following order:

Deactivate (Form losing focus) ➲ **Activate** (Form getting focus)
Both these Events were covered previously in the Open Form Events and the Close Form Events.

Events Associated with moving between Form Controls.

When you first open a Form, the following Events occur:

Open ➲ Load ➲ Resize ➲ Activate ➲ (GotFocus) ➲ Current

These are Form Events, but there are also Control Events that occur after the Current Event. These are:

Enter ➲ GotFocus (Control)

When you move the focus from one to Control to another, the following Events occur for the Control that is losing the focus;

Exit ➲ LostFocus (Control)

On Enter Event

The Enter Event is triggered before a control receives the focus from a control on the same form, or when the form is opened. The Enter Event applies to text boxes, option groups, combo boxes, list boxes, command buttons, object frames, and subforms.

Note: The Enter Event does **not** occur when the original form receives the focus for the second time, i.e. if the focus moves to a second form then back to the original form, the Enter Event does not occur because the control already had focus in the original form.

Two examples of actions you may wish to code into the On Enter Event are:

- Check to see if the required data has been entered in the main form when moving (setting focus) to a subform;
- Display a message box on how the Control is used and what sort of data it contains.

On GotFocus Event (Control)

The GotFocus Event for a Control is triggered when the control receives the focus. The GotFocus Event applies to text boxes, option groups, combo boxes, list boxes, command buttons, object frames, and subforms. The GotFocus Event occurs every time the Control has the focus, whereas the On Enter Event only occurs the first time it receives the focus i.e. the Enter Event does **not** occur when the original form receives the focus for the second time. I.e. if the focus moves to a second Form then back to the original form, the Enter Event does not occur because the control already had focus in the original form, but the GotFocus Event would occur. It is important to remember these points when it comes to writing your code. Putting your code in a slightly different spot can cause the code to be triggered or not.

The GotFocus Event for a Control is triggered by the following actions:

- Pressing the TAB key to move the focus to the next Control in the TAB order; or
- Using the following methods or actions: Set Focus; SelectObject, GoToRecord and GoToControl in VBA coding.

Two examples of actions you may wish to code into the On GotFocus Event for Controls are:

- Change the background colour or text of the control box that has the focus;
- Check to see if the value of the Control that has the focus, or if the Control that has the focus, has any data at all (Null) and carry out actions depending on the result.

On Exit Event

The Exit event occurs just before a control loses the focus to another control on the same form. The Exit Event applies to text boxes, option groups, combo boxes, list boxes, command buttons, object frames, and subforms. It occurs just before the LostFocus Event. The Exit Event does not occur when the form loses focus, only when the control loses focus. If a form was to lose focus and regain focus, then the control that originally had the focus would still have the focus. Therefore, the Exit Event is not triggered as the focus has not changed to another Control.

An example of an action you may wish to code into the On Exit Event is:

- Change the background colour or text of the control box that does not have the focus.

On LostFocus Event (Control)

The LostFocus Event for a Control is triggered when the focus moves away from the Control that has the focus. The LostFocus Event applies to text boxes, option groups, toggle buttons, combo boxes, list boxes and command buttons.

The LostFocus Event for a Control is triggered by the following actions:

- Pressing the TAB key to move the focus to the next Control in the TAB order; or
- Using the following methods or actions: Set Focus; SelectObject, GoToRecord, and GoToControl in VBA coding.

The LostFocus event is different to the Exit Event in that the LostFocus Event occurs every time a control loses the focus, even when the focus moves to another form or window.

An example of an action you may wish to code into the On LostFocus Event is:

- Change the background colour or text of the control box that does not have the focus.

Events Associated with changing data on a Form.

So far we have covered what happens when the focus moves from one control to another with the Enter and GotFocus Events and the Exit and GotFocus Events. This is okay if the data is not changed, but what happens if you wish to activate code if the data in a Control in a Form has been altered? There are events for this scenario; they occur before the Exit and Got Focus Events. Therefore, if the data in a Control on a form was changed, the following Events would take place:

BeforeUpdate ➲ AfterUpdate ➲ Exit ➲ LostFocus

There is an even more sensitive action the 'Change' event, this occurs if the Text in a Text Box changes or if the Text, in the Text section of the Combo Box changes, but before you move to another control (therefore this event occurs before the BeforeUpdate Event). The Change Event occurs along with some other Events as per the sequence below:

KeyDown ➲ KeyPress ➲ Dirty ➲ Change ➲ KeyUp

This means for every key you press, the above list of Events are triggered.

BeforeUpdate Event (Control)

The BeforeUpdate event is triggered before changed data in a control is updated (saved). The BeforeUpdate Event applies to text boxes, combo boxes, list boxes, option groups, and bound Object frames.

Note: Changing the value of a control using VBA or a Macro, or in a calculated control will not trigger the BeforeUpdate Event. The data has to be changed manually, i.e. by the use of a keyboard. If you wish to trigger an Event which would detect a change in the value of a Control which has been altered using VBA or a Macro or a calculated control, then use the BeforeUpdate Event of the form itself.

Two examples of actions you may wish to code into the BeforeUpdate Event for Controls are:

- Check to see if the data from the control has been removed (Null) and check whether the user wishes to save this change.If the user chooses not to accept the Null value, the original value will be reinstated into the Control;
- Display different error messages for different values entered (if a validation rule is not met).

BeforeUpdate Event (Form)

Like the Control, there is also a BeforeUpdate Event that can be triggered on a Form. The BeforeUpdate event is triggered when you update (save) a record. The process of moving from one record to another causes the underlying record to be updated (saved), thus triggering the BeforeUpdate Event. The BeforeUpdate Event for a form is triggered only if data has changed in one or more controls on the form.

An example of an action you may wish to code into the BeforeUpdate Event for Forms is:

- To cancel updating of a record before moving to another record. This could because of a validation error or another reason.

AfterUpdate Event (Control)

The AfterUpdate event is triggered after changed data in a control is updated (saved). The AfterUpdate Event applies to text boxes, combo boxes, list boxes, option groups, and bound Object frames.

The following are examples of actions you may wish to code into the AfterUpdate Event for Controls:

- To Requery subforms so they display the correct record when a value in a text box has been altered (updated);
- To make another control visible or enabled depending upon the value in the control being updated;
- Update other fields depending on the value in the control being updated. For example, you could enter a Post Code and have the Town, State and Country entered automatically by using the AfterUpdate Event;
- Perform an action depending on the value chosen in an Option Group. This is particularly useful if you have a series of reports and you wish the user to be able to choose which report to preview from an Option Group;
- To change the value of controls in other open Forms;
- To find a record matching the selection chosen from a combo box.

AfterUpdate Event (Form)

The AfterUpdate Event of a Form is triggered when the changed data in a record is updated (saved).

A few examples of actions you may wish to code into the AfterUpdate Event for Forms are:

- Use a message box to confirm to users that their record has been saved when moving to another record or entering a new record;
- To log changes on a form;

- To Requery related Combo Boxes in other open/loaded Forms.

On KeyDown Event (Control)

The KeyDown Event for a Control is triggered when the user presses a key on the keyboard while a control has focus. The Event occurs repeatedly until the key is released. It can be cancelled by setting the KeyCode equal to Zero. The KeyDown Event can also be triggered by the use of a Macro or VBA code (SendKeys).

The most common key strokes used in combination with the KeyDown Event are:

- Function keys;
- Navigation keys;
- Combination of keys;
- The numeric keypad or the keyboard number keys.

The KeyDown event does not occur if you press:

- The ENTER key, if the form has a command button for which the Default property is set to Yes;
- The ESC key, if the form has a command button for which the Cancel property is set to Yes.

On KeyDown Event (Forms)

The KeyDown Event for a Form is triggered when the user presses a key on the keyboard; this can only occur for a Form, if there are **no** Controls on the Form or if all the visible Controls on the Form are disabled.

You can get around this by setting the KeyPreview property of the form to Yes, and then the KeyDown Event for forms will occur prior to the KeyDown Event for the Control that has the focus. This is handy if you wish a certain key stroke, such as Ctrl & G, to have the same action no matter where in the form it is entered.

Two examples of actions you may wish to code into the KeyDown Event for Forms or Controls are:

- To set up short cut keys for certain actions to be performed;

- Prevents the user from doing such items as using the Page Up or Page Down Keys.

On KeyPress Event (Controls/Forms)

The KeyPress Event occurs when the user presses and then releases a single key or key combination. These single key and key combinations correspond to the ANSI Code for the Event to occur. Once again, the KeyPress Event for a form will only occur if there are **no** controls on the form, or all visible controls are disabled, or if the forms KeyView property is set to Yes. It can be cancelled by setting the KeyCode to Zero.

On Dirty Event (Controls/Form)

The Dirty Event is triggered if:

- The contents of a form have been altered;
- If text has been entered into the text portion of a combo box;
- If the control's Text Property has been changed by the use of a Macro or VBA; or
- Moving from one page to another in a Tab control.

Two examples of actions you may wish to code into the Dirty Event for Forms or Controls are:

- To cancel saving the record;
- To disable command buttons and navigation buttons, whilst the form is in a Dirty state.

On Change Event (Controls/Form)

The change event is triggered in a similar fashion to the Dirty Event:

- The contents of a form have been altered;
- If text has been entered into the text portion of a combo box;
- An item is selected from a combo box list;
- If the control's Text Property has been changed by the use of a Macro or VBA; or
- Moving from one page to another in a Tab control.

You would use this event if you wanted to trap for something, as a user enters the data character by character.

On KeyUp Event (Controls/Forms)

If you hold the key or key combination down, the KeyDown and KeyPress Events will occur repeatedly until released, and then the KeyUp Event occurs. The KeyUp Event also occurs when you send a keystroke command via a Macro or VBA. The KeyUp Event for a form will only occur if there are **no** controls on the form, or all visible controls are disabled, or if the forms KeyView property is set to Yes.

Note: The KeyUp Event only occurs after any action caused by pressing or sending a key is complete. Therefore, if the action causes the focus to move to another control, the action triggered by the KeyUp Event occurs for the second control.

Conclusion

This concludes this week's Newsletter. As you can see there is a lot involved with Events. We will continue to work through this important aspect of coding next week. I hope you have not found this week too involved and you able to grasp something of the order of Events and what sort of coding you would add to a certain Event. We have started to look at Form and Control Events this week. Next week we will conclude this and the following week we will look at Events which occur for a Report.