VBEVENTS ONLINE HELP

An Overview

Using VBEVENTS to Test Event Chains

Introduction

In Visual Basic 3.0 Professional Edition, there are 36 unique events. VBEVENTS is a simple, test program that demonstrates a few of the unusual (and often paradoxical) behaviors exhibited when the Change, Click, GotFocus, and LostFocus events interact. Several test cases are provided to show how both the order and recognition of these particular events is affected by four, primary types of controls: (1) combo box, (2) push button, (3) text box, and (4) vertical scroll bar. Test cases demonstrating the affect of the MsgBox statement are also provided.

One of the most important tasks that must be done in a professional application is data validation; so, it is important to understand the ways in which the Change, GotFocus, and LostFocus events interact with other Visual Basic language constructs and events, because some types of data validation can only be done with the Change and GotFocus events. In general, LostFocus is essentially useless for some types of data validation, and VBEVENTS clearly demonstrates why this is the case.

This online help file for VBEVENTS describes four tests that can be performed and observed with the program. There are other combinations of test, but they are left as an exercise.

The Quantum Nature of Visual Basic Events

When discussing Visual Basic events, it is important to consider particular groups of events rather than to concentrate on individual events. In this respect, an event chain is a particular group of events; and since event chains exhibit behaviors that are somewhat unusual, transient, and paradoxical, I use the word, quantum, to describe the often peculiar behaviors exhibited by Visual Basic event chains. By using "quantum" to describe Visual Basic events and event chain behaviors, I am not referring to the (perhaps) more traditional definition of "quantum" that is usually found in textbooks on Quantum Mechanics and Q.E.D. Rather, I am using "quantum" to describe behaviors that are somewhat peculiar, unusual, paradoxical, and transient. I am not suggesting that Visual Basic events do not occur exactly as they should, but I am stating that the ways in which Visual Basic events occur and the rules that they follow are very different from most the way most Visual Basic (and Windows) programmers presume Visual Basic events behave.

In other words, from my perspective Visual Basic events follow a clearly defined set of rules; but those rules are very different from the rules that many Visual Basic (and Windows) programmers intuitively think Visual Basic events follow. Because there is such a dramatic difference between the rules that nearly everyone thinks Visual Basic events follow and the actual rules that Visual Basic events follow, I clearly state that Visual Basic events and event chains are very quantum. Events that we (perhaps) believe always occur actually do not always occur, and events that we (perhaps) believe do not occur in fact actually do occur. What is, is not; and what is not, is (at least in some event chains)!

Of course, I am not suggesting that Visual Basic events are either the least bit unpredictable or flawed. Visual Basic events follow a very precise set of rules, and those rules are both very predictable and correct. I call them quantum rules because they are not so intuitive and because, in some cases, they require that an event chain that exists at one moment be modified so that it can exist at another moment. In other words, it is very naive simply to state that event B follows event A, as a general rule. In the quantum world of Visual Basic event processing, it is considerably more likely that at time(0) event B follows event A, but that at time(1) event B is discarded, ignored, and never recognized at the application level because of some requirement necessary to insure that event A is handled correctly.

In a Visual Basic application, there are several ways in which an event can become part of an event chain. Perhaps the most obvious way is as a direct result of user interaction with the application's Graphic User Interface (GUI). For example, when the application's user clicks on a push button, a Click event for the push button is triggered. However, this is not the only way in which the Click event for a push button can be triggered. Within some other event or code procedure, the programmer can remotely trigger a Click event for a particular push button by setting the Value property of the push button to True. Also, within the same form, the particular push button's Click event can be triggered by simply calling the Click event subroutine for that particular push button. Methods can also trigger events (e.g., using the SetFocus method can trigger both GotFocus and LostFocus events, and in some cases, it actually does).

There are different kinds of event chains, just as there are different ways in which events are triggered. Events can be triggered (1) directly (i.e., by the user interacting with the application's GUI), (2) remotely (i.e., by doing something in another event or procedure that explicitly triggers an event [e.g., setting a property or calling an event subroutine]), and (3) indirectly (i.e., by using a method that performs some action that triggers an event). When an application is simple, there is a tendency for event chains also to be simple; but as an application becomes complex, it is likely that event chains will also become complex.

I recommend clearly separating forms from code, and if that rule is followed it produces an application which has procedures that reside in modules. At some point, it is likely that the recognition of an event will transfer control from the form containing that event to a first level, module-resident procedure that does the work necessary successfully to respond to the primary event. In that case, it is also likely that the first level procedure will call some other procedure, and there may be a series of procedure calls necessary to process the primary event. It is also likely that one or more of the procedures in a procedure chain will both remotely and indirectly trigger other events. These other events are called secondary events, and it is very important to notice that the primary event has not completed its work until

all of the secondary events complete their work. In turn, each of these secondary events can be considered to be a primary event for its particular branch of a complex, event chain; and in that respect, none of those secondary events (which are primary for their particular event chain branches) have completed until all of their secondary events have completed, and so forth.

It is also important to observe that an event can be handled in such a way that the effect of the work done in the event is to make the event not occur. For example, consider the KeyPress event that is directly triggered when the application's user presses the "a" key on the keyboard. In that case, Windows notifies Visual Basic that the "a" key was pressed, and Visual Basic then notifies the application that the "a" key was pressed by triggering a KeyPress event and setting the KeyAscii parameter to the ASCII value of the "a" character. If the code in the KeyPress event sets KeyAscii to 0 before it completes, the effect is that (insofar as the application is concerned) the "a" key was never pressed. In other words, the application can choose to ignore the "a" key, even though the application's user did press the "a" key. This is a simple example of one of the ways in which an event can be made not to occur. Of course, the event did actually occur, but the effect in the program is as though the event did not occur.

While it is easy to say that clicking on a push button directly triggers the Click event for that push button, looking at the event chain that begins with that push button's Click event as being a simple is probably very naive. In a sophisticated application, it is considerably more probable that the code for the push button's Click event will call a procedure that itself both remotely and indirectly triggers events and calls other procedures that also both remotely and indirectly trigger events, and so forth for perhaps several more levels. The actual event chain that begins with the push button's directly triggered, Click event then becomes very complex, and as it becomes increasingly complex with each additional procedure call, with each additional remotely or indirectly triggered event, the rules that control the handling of the event chain, the event chain branches, and the events, themselves, become somewhat quantum in the sense that what our intuition suggests should occur does not always correspond actually to what does, in fact, occur.

Nevertheless, nothing prevents us from learning the real rules so that our intuition can become accurate and so that what appeared to be quantum event behaviors are transformed into very logical, event behaviors. To that effect, VBEVENTS has been designed to demonstrate some of the most simple, logical, event behaviors that appear to be quantum when we do not understand the VB event model.

Test 1: The Simple Case (Everything is Wonderful!)

In this first test case, the easiest way to begin is to start VBEVENTS, because there are several possible test cases and the program is coded to begin with everything set correctly for the first test case. Once you have started VBEVENTS, perform the following steps and observe the order and recognition of events in the Trace List. When VBEVENTS begins, focus is automatically set to the text box control.

1. Using the mouse, click on the vertical scroll bar in the grey area between the thumbnail and the up arrow. The scroll bar area across from the label showing the current value of the scroll bar is a good location. The Trace List should now contain four items in the following order:

   (1) txtText-> GotFocus
   (2) vsrScroll-> Change
   (3) txtText-> LostFocus
   (4) vsrScroll-> GotFocus

2. Using the mouse, click inside the data entry portion of the combo box. The Trace List should now contain six items in the following order:

   (1) txtText-> GotFocus
   (2) vsrScroll-> Change
   (3) txtText-> LostFocus
   (4) vsrScroll-> GotFocus
   (5) vsrScroll-> LostFocus
   (6) clbList-> GotFocus

3. Using the mouse, click inside the text box control. The Trace List should now contain eight items in the following order:

   (1) txtText-> GotFocus
   (2) vsrScroll-> Change
   (3) txtText-> LostFocus
   (4) vsrScroll-> GotFocus
   (5) vsrScroll-> LostFocus
   (6) clbList-> GotFocus
   (7) clbList-> LostFocus
   (8) txtText-> GotFocus

4. For this next step, divide the mouse click action into two parts. In the first part, move the mouse pointer so that it is on top of the "Clear Trace List" push button, and then press down the left mouse button, but do not release the mouse button. Look at the Trace List and observe that it now contains the following items in this order:

   (1) txtText-> GotFocus
   (2) vsrScroll-> Change
   (3) txtText-> LostFocus
   (4) vsrScroll-> GotFocus
   (5) vsrScroll-> LostFocus
   (6) clbList-> GotFocus
   (7) clbList-> LostFocus
   (8) txtText-> GotFocus
   (9) txtText-> LostFocus
   (10) pbtClearTraceList-> GotFocus

5. Now, release the left mouse button. This completes the Click action that began with step 4. Look at the Trace List and observe that it has been cleared, and that the first item in the list is now the Click event for the "Clear Trace List" push button:

   (1) pbtClearTraceList-> Click

If you want to experiment further with this part of the program, then you can tab from control to control,

and you can change values for the text box, combo box, and vertical scroll bar controls. You can press the "Clear Trace List" push button at any time, but do not press or click any of the other push buttons, because doing so will change the test conditions.

As mentioned in the title for this test case, this is the simple case; and everything works wonderfully. Events occur in the order we intuitively think they should occur (once we understand that the Change event is always processed with a higher priority than the GotFocus and LostFocus events). This test case clearly demonstrates that when events contain passive code, nothing unusual happens to the native, event chain. However, this is (perhaps) an idyllic test case, because it is probably never going to appear in a program that actually does some kind of meaningful work.

Test 2: The SetFocus Case (Everything is not so Wonderful!)

In this second test case, the easiest way to begin is to click on the "Clear Trace List" push button. The captions of the other three push buttons should read as follows: "MsgBox Statements ON", "Set Scrollbar Test 1", and "Set Scrollbar Test 2". It is very important that the push buttons have these captions. If the captions are different, then you are in another set of test conditions, and if that is the case you should click on the buttons until they have the correct captions. Note that because of the effect of different test settings (controlled by the push buttons), you may have to click on one or more of the push buttons two times to activate the Click event. This is because some of the other test settings cause the VB event chain to behave differently. Once the push button caption are correct, perform the following steps:

1. Using the mouse, click on the "Set Scrollbar Test 1" push button. Its caption will change to "Clear Scrollbar Test 1", and the Trace List will contain the following items in this order:

   (1) pbtClearTraceList-> Click
   (2) pbtClearTraceList-> LostFocus
   (3) pbtScrollTest1-> GotFocus
   (4) pbtScrollTest1-> Click

2. Before proceeding, observe the value of the vertical scroll bar, and remember it. Using the mouse, click on the vertical scroll bar in the grey area between the thumbnail and the up arrow. The scroll bar area across from the label showing the current value of the scroll bar is a good location. Before doing anything else, observe that the value of the scroll bar changed. The Trace List should now contain nine items in the following order:

   (1) pbtClearTraceList-> Click
   (2) pbtClearTraceList-> LostFocus
   (3) pbtScrollTest1-> GotFocus
   (4) pbtScrollTest1-> Click
   (5) vsrScroll-> Change
   (6) pbtScrollTest1-> LostFocus
   (7) vsrScroll-> GotFocus
   (8) vsrScroll-> LostFocus
   (9) txtText-> GotFocus

If you want to experiment further with this part of the program, then you can tab from control to control, and you can change values for the text box, combo box, and vertical scroll bar controls. You can press the "Clear Trace List" push button at any time, but do not press or click any of the other push buttons, because doing so will change the test conditions.

As mentioned in its title, this is the not so simple test case; and everything does not work wonderfully. What happens in this test is that clicking on the "Set Scrollbar Test 1" push button activates code in the Change event of the scroll bar. This newly activated code does a SetFocus method that transfers focus to the text box whenever the Change event of the scroll bar occurs. This is typical of the work done by an indirect event trigger when doing data validation and controlling focus transforms. The important thing to observe is that the focus transform from the scroll bar to the text box was completed, but the scroll bar value was not prevented from changing.

When doing data validation, one of the fundamental rules is that you do not allow other field values to change until you have completed the data validation for the field being validated. To do otherwise is to introduce chaos into the data validation algorithm. Of course, there are going to be exceptions to nearly every rule (and there are other ways to handle the problem), but the general rule is prudent. If you do not ruthlessly control focus and prevent other controls from changing when doing data validation, you will have to add additional code to do full screen data validation to cover the cases when you were not able to prevent other fields from changing when they were not supposed to change. In some cases, full screen

data validation is useful and necessary, but most of the time it is not so useful and generally requires extra, unnecessary work.

Another interesting thing to do with this test case is to click on the up or down arrow of the scroll bar and then leave the left mouse button depressed. What you then observe is that the I-bar focus indicator remains in the text box, but the scroll bar value changes rapidly. Also, the Trace List table will be filled with scroll bar Change events. You might ask yourself, "What happens to the GotFocus and LostFocus events for both the text box and the scroll bar that were being recognized after the Change event for the scroll bar?" Then, ask yourself, "Did those events disappear?"

This is where VB event chain behaviors begin to become rather quantum. Events that clearly were occurring now appear not to be occurring; and events that did not occur sequentially (i.e., Change, Change, Change, and so forth) now appear to be occurring quite consistently. Have the rules changed, or does this test case demonstrate that the rules are (perhaps) more elaborate than we originally thought?

Whatever the case may be (and I will leave it as an exercise for you to solve), I think it is reasonable to say that this test case provides a first glimpse into the wonderfully quantum world of VB event processing. However, I will give you a little hint regarding what happens in this test case. At all times, something is controlling the order and sequence of events in a Visual Basic event chain. That something can be Windows (indirectly), Visual Basic (both directly and indirectly), and the programmer (both directly and indirectly); and each of these *somethings* has at its disposal various sets of rules that can be applied to (or can indirectly influence) Visual Basic event chain processing. As a general rule, when a Visual Basic event chain appears to behave in a quantum manner, it is most likely doing so because one of the *somethings* is letting the other somethings dictate the rules.

Test 3: The Controlled SetFocus Case (Everything is Wonderful!)

In this third test case, the easiest way to begin is to click on the "Clear Trace List" push button. The captions of the other three push buttons should read as follows: "MsgBox Statements ON", "Set Scrollbar Test 1", and "Set Scrollbar Test 2". It is very important that the push buttons have these captions. If the captions are different, then you are in another set of test conditions, and if that is the case you should click on the buttons until they have the correct captions. Note that because of the effect of different test settings (controlled by the push buttons), you may have to click on one or more of the push buttons two times to activate the Click event. This is because some of the other test settings cause the VB event chain to behave differently. Once the push button caption are correct, perform the following steps:

1. Change the scroll bar's value to 50, and then (using the mouse), click on the "Clear Trace List" push button. The Trace List will contain the following item:

   (1) pbtClearTraceList-> Click

2. Using the mouse, click on the "Set Scrollbar Test 2" push button. Its caption will change to "Clear Scrollbar Test 2", and the scroll bar's value will change to 1. The Trace List will contain the following items in this order:

   (1) pbtClearTraceList-> Click
   (2) pbtClearTraceList-> LostFocus
   (3) pbtScrollTest2-> GotFocus
   (4) pbtScrollTest2-> Click
   (5) vsrScroll->Change

At this point, observe that the "Clear Scrollbar Test 2" push button still has focus (i.e., there is a bold outline around the push button), but no record of GotFocus and LostFocus events appears in the Trace List. It appears as though the Change event of the scroll bar occurred rather independently when it was remotely triggered by the code for the pbtScrollTest2 push button's Click event.

3. Before proceeding, observe the value of the vertical scroll bar, and remember it. Using the mouse, click on the vertical scroll bar in the grey area between the thumbnail and the up arrow. The scroll bar area across from the label showing the current value of the scroll bar is a good location. Before doing anything else, observe that the value of the scroll bar did not change. The Trace List should now contain nine items in the following order:

   (1) txtText-> GotFocus
   (2) pbtClearTraceList-> LostFocus
   (3) pbtScrollTest2-> GotFocus
   (4) pbtScrollTest2-> Click
   (5) vsrScroll->Change
   (6) vsrScroll->Change
   (7) vsrScroll->Change
   (8) pbtScrollTest2-> LostFocus
   (9) vsrScroll-> GotFocus
   (10) vsrScroll-> LostFocus

In this particular Trace List, the event chain begins with the second item and ends with the first item (i.e., the chain is [2, 3, 4, 5, 6, 7, 8, 9, 10, 1]). The original first item (i.e., "**pbtClearTraceList->Click**") was overwritten by the "**txtText-> GotFocus**" item.

If you want to experiment further with this test case, then click on the text box, combo box, and vertical scroll bar; and observe the Trace List. Tab from control to control, but do not press any of the push buttons except for the "Clear Trace List" push button (which can be clicked at any time).

The primary difference between this test case and test case 3 is that code is activated to prevent the

scroll bar's value from changing when the scroll bar responds to a Change event. In other words, this test case demonstrates that you can prevent a scroll bar's value from changing when doing data validation for some other control. As you examine the code for the various events, note that some special code is required to prevent returning the scroll bar's changed value back to its original value from introducing recursion into the event chain. In other words, all of these events behave the way they do because the programmer is actively controlling the order and recognition of events in the event chain.

Of course, this test case only demonstrates some of the work required to do data validation correctly. In a real data validation test case, there would be special code in all of the other controls; but that is not necessary for this particular example. Here, we are only observing what happens in a few, special conditions.

Test 4: The MsgBox Case (It's Tea Time in Wonderland!)

To begin this test case, it is necessary to do whatever is required to put the push buttons in a state in which they have the following captions: "Clear Trace List", "MsgBox Statements ON", "Set Scrollbar Test 1", and "Set Scrollbar Test 2". Depending on which test case you just completed, it may require clicking on some of the push buttons two times to get them into their proper states.

Once all of the push buttons are in their correct, beginning states, perform the following steps:

1. Using the mouse, click on the "Clear Trace List" push button. The Trace List will contain the following item:

   (1) pbtClearTraceList-> Click

2. Using the mouse, click on the "Set MsgBox ON" push button. The Trace List now contains the following item:

   (1) pbtClearTraceList-> Click
   (2) pbtClearTraceList-> LostFocus
   (3) pbtMsgBoxToggle-> GotFocus
   (4) pbtMsgBoxToggle-> Click

3. Using the mouse, click on the text box. Observe that a MsgBox is displayed with the following message: "**pbtMsgBoxToggle-> LostFocus**". The Trace List now contains the following item:

   (1) pbtClearTraceList-> Click
   (2) pbtClearTraceList-> LostFocus
   (3) pbtMsgBoxToggle-> GotFocus
   (4) pbtMsgBoxToggle-> Click
   (5) pbtMsgBoxToggle-> LostFocus

4. Click on the "OK" button of the MsgBox, and observe that focus is now in the text box. Observe that the Trace List has not changed. What happened to the GotFocus and LostFocus events? Did they disappear?

5. Click on the scroll bar, and observe that a MsgBox is displayed with the following message: "**vsrScroll-> Change**" The Trace List now contains the following items:

   (1) pbtClearTraceList-> Click
   (2) pbtClearTraceList-> LostFocus
   (3) pbtMsgBoxToggle-> GotFocus
   (4) pbtMsgBoxToggle-> Click
   (5) pbtMsgBoxToggle-> LostFocus
   (6) vsrScroll-> Change

6. Click on the "OK" button of the MsgBox, and observe that focus is now in the scroll bar. Observe that the Trace List has not changed. What happened to the GotFocus and LostFocus events? Did they disappear?

7. Click on the "Clear Trace List" push button, and observe that a MsgBox is displayed with the following message: "**vsrScroll-> LostFocus**" The Trace List now contains the following items:

   (1) pbtClearTraceList-> Click
   (2) pbtClearTraceList-> LostFocus
   (3) pbtMsgBoxToggle-> GotFocus
   (4) pbtMsgBoxToggle-> Click
   (5) pbtMsgBoxToggle-> LostFocus
   (6) vsrScroll-> Change

(7) vsrScroll-> LostFocus

8. Click on the "OK" button of the MsgBox, and observe that focus is now in the "Clear Trace List" push button (i.e., it has a bold outline). Observe that the Trace List has not changed. Most importantly, observe that the Click event for the "Clear Trace List" push button has not occurred! The Trace List did not change, and the Click event for the "Clear Trace List" push button did not occur! What happened to the GotFocus, LostFocus, and Click events? Did all of them just disappear? Is something quantum happening? Is it Tea Time in Wonderland?

Of all the test cases in VBEVENTS, this one most clearly demonstrates that the rules followed by VB events and event chains are quantum. By comparing Trace List results for each of the test cases, it certainly appears as though there may be several rules that apply both to each individual event and to each particular event chain. In other words, it should be very clear at this point that it is incorrect to say that every control always gets a LostFocus event when it loses focus. Similarly, it should be clear that it is incorrect to say that every control always gets a GotFocus event when it receives focus. In fact, it should be clear that some of us may not have the vaguest notion of the real rules followed by Visual Basic events and event chains.

Nevertheless, we can say that we know what does actually happen in these test cases, and we can say that we know the rules followed by the events of the particular controls on the VBEVENTS form in the these test cases. We also can state very clearly that the rules we observed are very different from the rules contained in nearly all books on Visual Basic designing and programming with two exceptions: "VB=mc2: The Art of Visual Basic Programming", and "VB=mc4: Visual Basic Standards and Practices" (both available from ETN Corporation). That is the case because only these two, exceptional books were written by me (J.D. Evans, Jr.), and (apparently) I am the only Visual Basic programming book author who actually does real world, Visual Basic application designing and programming. Or, perhaps I am the only author who understands Visual Basic and pays attention to detail; and perhaps I am the only author who carefully verifies that the techniques I write about do, in fact, work precisely as I say they work. I will let you be the judge of that.

Glossary of Terms

Tea Time in Wonderland
focus
passive code
thumbnail
procedure chain
trigger
event chains
Q.E.D.
quantum
36 unique events
ETN Corporation

Tea Time in Wonderland

Alice sighed wearily. "I think you might do something better with the time," she said, "than wasting it asking riddles that have no answers." "If you know Time as well as I do," said the Hatter, "you wouldn't talk about wasting *it*. It's *him*." "I don't know what you mean," said Alice. "Of course you don't!" the Hatter said, tossing his head contemptuously. "I dare say you never even spoke to Time!" "Perhaps not," Alice cautiously replied; "but I know I have to beat time when I learn music."

"Ah! That accounts for it," said the Hatter. "He wo'n't stand beating. Now, if you only kept on good terms with him, he'd do almost anything you liked with the clock. For instance, suppose it were nine o'clock in the morning, just time to begin lessons: you'd only have to whisper a hint to Time, and round goes the clock in a twinkling! Half-past one, time for dinner!" ("I only wish it was," the March Hare said to itself in a whisper.) "That would be grand, certainly," said Alice thoughtfully; "but then--I shouldn't be hungry for it, you know." "Not at first, perhaps," said the Hatter: "but you could keep it to half-past one as long as you liked." "Is that the way you manage?" Alice asked.

"The Hatter shook his head mournfully. "Not I!" he replied. "We quarreled last March--just before *he* went mad, you know--" (pointing his teaspoon at the March Hare,) "--it was at the great concert given by the Queen of Hearts, and I had to sing '*Twinkle, twinkle, little bat! How I wonder what you're at!*' You know the song, perhaps?" "I've heard something like it," said Alice. "It goes on, you know," the Hatter continued, "in this way:--'*Up above the world you fly*, *Like a tea-tray in the sky*. *Twinkle, twinkle*--'" Here the Dormouse shook itself, and began singing in its sleep "*Twinkle, twinkle, twinkle, twinkle*--" and went on so long that they had to pinch it to make it stop. "Well, I'd hardly finished the first verse," said the Hatter, "when the Queen bawled out 'He's murdering the time! Off with his head!'" "How dreadfully savage!" exclaimed Alice. "And ever since that," the Hatter went on in a mournful tone, "he wo'n't do a thing I ask! It's always six o'clock now."

Quoted from "Alice's Adventures in Wonderland" by Lewis Carroll

focus

"In very simple words, Focus is where you *are*. It is not very difficult (most of the time) to know where you *are*, but it can become confusing if you do not know the **Rules of *Focus*, Unannotated**." Quoted from "VB=mc2: The Art of Visual Basic Programming" (1992, ETN Corporation) by J. D. Evans, Jr.

To be eligible to receive focus, an object must be both visible and enabled. Focus transforms (i.e., the transfer of focus from one object to another) are usually initiated by the user interacting with the application's Graphic User Interface (GUI). However, controlling focus transforms is something that must be done by the application, itself. In other words, while the user can certainly take some action that indicates a desire to change focus from one object to another, the application programmer must anticipate this and must provide code that explicitly (and quite ruthlessly) examines the focus transform that has been initiated to determine whether is should be allowed. As a general rule, by the time Visual Basic and the application are notified that a focus transform has been initiated, focus has already been transferred insofar as Windows is concerned. So, when I refer to determining whether to allow a focus transform, I am referring to something that is usually done "ex post facto" (i.e., after the fact). Nevertheless, the most simple way to control focus transforms is to use the Visible and Enabled properties of objects, because (1) if an object is not visible, it cannot receive focus, and (2) if an object is not enabled, it cannot receive focus. Proper use of the Visible and Enabled properties can greatly reduce the amount of programmer intervention required to control focus transforms ruthlessly. I use the word "ruthlessly" when talking about controlling focus transforms, because if you, as a programmer, do not ruthlessly control focus, focus transforms will introduce considerable chaos into your application. In this respect, ruthlessly controlling focus may be the first rule of Windows and Visual Basic programming.

passive code

When used in the context of event processing, the phrase "passive code" refers to code that does not introduce quantum event behaviors. In this context, changing the caption of a label, calculating a total by adding two variables, assigning a literal, string value to a string variable, and so forth are considered to be "passive code" because they have little or no direct influence on either event chains or procedure chains. They may, however, have an indirect influence on chains in the sense that some event in an event chain or some procedure in a procedure chain may use a variable or caption to determine the path taken when a decision statement is executed. Nevertheless, the immediate effect of "passive code" on both event chains and procedure chains is typically negligible.

thumbnail

"A "scroll box", or "thumb," travels the length of the scroll bar to indicate the approximate location of the material shown on the display in relation to the entire document. You can also drag the thumb with the mouse to move to a particular location." Quoted from "Programming Windows 3.1" (1992, Microsoft Press) by Charles Petzold.

I prefer to use the term "thumbnail" because the scroll box looks (to me, anyway) more like a thumbnail than a thumb. I suppose this is a matter of personal preference.

procedure chain

A procedure chain is the procedure-specific analog of an event chain. A procedure chain begins when some procedure is called (typically the procedure call is made within an event). The procedure that begins the procedure chain is called the primary procedure, and all subsequent procedures are called secondary procedures. However, each secondary procedure can become the primary procedure for its procedure chain branch. Procedure chains usually follow a more simple set of rules than event chains; but procedure chains can become recursive. It is important to note that any procedure within a procedure chain can trigger one or more events. Similarly, it is important to note that any event can make a procedure call, and thereby can create another procedure chain. In other words, event chains influence procedure chains, and procedure chains influence event chains. For all practical purposes, event chains and procedure chains are both highly interdependent and very dynamic. Procedures belonging to a procedure chain are often placed in a procedure queue, but once placed in a procedure queue, they are not discarded, except when uncontrolled recursion occurs and program control is lost.

trigger

Whereas code statements are executed, events are triggered. There are several ways in which an event can be triggered: (1) directly by user interaction with the application's Graphic User Interface (GUI), (2) remotely by calling event subroutines (possible only within the same form) and by changing properties of forms and controls that require event processing to complete whatever work they do (possible both in all forms and in all modules), and (3) indirectly by using form and control methods that require event processing to complete whatever work they do (possible both in all forms and in all modules). In some cases, it is necessary to trigger an event remotely from within the same event (e.g., setting the Value property of a scroll bar while in the Change event of that scroll bar is said to remotely trigger another Change event for that same scroll bar). With the exception of explicitly calling event subroutines (which can only be done within the same form, because event subroutines have restricted, form-only visibility), there are also cases in which special techniques can only be performed within a specific form.

event chains

An event chain is a series of events that begins with when a primary event is triggered. The remaining events in the event chain are called secondary events. Prior to the beginning of the event chain, the program is in a rest state (i.e., all outstanding work has been completed, and the program is essentially doing nothing); and, once initiated, the event chain continues until the program is, once again, in a rest state.

Because each of the secondary events in the event chain can, themselves, become primary events for their own event chain branches, it is important to observe that an event chain is not typically a static (i.e., unchanging) entity; rather, an event chain expands (i.e., adds new secondary events) and contracts (i.e., removes secondary events) when viewed over time. In this regard, an event chain can be (and usually is) very dynamic in its behavior.

An event joins an event chain by being placed on the Visual Basic event queue; and an event relinquishes its membership in an event chain by being removed from the Visual Basic event queue.

An event can be removed from the Visual Basic event queue in several ways: (1) by being recognized by the application as a direct result of event occurrence notification, and (2) by being ignored by Visual Basic, internally, in such a way that application-level recognition and notification do not occur. In a poorly managed application, events can be removed from the Visual Basic event queue as a direct result of queue overflow caused by uncontrolled, event recursion (which usually produces a General Protection Fault (GPF) and causes the application to abruptly terminate).

Q.E.D.

Quantum Electrodynamics

quantum

"Working out another system to replace Newton's laws took a long time because phenomena at the atomic level were quite strange. One had to lose one's common sense in order to perceive what was happening at the atomic level. Finally, in 1926, an "uncommon-sensy" theory was developed to explain the "new type of behavior" of electrons in matter. It looked cockeyed, but in reality it was not: it was called the theory of quantum mechanics. The word "quantum" refers to this peculiar aspect of nature that goes against common sense. It is this aspect that I am going to tell you about." Quoted from "QED: The Strange Theory of Light and Matter" (1988, Princeton University Press) by Richard P. Feynman .

"For what quantum mechanics says is that nothing is real and that we cannot say anything about what things are doing when we are not looking at them." Quoted from "In Search of Schrodinger's Cat: Quantum Physics and Reality" (1988, Bantam Books) by John Gribbon.

"Anyone who is not shocked by quantum theory has not understood it." Niels Bohr.

Quotation found in the Gribbon book listed above.

36 unique events

Activate, Change, Click, DblClick, Deactivate, DragDrop, DragOver, DropDown, Error, GotFocus, KeyDown, KeyPress, KeyUp, LinkClose, LinkError, LinkExecute, LinkNotify, LinkOpen, Load, LostFocus MouseDown, MouseMove, MouseUp, Paint, PathChange, PatternChange, QueryUnload, Reposition, Resize, RowColChange, Scroll, SelChange,Timer, Unload, Updated, Validate

ETN Corporation

ETN Corporation
RR4 Box 659 Montoursville, PA 17754-9433 U. S. A.
Tel: (717) 435-2202 Fax: (717) 435-2802