Trapping Mouse and Keyboard Events in Graphics

Table of Contents

Some information and code in this article are taken from the Liberty BASIC helpfile.

Box or Window?

The event trapping information given below works in the same way for graphicbox controls and for windows opened for graphics.

WARNING: graphicboxes in dialog-type windows do not properly accept the input focus for keyboard events. If a program needs graphicboxes that trap keyboard events, use a window of type "window".

Focus

Keyboard events are only trapped when a graphicbox or graphics window has the input focus. For example, to cause focus to be directed to a graphicbox, issue a SETFOCUS command.

```
#main.graphicbox "setfocus"
```

When Event

The only way to trap mouse events and keyboard events in a Liberty BASIC program is within a graphicbox or graphics window. This is done with the "when event" command. The synax is as follows:

```
#graphicsHandle "when event eventHandler"
```

Mouse events consist of left button mouse single-clicks, double-clicks, and drags, right button single-clicks, double-clicks and drags, middle button single-clicks, double-clicks and drags, and mouse moves when no button has been clicked.

Keyboard events are triggered when the user presses a key on the keyboard.

The eventHandler for the "when event" command can be a valid branch label or the name of a subroutine.

List of Events

Most commands are not case sensitive. That is not true of the "when event" commands. The event names **are** case sensitive. The capitalization scheme is different than other case sensitive keywords, where the first letter is usually upper case. The first letter in "when event" commands is lower case. "LeftButtonDown" is not correct and will not work. "leftButtonDown" is correct and will work.

Case Counts!

leftButtonDown	the left mouse button has been pressed
leftButtonUp	the left mouse button has been released
leftButtonMove	the mouse moved while the left button was down
leftButtonDouble	the left mouse button has been double-clicked
rightButtonDown	the right mouse button has been pressed
rightButtonUp	the right mouse button has been released

rightButtonMove	the mouse moved while the right button was down
rightButtonDouble	the right mouse button has been double-clicked
middleButtonDown	the middle mouse button has been pressed
middleButtonUp	the middle mouse button has been released
middleButtonMove	the mouse moved while the middle button was down
middleButtonDouble	the middle mouse button has been double-clicked
mouseMove	the mouse moved when no button was down
characterInput	a key was pressed while the graphics window has input focus

Turning Off Event Trapping

To stop trapping an event, issue the "when event" command without a sub or branch label, like this:

```
#main.gbox "when leftButtonDown"
```

Trapping for any event can be turned on or off as many times as needed in the program.

Mouse Coordinates and Keys Pressed

Whenever a mouse event is trapped, Liberty BASIC places the x and y position of the mouse in the variables MouseX, and MouseY. The values represent the mouse location as the number of pixels in x and y from the upper left corner of the graphic window display pane. The special variables "MouseX" and "MouseY" are case sensitive, as are all special variables in Liberty BASIC.

Whenever a keyboard event is trapped, Liberty BASIC places the value of the key(s) pressed into the special variable, Inkey$. The special variable Inkey$ is case sensitive, as are all special variables in Liberty BASIC.

Branch Label Event Handlers

If the event handler for the mouse or keyboard input is a branch label, Liberty BASIC will fill special variables with information. If the event is a mouse event, the location of the mouse is placed into the special variables MouseX and MouseY. These special variable names are case sensitive, so mouseX or mousex are incorrect, for instance. The values are expressed as the number of pixels from the upper left corner of the graphics area.

The following small demo allows you to see the mouse coordinates when the left button is clicked and when the mouse is moved with no buttons pressed.

```
graphicbox #1.g, 0,0,200,200
statictext #1.s, "Mouse Coordinates", 10,210,200,50
statictext #1.t,
"Click left mouse button or move mouse.",10,260,200,50
open "Mouse Events" for window as #1
#1 "trapclose [quit]"
#1.g "when leftButtonDown [leftDown]"
#1.g "when mouseMove [mouseMoved]"
wait

[leftDown]
#1.s "MouseX is ";MouseX;"      MouseY is ";MouseY
wait

[mouseMoved]
#1.s MouseX;"      ";MouseY
wait

[quit]close #1:end
nomainwin
```

Sub Event Handlers

If you use subroutines as mouse event handlers, be aware that branch labels elsewhere in the program are not visible inside of subs. If the program attempts to handle other events while processing a mouse event and it attempts to access a branch lable in the main program, the program will crash.

It's best to use subs for all other routines that handle user events when using subs for mouse event handlers. Be sure to turn off mouse handling events when they are not needed, or when the event triggers a complex code block. Events can stack up while waiting for code to finish executing and unexpected errors can occur.

```
graphicbox #1.g, 0,0,200,200
```

```
statictext #1.s, "Mouse Coordinates", 10,210,200,50
statictext #1.t,
"Click left mouse button or move mouse.",10,260,200,50
open "Mouse Events" for window as #1
#1 "trapclose Quit"
#1.g "when leftButtonDown leftDown"
#1.g "when mouseMove mouseMoved"
wait

sub leftDown hndle$, mx, my
    #1.s "MouseX is ";mx;"      MouseY is ";my
    end sub

sub mouseMoved hndle$, mx, my
    #1.s mx;"      ";my
    end sub

sub Quit hndle$
    close #hndle$:end
    end sub

nomainwin
```

Reading Keystrokes

Keyboard input can only be trapped in graphics windows or graphicboxes that have the input focus. When a key is pressed, the information is stored in the special variable Inkey$. The name Inkey$ is case sensitive, as are all variable names in Liberty BASIC.

This special variable holds either a single typed character or multiple characters including a Windows virtual key code. The virtual key codes are standard Windows constants, and include arrow keys, function keys, the ALT, SHIFT, and CTRL keys, etc.

If Inkey$ is a single character, that character will be the key pressed. If the length of Inkey$ is more than 1, it holds multiple key information:. If Inkey$ holds more than one character, the first character will indicate whether the Shift, Ctrl, or Alt keys was depressed when the key was pressed. These keys have the following values:

Shift = 4
Ctrl = 8
Alt = 16

They can be used in any combination. If Inkey$ contains more than one character, you can check to see which (if any) of the three special keys was also pressed by using the bitwise AND operator. If shift alone was pressed, then the value of the first character will be 4. If Shift and Alt were both pressed, then the value of the first character will be 20, and so on. Special keys trigger a new value for Inkey$ when they are

pressed and again when they are released. Here is an example that uses bitwise AND to determine which special keys were pressed.

```
 open "Inkey$ with Shift" for graphics_nf_nsb as #1
  #1 "setfocus; when characterInput [check]"
  #1 "down; place 10 30"
  #1 "\Make the mainwindow visible,"
  #1 "\then click this window and"
  #1 "\begin pressing key combinations."
  #1 "\Watch the printout in the mainwindow."
  #1 "flush"
  #1 "trapclose [quit]"
  wait

[check]
  shift=4
  ctrl=8
  alt=16
  a=asc(left$(Inkey$,1))
  if len(Inkey$)>1 then
    m$=""
    if a and shift then m$="shift "
    if a and ctrl then m$=m$+"ctrl "
    if a and alt then m$=m$+"alt "
    print "Special keys pressed: " + m$
  else
    print "Key pressed: " + Inkey$
  end if
  wait

[quit]
  close #1
  end
```

Virtual-Key Codes

Liberty BASIC has a library of Virtual-Key codes that it understands, such as

_VK_NUMPAD2

```
'Alpha-numeric keys are expressed as ASCII values:
PRINT ASC("A")
```

```
PRINT ASC("a")

'Special keys use Virtual Key Constant values:
PRINT _VK_NUMPAD2
PRINT _VK_APPS
PRINT _VK_F7
PRINT _VK_RSHIFT
PRINT _VK_F24
```

If you attempt to use an unknown Virtual-Key code, you'll get an error message from Liberty BASIC. In that case, use the value of the Virtual-Key constant. You can find the values online:

Here is a list of Virtual-Key Codes from the Microsoft Developer's Network Library:
Virtual-Key Codes

The values are given in hexadecimal form. Use HexDec() to find the decimal value. The 0x at the beginning of hexadecimal numbers simply signals the format in some languages. Omit that part in Liberty BASIC. Here is an example.

```
'VK_LAUNCH_APP2  (0xB7)
print hexdec("B7")
```

Key Up and Key Down

Special keys trigger a new value for Inkey$ when they are pressed and again when they are released.

A virtual key is the key that is actually pressed on the keyboard. The VK value for a lower case letter is the same as the value for the same letter in upper case because it refers to the key pressed on the keyboard, not to the ASCII value of the input. Most keys have a graphical representation. You can see letters, numbers and symbols in a texteditor as they are typed, for instance. There are some keys that do not have a graphical representation. You must use Virtual Key Codes to discover which of these keys has been pressed. They include the arrow keys, the F-keys, Shift, Ctrl, Alt, Del, etc.

Here is a program that gives a quick example:

```
    'Inkey$ example, part two
    ctrl$ = chr$(_VK_CONTROL)
    print "Keys pressed:"
    open "Inkey$ example" for graphics as #graph
    print #graph, "when characterInput [keyPressed]"
    print #graph, "trapclose [quit]"
[loopHere]
```

```
    'make sure #graph has input focus
    print #graph, "setfocus"
    'scan for events
    scan
    goto [loopHere]
[keyPressed]
    key$ = left$(Inkey$, 2)
    if len(key$) < 2 then
        print "pressed: "; key$
    else
        if right$(key$, 1) = ctrl$ then
            print "CTRL was pressed"
        else
            print "Unhandled special key"
        end if
    end if
    goto [loopHere]
[quit]
    print "Quitting"
    close #graph
    end
```

Table of Contents

Virtual-Key Codes

Key Up and Key Down