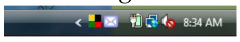
Running a LB program from the System Tray.

Author : Duncan Bushaw

Published June 17, 2007.

Editor's note. The system tray is more properly called the taskbar notification area. It is usually at the end of the desktop taskbar and contains small icons for running programs. Clicking the mouse on these icons causes the program windows to display. Hovering the mouse over these icons may invoke tooltips. The taskbar notification area looks like this. 

Introduction.

The example program, 'TrayIcon.bas' (hereafter refered to as the program) uses Brent Thorn's WMLiberty.dll, which is included in this file

TrayIcon.CAB

- Details
- Download
- 31 KB

. The source code for Brent Thorn's WMLiberty.dll is available at Bay 6 Software at http://www.b6sw.com/. A full working package of the program can be downloaded from my domain at http://duncanb.nh-networks.com/.

For the purpose of this article a windowless control is any type of control that does not necessarily have to be assigned to a active window (ie; Liberty BASIC controls such as NOTICE or POPUPMENU; Windows API controls such as ShellAboutA or MessageBoxExA), and a Window is any window created using the Liberty BASIC 'OPEN' command.

Any standard approach to writing Liberty BASIC code to place a program in the system tray only seems to work for very basic programs containing only a popup menu. At some point in writing more complex code the program starts to generate various errors during testing.

The most common errors I see and hear about are the 'Invalid branch label', 'stack overflow', and 'popup menu already active' errors.

The errors have to do with the CALLBACK function TrayIcon() (This is the CALLBACK function that is invoked by the CALLDLL command using the WMLiberty.dll) which is called whenever the mouse (cursor) moves over the icon on the system tray. This causes the program to look only at the code routine that is contained in the TrayIcon() CALLBACK funtion. I used the GLOBAL variable (TrayIconPointer) to remind the program what it should be doing whenever the TrayIcon() CALLBACK function is invoked.

The program uses many Liberty BASIC commands and a few API controls, and it shows one technique that seems to encompass most controls and windows. It has 2 Liberty BASIC windows opened and active at one time, without any of the previously mentioned errors occurring.

The technique used in this program is not the only way of writing a program in Liberty BASIC which can run on the system tray, but I find it to be a useful technique that seems to encompass all possible controls whether Liberty BASIC native commands or windows API calls.

To do this I had to 'Think Outside Of The Box' when using function calls in the program. You will see a lot of WAIT commands, which in itself is not unexpected. The difference is that when the TrayIcon() CALLBACK function is called and a window is then opened using it, the program does not exit the function which is called to open the window, whereas when the function which opens the window is called directly by the program and not through the TrayIcon() CALLBACK function, it must be exited properly for the program to continue working correctly.

Think of it this way. Only when a windowless control such as a notice message, has been activated, will the program exit the TrayIcon() CALLBACK function. If any window or regular control has been activated the program will encounter the Liberty BASIC 'WAIT' command. It will then only refer to the code in TrayIcon() CALLBACK function if no windows are active.

If the 'Test Window' is active the program will call the Testwindow() function and then only refer to the code in this function.

The same idea happens when the dialog window is active, but it will call the Retrieve() function, which contains duplicated code from the TestWindow() function, so both windows can be enabed and all controls can work.

When the program closes the 'Test window' and the TrayIcon() CALLBACK function is active, the program must encounter a 'WAIT' command and not exit the Testwindow() function. If the TrayIcon() CALLBACK function is not active the program must exit the Testwindow() function.

To demonstrate this, the Testwindow() function can be called two ways; before or after the TrayIcon() CALLBACK command is initiated.

```
IF INSTR(CommandLine$,"-TestWindow")>0 then OP=1:null=TestWindow()
```

The above line of code is why the TKN file and the runtime engine must be in the same folder/directory as the BAS file. This is how the menu item 'Test window on Restart' opens the window before putting the icon on the system tray.

If the CommandLine$ contains the string "-TestWindow" in it, the program sets the GLOBAL variable (OP) to the value of 1 and then calls the TestWindow() function, which will then open the Test Window and the program will wait and update the clock in the GRAPHICBOX control every second using the TIMER command.

All branch labels for the window are contained within the function code itself and when the close button on the title bar is clicked or (ALT-F4) is pressed the program looks for the branch lable [quit.WindowIcon] within the TestWindow() function and turns off the TIMER, closes the window, sets the TrayIconPointer to the value of Zero, and then checks the (OP) variable. If it is 0 then the program does not exit the function. Instead it waits and exits when called from the TrayIcon() CALLBACK function to prevent errors from happening. However if the dialog window has also been opened then the branch label [quit.WindowIcon] is looked for in the Retrieve() function as that is the active function at that time.

TrayIcon.bas the program.

The Liberty BASIC runtime engine (run403.exe) copied and renamed as TrayIcon.exe and the TKN (tokenized BAS file) must be in the same dir./folder as TrayIcon.bas for all the program features to work without errors. A full working package can be downloaded from my domain at http://duncanb.nh-networks.com/.

The program begins by extracting the icon from TrayIcon.exe and puts this icon on the system tray. From here there are 2 popup menus available.

Double clicking the left mouse button calls one of the popup menus and a single click of the right mouse button causes a popup menu to appear when the button is released. It is best not to make a popup menu appear when a mouse button is pressed down but when the mouse button is released. This allows any other popup menus that may be active to close before the new popup menu is displayed.

The left button menu has 4 menu items which are 'About', 'Test window', 'NOTICE', and 'Help'.

The right button menu has 4 menu items which are 'Test window', 'Test window on Restart', 'About', and 'Exit'.

The 'About' menu item displays the system's about dialog window.

The 'NOTICE' menu item uses the native LB 'NOTICE' command.

The 'Help' menu item opens the system's default browser and goes to this LBPE web site.

The 'Exit' menu item ends the program.

The 'Test window on Restart' starts another instance of the program, which will show the Test window before putting the icon on the system tray, and then ends the first instance of the program. You must have a compiled (TKN) file in the same directory as TrayIcon.bas, and have a copy of the runtime engine (run403.exe) named TrayIcon.exe for this to work.

The 'Test window' will open the Test window which is of a window_nf type, and the tray icon will have a different popup menu available with the right mouse button. The left mouse button is disabled while the Test window is open.

The Test window.

Is of a window_nf type.

I have included many native Liberty BASIC controls in the Test Window. They are BUTTON, CHECKBOX, COMBOBOX, GRAPHICBOX, GROUPBOX, LISTBOX, MENU, NOTICE, POPUPMENU, RADIOBUTTON, STATICTEXT, TEXTBOX, TEXTEDITOR, and TIMER.

Selecting a CHECKBOX, COMBOBOX, LISTBOX, or RADIOBUTTON item causes a NOTICE window to appear displaying the selected item or state of control (set/reset).

The GRAPHICBOX is used to display a clock which I ported from 'clasic Face clock.bas' (see this article of mine http://lbpe.wikispaces.com/A+True+API+Popup+Menu+-+By+Duncan+B) which uses the TIMER control to update the clock every second.

The 'Retrieve Text' button will retrieve anything in the TEXTBOX and display it in a NOTICE window.

The 'Retrieve All Items' button will cause a POPUPMENU to appear and the user can select which window the dialog_nf_modal window will be attached to (the hidden window or the Test Window). After selecting the window to attach the dialog_nf_modal window to the program will then open the dlalog_nf_modal window with only a TEXTEDITOR control in the window (neat no menu bar in this window) and then retrieve the current state of all CHECKBOXES and RADIOBUTTONS (set/reset), the current selected LISTBOX and COMBOBOX items (if any), the contents of the COMBOBOX, TEXTBOX, and TEXTEDITOR (if any), and then display them in the TEXTEDITOR in the dialog_nf_modal window.

The 'Retrieved All Items' window.

Is of a dialog_nf_modal type.

When this window is opened the tray icon will have a different popupmenu again (right click only) and the clock in the GRAPHICBOX in the 'Test Window' will continue to be updated every second.

If this window is attached to the 'Test Window' then the 'Test Window' will be disabled but will not appear to be disabled, as the program makes the 'Test Window' the foreground window even though the dialog_nf_modal window is on top. Clicking on the 'Test Window' with the mouse will cause it to gray and make the dialog_nf_modal window the foreground window.

If this window is attached to the hidden window then the 'Test Window' will not be disabled and will be set as the foreground window putting it on top of the dialog_nf_modal window. Moving the mouse over (do not click) the tray icon the 'Retrieved All Items' window will be set as the foreground window putting it on top of the 'Test Window'. I did this to show that the CALLBACK function will be tripped just by moving the mouse over the tray icon. The 'Test Window' will remain enabled and all the controls will still work.

The TrayIconpointer.

This variable works best if it is set before any windows or windowless controls have been opened or are made active .

The TrayIconpointer variable has 4 states:

- [-1] - This value tells the program that a windowless contol is active and to exit the Traylcon() CALLBACK function right away.
- [0] - This value tells the program that there are no windowless controls or windows active and causes the program to wait inside the Traylcon() CALLBACK function.
- [1] - This value tells the program that the Test window is active and causes the program to jump to the Testwindow() function and to wait inside the function.
- [2] - This value tells the program that the dialog window is active and causes the program to jump to the Retrieve() function and to wait inside the function.

The ON ERROR GOTO [Error.Handler].

This was added on Oct., 28, 2007 and solves a fatal error issue when a window is already open and cursor clicks on the notification area icon and there are active controls available at the icon, in my case a POPUPMENU. If the icon has no controls and any Windows messages are ignored the error does not seem to occur.
This error does not happen on all computers. I have run the same programs on similar Operating Systems (XP w/SP2,1 updated via internet the other not) but different PC manufaturers and it occurs on some but not others and took me a while to realize what was going on. So I have used the ON ERROR GOTO routine to handle this error. (Thanks Carl for including this ability to handle error messages, I used the QBASIC form for years and it can be very helpfull at times.)
ON ERROR GOTO [Error.Handler]
This sends the program to the specified branch in the program when any errors occur.
[Error.Handler]
The branch that handles all errors.
IF Err=0 AND LEFT$(Err$,6)="Handle" AND RIGHT$(Err$,14)="already in use" THEN WAIT
This line checks the type of error # and error message string for certain word strings. An error message

like "Handle #WindowIcon already in use" can still happen the only thing that changes from error message to error message is the active window handle the rest of the message remains the same.
If all parameters match then the program just waits as if nothing happened at all, and the program does not crash.
IF Err=0 AND Err$="Handle #WindowIcon already in use" THEN WAIT
This is an alternate form I have used but do not use now.The reason for not using it is that it only checks against one window handle. Therefore the more windows the more lines of code to deal with the error, the first example will catch all the error messages regardless of the window handle being returned by the error message so less code to write.

NOTICE "Liberty Basic Error Handler."+Cr$+"Error number "+STR$(Err)+Cr$+"Error string is "+Qu$+Err$+Qu$:END
This line deals with all other errors and displays them before ending the program. With the information that is provided in the NOTICE box before the program ENDs it is then possible to include code that would deal with it to prevent the program from crashing in the future. The program has not generated any other errors and since I have started to use this error routine in my current project of 2500+ lines with 5 different LB windows and up to three being opened at one time and each window has its own POPUPMENU when the icon is clicked on.

A Hint or Two.

I have noticed that I have to define a lot of variables as GLOBAL for things to work smoothly. This I find is my most frequent problem when adding new code. Everything seems fine when a window is first opened but when the TrayIcon CALLBACK function is envoked the variables that a window might need to remember things are reset to the value of zero or a null string unless they have been defined as GLOBAL when the program starts. Array variables { a(1) a$(1) a(1,1) a$(1,1) }, window handles { #WindowIcon }, control handles { #WindowIcon.button1 }, and structures { iconData.hIcon.struct } are GLOBAL in nature and can be used from within any FUNCTION or any SUB.

The TrayIconpointer variable should be set/changed right away before the code goes onto anything else.

I hope that this helps those frustrated LB programmers who have tinkered with the WMLiberty.dll, I spent many hours trying different codes and this solution seems so obvious to me now. Enjoy!