Coding with Sub Event Handlers

Janet Terra

Table of Contents

Liberty BASIC supports both ***Branch Label*** handling and ***Sub*** handling for control events. Alyce Watson Liberty BASIC Newsletter Issue #137, Tip Corner - a SUB for Resizehandler discussed local visibility and global visibility in terms of coding a resizehandler. A sub can call another sub, but a sub cannot branch to a branch label outside that sub. Using all subs may prevent a program crash due to *unseen* branch labels. Another distinct advantage of using a sub handler rather than a branch label handler for control events is that the handle of the selected control is passed into the sub. The defined sub must contain a string variable to receive the handle, but any handle can be passed. This allows for multiple controls accessing the same sub. Of course multiple controls may be directed to the same branch label, but the branch label has no way of discerning which control triggered the code. In a sub, the passed handle can be parsed for identification. If you are using a number of like controls, this parsing may afford more streamlined code.

One Sub Rather than Many Branch Labels

Consider a program with 5 buttons. Each button is designed to launch a different application. Using branch labels, a separate branch label is required for each button.

```
    Nomainwin
    WindowWidth = 200
    WindowHeight = 300

    UpperLeftX = Int((DisplayWidth - WindowWidth)/2)
    UpperLeftY = Int((DisplayHeight - WindowHeight)/2)

    Menu #main, "&Options", "E&xit", [EndDemo]
    Button #main.app1, "Notepad", [app1], UL, 30, 20, 110, 20
    Button #main.app2, "MS Paint", [app2], UL, 30, 50, 110, 20
    Button #main.app3, "Calculator", [app3], UL, 30, 80, 110, 20
    Button #main.app4, "Sound Recorder", [app4], UL, 30, 110, 120, 20
    Button #main.app5, "Spider", [app5], UL, 30, 140, 120, 20

    Open "Launching Applications" for Window as #main
    #main "Trapclose [EndDemo]"

Wait

[EndDemo]
    Close #main
End

[app1]
    Run "Notepad.exe"
Wait

[app2]
    Run "MSPaint.exe"
Wait

[app3]
    Run "Calc.exe"
Wait

[app4]
    Run "sndrec32.exe"
Wait

[app5]
    Run "Spider.exe"
Wait
```

With a sub event handler, that code can be greatly optimized.

```
'Choose 5 common MS Applications
    App$(1) = "Notepad.exe"
    App$(2) = "MSPaint.exe"
    App$(3) = "Calc.exe"
    App$(4) = "sndrec32.exe"
    App$(5) = "Spider.exe"
    Nomainwin
    WindowWidth = 200
    WindowHeight = 300

    UpperLeftX = Int((DisplayWidth - WindowWidth)/2)
    UpperLeftY = Int((DisplayHeight - WindowHeight)/2)

    Menu #main, "&Options", "E&xit", [EndDemo]

    Button #main.app1, "Notepad", App2Run, UL, 30, 20, 110, 20
    Button #main.app2, "MS Paint", App2Run, UL, 30, 50, 110, 20
    Button #main.app3, "Calculator", App2Run, UL, 30, 80, 110, 20
    Button #main.app4, "Sound Recorder", App2Run, UL, 30, 110, 120, 20
    Button #main.app5, "Spider", App2Run, UL, 30, 140, 120, 20

    Open "Launching Applications" for Window as #main
    #main "Trapclose [EndDemo]"

Wait

[EndDemo]
    Close #main
End

Sub App2Run handle$
    app = Val(Right$(handle$, 1))
    Run App$(app)
End Sub
```

The common sub is then parsed for the control that triggered it and the appropriate code is executed. Constructing an array that correlates with the numbered handle extensions, as in the preceding demo, will streamline your code even further.

Menus, Controls, and Mouse Events

When using a branch label, menu items, controls (buttons, listboxes, comboboxes, etc.), and even mouse events, can branch to the same label.

```
    Nomainwin
    WindowWidth = 250
    WindowHeight = 154
    Menu #main, "&File", "&Random Color", [RandomColor],|,
"E&xit", [EndDemo]
    Graphicbox #main.gbx, 0, 0, 100, 100
    Button #main.btn, "Random Color", [RandomColor], UL, 120, 10,
100, 30
    Button #main.exit, "Quit", [EndDemo], UL, 120, 50, 100, 30
    Open "Sharing Branch Labels" for Window as #main
    #main "Trapclose [EndDemo]"
    #main.gbx "Down"
    #main.gbx "When leftButtonUp [RandomColor]"

Wait

[EndDemo]
    Close #main
End

[RandomColor]
    redHue = Int(Rnd(1) * 256)
    greenHue = Int(Rnd(1) * 256)
    blueHue = Int(Rnd(1) * 256)
    #main.gbx "Fill ";redHue;" ";greenHue;" ";blueHue
Wait
```

Because handles are passed to subs, different event handles may need to be assigned for different types of control. Menus do not pass handles, controls do. A mouse movement passes not only the handle, but also the current `MouseX` and `MouseY` coordinates. Event handling subs must be assigned that will accommodate these handles and parameters. The menu sub with no handle variable must be different from the button sub requiring a handle variable which must be different from the mouse event sub requiring a handle variable and `MouseX` / `MouseY` parameters. This doesn't mean that code needs to be duplicated in each sub. Simply call one sub from another. In this demo, the menu and the left mouse click both call the button code, as though the button itself had been clicked. Exiting with the menu option calls the trapclose exit sub.

```
    Nomainwin
    WindowWidth = 250
    WindowHeight = 154
    Menu #main, "&File", "&Random Color"
, RandomColorMenu,|, "E&xit", XbyMenu
    Graphicbox #main.gbx, 0, 0, 100, 100
    Button #main.btn, "Random Color"
, RandomColor, UL, 120, 10, 100, 30
    Button #main.exit, "Quit", EndDemo, UL, 120, 50, 100, 30
    Open "Assigning Subs" for Window as #main
    #main "Trapclose XbyTrap"
    #main.gbx "Down"
    #main.gbx "When leftButtonUp RandomColorMouse"

Wait

Sub XbyMenu
    Call XbyTrap "#main"
End Sub

Sub XbyTrap handle$
    Close #main
    End
End Sub

Sub RandomColorMenu
    Call RandomColor "#main.btn"
End Sub

Sub RandomColorMouse handle$, xVar, yVar
    Call RandomColor "#main.btn"
End Sub

Sub RandomColor handle$
    redHue = Int(Rnd(1) * 256)
    greenHue = Int(Rnd(1) * 256)
    blueHue = Int(Rnd(1) * 256)
    #main.gbx "Fill ";redHue;" ";greenHue;" ";blueHue
End Sub
```

Remember you will need to include the missing parameters whenever a sub event is being called from another sub. The menu exit option must include a handle to be passed when calling the trapclose sub event. Unless you are parsing, this parameter doesn't have to be valid.

```
    Call EndDemo "whatchamacallit"
```

or even

```
    Call EndDemo ""
```

will work just as well as

```
    Call EndDemo "#main"
```

provided the handle$ variable isn't being relied upon in the sub.

In the Liberty BASIC Newsletter Issue #126, Mike Bradbury uses one sub event handler to identify and manage seating arrangements using 48 separate graphicboxes Demo: Sub Handlers. That same program would require 48 separate branch label events. Aside from streamlining code, there is at least one more advantage to using event sub handlers -- keeping track of open windows.

Managing Multiple Windows with Sub Events

When multiple windows can be opened by the user within the same application, the programmer must find a way to know which windows are opened and which are closed. Using sub events can help the program to keep track of open windows and prevent program crashes from trying to reopen an already open window, or from trying to end with one or more windows still open. In this next demo, an array is used to keep track of open windows. When the window is opened, the handle is passed into the array. When the window is closed, the array element is reset to null. Looping through the array when closing the main window finds which handles have yet to be closed.

```
    Dim OpenWindow$(12)
    Nomainwin

    WindowWidth = 400
    WindowHeight = 400

    UpperLeftX = Int((DisplayWidth - WindowWidth)/2)
    UpperLeftY = Int((DisplayHeight - WindowHeight)/2)

    Button #main.w01, "Accessory Window #1"
, AccWin, UL, 30, 50, 140, 26
    Button #main.w02, "Accessory Window #2"
, AccWin, UL, 30, 100, 140, 26
    Button #main.w03, "Accessory Window #3"
, AccWin, UL, 30, 150, 140, 26
    Button #main.w04, "Accessory Window #4"
, AccWin, UL, 30, 200, 140, 26
    Button #main.w05, "Accessory Window #5"
, AccWin, UL, 30, 250, 140, 26
    Button #main.w06, "Accessory Window #6"
, AccWin, UL, 30, 300, 140, 26
```

```
    Button #main.w07, "Accessory Window #7"
, AccWin, UL, 220, 50, 140, 26
    Button #main.w08, "Accessory Window #8"
, AccWin, UL, 220, 100, 140, 26
    Button #main.w09, "Accessory Window #9"
, AccWin, UL, 220, 150, 140, 26
    Button #main.w10, "Accessory Window #10"
, AccWin, UL, 220, 200, 140, 26
    Button #main.w11, "Accessory Window #11"
, AccWin, UL, 220, 250, 140, 26
    Button #main.w12, "Accessory Window #12"
, AccWin, UL, 220, 300, 140, 26

    Open "Multiple Windows" for Window as #main
    #main "Trapclose XbyTrap"
    #main "Font Ariel 8 Bold"
    Wait

Sub EndDemo handle$
    For i = 1 to 12
        If OpenWindow$(i) <> "" Then
            winHandle$ = OpenWindow$(i)
            Close #winHandle$
        End If
    Next i
    Close #main
    End
End Sub

Sub AccWin handle$
    win = Val(Right$(handle$, 2))
    winHandle$ = "#acc";Right$("0";win, 2)
    If OpenWindow$(win) <> "" Then
        Exit Sub ' Don't reopen an already open window
    End If
    If win < 7 Then
        ulx = Int(DisplayWidth / 5) + 1
        uly = Int(DisplayHeight / 8) * (win - 1) + 1
    Else
        ulx = Int(DisplayWidth / 5) * 3 + 1
        uly = Int(DisplayHeight / 8) * (win - 7) + 1
    End If
    WindowWidth = 100
    WindowHeight = 80
    UpperLeftX = ulx
    UpperLeftY = uly
    title$ = "Accessory Window #";win
' Following requires Case Select because variables cannot be used for
handles
' prior to opening the window
    Select Case win
        Case 1
            Open title$ for Window as #acc01
        Case 2
            Open title$ for Window as #acc02
        Case 3
            Open title$ for Window as #acc03
        Case 4
            Open title$ for Window as #acc04
        Case 5
            Open title$ for Window as #acc05
        Case 6
            Open title$ for Window as #acc06
        Case 7
            Open title$ for Window as #acc07
        Case 8
            Open title$ for Window as #acc08
        Case 9
            Open title$ for Window as #acc09
        Case 10
            Open title$ for Window as #acc10
        Case 11
            Open title$ for Window as #acc11
        Case 12
            Open title$ for Window as #acc12
    End Select
    OpenWindow$(win) = winHandle$
' Now that the window is open, variables can be used for handles
    #winHandle$ "Trapclose CloseAcc"
End Sub

Sub CloseAcc handle$
    win = Val(Right$(handle$, 2))
    Close #handle$
    OpenWindow$(win) = ""
End Sub
```

It is not necessary to number the accessory windows as such. Any names will do. This demo looks at `OpenWindow$(win)`. If your window names aren't in any logical sequence, just loop through the entire array to find a match.

An End to Multiple WAITS, GOTOs

Once a sub has been executed, program execution reverts to the state prior to calling the sub. In most cases, your program will need only one WAIT statement. Since the events are triggered by controls, there is no need for a single GOTO statement. In a recent discussion of Sub Events at the Liberty BASIC Forum, Carl Gundel, author of Liberty BASIC clarified, "*WAIT does not use GOTO if your event handlers are all SUBs. The SUB will get executed, and when it is finished you will be left at the same WAIT statement.*"

So get control of your controls using **Sub Event Handlers**. You may find the results well worth the effort.