Creating a Shell

Painting the Desktop

by Alyce Watson - http://alycesrestaurant.com/

-

Alyce

Table of Contents

What is a Shell?

A shell is an application that manages the user's applications and documents. It is not the same as an Operating System. A shell runs on top of an operating system. We can create a shell for Windows in Liberty BASIC. We'll paint a desktop over the existing desktop and allow the user to add icons to it for their applications.

Setting Up a Desktop

We'll use a window of style "window_popup" to cover the desktop. This style does not have a titlebar, so it's a great choice for a desktop replacement. We'll make it the DisplayWidth and DisplayHeight dimensions, so it covers the entire existing desktop. We'll include a graphicbox that covers the entire client area of the window. The graphicbox allows us to capture mouse clicks and to flush the graphics to make them stick. We'll add a button to the lower left corner of the desktop that allows the user to exit our shell. That's the default location for the Windows start button, so the user will expect to find it there. The user could exit our shell by pressing and holding the Alt button, then pressing F-4, but it's more polished to provide an easy way to exit an application.

```
WindowWidth=DisplayWidth:WindowHeight=DisplayHeight

graphicbox #desk.g, -1, -1, DisplayWidth+2, DisplayHeight+2
button #desk.exit, "Exit", DoExit, UL, 4, DisplayHeight-30, 56, 28

open "My Desktop Shell" for window_popup as #desk
```

Painting the Desktop

Windows provides an API function called "PaintDesktop". It paints the desktop image onto the Device Context specified. The desktop image includes the user's selected background color and image file. It does not include the icons displayed on the user's desktop. The function is provided to allow people to create their own Windows shell programs.

For the Device Context, we will "GetDC" for the grahicbox at the start of the program and "ReleaseDC" when the program ends. Those functions look like this:

```
    CallDLL #user32, "GetDC",_
        hWnd as uLong,_  'graphicbox handle
        hDC as uLong     'returns handle to Device Context

    CallDLL#user32,"ReleaseDC",_
        hWnd as uLong,_  'graphicbox handle
        hDC as uLong,_   'handle to Device Context
        result As Long
```

Once we have the DC of the graphicbox, it is super-simple to paint the user's desktop image onto it.

```
    calldll #user32, "PaintDesktop",_
```

```
        hDC as ulong,_   'handle of graphicbox device context
        re as long    'nonzero = success
```

Any time we make a change to our desktop's appearance, we'll flush it with "GetBmp", then "DrawBmp" then "Flush". Because we are giving our segments a name when we flush them, we can easily delete each segment when we are ready to flush a new one. This conserves memory. Each flush operation consumes memory, but that memory can be released by deleting the flushed segments with "delsegment".

```
Sub DoFlush
    #desk.g "delsegment flushMe"
    #desk.g "getbmp desk 0 0 ";DisplayWidth;" ";DisplayHeight
    #desk.g "drawbmp desk 0 0; flush flushMe"
    end sub
```

Adding Application Icons

We'll extract icons from the applications the user chooses to place on his desktop with "ExtractIcon". We'll then draw them with "DrawIcon". These functions are explained in issue 134 of The Liberty BASIC Newsletter. We've set up a double dimensioned array to hold information about each application the user adds to his desktop.

```
'apps$(i,1) = filename of exe
'apps$(i,2) = x location on desktop
'apps$(i,3) = y location on desktop
'apps$(i,4) = icon handle
```

We set up event handlers for mouse events on the desktop. When the user clicks the right mouse button, we allow him to add an icon to his desktop. He is first presented with a filedialog to select the application. When he has done this, we add the information about this application to our "apps$()" array. The xy location on the desktop are determined by checking the applications that are already in the array and incrementing the values accordingly. We've chosen to add icons in a row across the top of the desktop. We check to see if the width of the desktop has been reached, and if it has, we drop down to the next row to add the new icon. We extract the icon, draw it on the desktop, and flush the new image. Because we've put the information into an array, it is very easy to iterate through the array, looking for the first empty index, and filling that index with information about the new application being added by the user.

```
Sub AddApp handle$, mx, my
    for index = 1 to appMax
        if apps$(index,1) = "" then
            filedialog "Add App", "*.exe", exe$
```

```
            if exe$ <> "" then
                apps$(index,1) = exe$
                newX = val(apps$(index-1,2)) + gridWidth
'next column in this row
                newY = val(apps$(index-1,3)) 'use same row
                if newX > DisplayWidth - gridHeight then
                    newX = gridWidth         'beginning of new row
                    newY = newY + gridHeight 'move to next row
                end if
                apps$(index,2) = str$(newX)
                apps$(index,3) = str$(newY)
                hIcon = ExtractIcon(hwnd(#desk.g), apps$(index,1))
                apps$(index,4) = str$(hIcon)
                'draw icon on desktop at designated location
                result = DrawIcon(hDC, val(apps$(index,4)), val(
apps$(index,2)), val(apps$(index,3)) )
            end if
            exit for
        end if
    next
    call DoFlush
    if index = appMax then  'all slots are filled
        notice "All slots are filled. Cannot add more apps."
    end if
    end sub
```

Remembering the User's Choices

We'll use an API initialization file to remember and retrieve the user's choices. The functions to do this are explained in detail in issue 143 of the Liberty BASIC Newsletter. When the program closes, we write the information to the ini file by iterating through the apps$() array. The routine is as follows.

```
Sub WriteIniInfo
    for i = 1 to appMax
        if apps$(i,1) = "" then exit for
        call WriteIniFile "My Desktop Shell", "appName"+str$(i),
 apps$(i,1), "mydesktopshell.ini"
        call WriteIniFile "My Desktop Shell", "appX"+str$(i),
 apps$(i,2), "mydesktopshell.ini"
        call WriteIniFile "My Desktop Shell", "appY"+str$(i),
 apps$(i,3), "mydesktopshell.ini"
    next
    end sub
```

Each time the program opens, we'll call a routine that reads the ini file, then displays the user's selections

on the desktop. It places the information into our apps$() array as it is read from the ini file.

```
Sub GetIniInfo
    for i = 1 to appMax
        exe$ = GetIniFile$("My Desktop Shell", "appName"+str$(i),
"none","mydesktopshell.ini")
        if exe$ = "none" then exit for
        appX$ = GetIniFile$("My Desktop Shell", "appX"+str$(i),"0",
"mydesktopshell.ini")
        appY$ = GetIniFile$("My Desktop Shell", "appY"+str$(i),"0",
"mydesktopshell.ini")
        apps$(i,1) = exe$   'full path to app exe
        apps$(i,2) = appX$  'x location on desktop of app icon
        apps$(i,3) = appY$  'y location on desktop of app icon
        'get handle of icon extracted from exe:
        hIcon = ExtractIcon(hwnd(#desk.g), apps$(i,1))
        apps$(i,4) = str$(hIcon)
        'draw icon on desktop at designated location
        result = DrawIcon(hDC, val(apps$(i,4)), val(apps$(i,2)),
val(apps$(i,3)) )
    next
    call DoFlush
    end sub
```

Running Programs from the Shell

We set up a left button double-click event handler in the graphicbox. When the user double-clicks the left mouse button, we check to see if the mouse pointer is on one of the desktop icons. If it is, we use the "run" command to run the application. For more on using the "run" command, see issue 114 of The Liberty BASIC Newsletter.

To determine if the mouse is clicking on one of the desktop icons, we iterate through the array, checking the xy value of each icon against the MouseX and MouseY value. If the mouse pointer X value is at least as large as an icon's X location, but no larger than the X location plus the gridWidth, then we do a similar check for the Y location of that icon compared to the MouseY location. If they match, we run the application associated with that icon.

```
Sub RunApp handle$, mx, my
    print "mx is ";mx
    print "my is ";my
    for i = 1 to appMax
        'if apps$(i,1) = "" then exit for
        x = val(apps$(i,2))
        y = val(apps$(i,3))
```

```
        print "xy is ";x;" ";y
        if (mx>x) and (mx<x+gridWidth) then
            if (mx>y) and (my<y+gridHeight) then
                run apps$(i,1)
                exit for
            end if
        end if
    next
    end sub
```

Enchancements

The demo program included at the end of this article works fine, but it would benefit from some additional features. Feel free to modify the code to create your own shell program! You might want to check issue 137 of The Liberty BASIC Newsletter for information on "Running Control Panet Applets." You might also consider some of these enhancements:

```
'possible enhancements:
'   allow user to remove apps
'   allow user to rearrange icons on desktop
'   add labels below icons (increase gridWidth value)
'   make text for labels have transparent background
'   allow user to create custom folders on this desktop
'   if no icon is extracted, use a default icon
'   allow user to use custom icons, instead of icon extracted from exe
'   enhance exit button with choices, like Windows start button
'   create custom taskbar
```

Making Code Easy to Modify

We've used global variables for some values, so that they are visible inside subs and functions. We've set up variables to hold certain information, so that the code can be modified easily by changing the values of these variables at the top of the code. This allows us to make a single change, instead of going through the entire program code to change many instances of a value. One such variable is called "gridWidth". It contains the value we use to locate the icons in the X direction. It is set to 40. If you want to place the icons closer together, make this number smaller. If you want them further apart, make the number larger. Don't make it smaller than 32, which is the default width for icons, or the icons will overlap one another.

DEMO

```
'Desktop Shell Demo
'if you use this code,
'    please credit Alyce Watson, http://alycesrestaurant.com/
```

```
'note that this demo allows user to add apps to desktop
'note that this demo does not allow user to remove apps or rearrange d
esktop
'possible enhancements:
'   allow user to remove apps
'   allow user to rearrange icons on desktop
'   add labels below icons (increase gridWidth value)
'   make text for labels have transparent background
'   allow user to create custom folders on this desktop
'   if no icon is extracted, use a default icon
'   allow user to use custom icons, instead of icon extracted from exe
'   enhance exit button with choices, like Windows start button
'   create custom taskbar

nomainwin
global hDC, gridWidth, gridHeight, appMax
gridWidth = 40 : gridHeight = 40
appMax = 50 'maximum number of apps on desktop

dim apps$(50,4) 'arrays are global by default
redim apps$(appMax, 4)
'apps$(i,1) = filename of exe
'apps$(i,2) = x location on desktop
'apps$(i,3) = y location on desktop
'apps$(i,4) = icon handle

WindowWidth=DisplayWidth:WindowHeight=DisplayHeight

graphicbox #desk.g, -1, -1, DisplayWidth+2, DisplayHeight+2
button #desk.exit, "Exit", DoExit, UL, 4, DisplayHeight-30, 56, 28

open "My Desktop Shell" for window_popup as #desk
    #desk "trapclose Quit"
    #desk.g "down; setfocus"
    #desk.g "when rightButtonUp AddApp"
    #desk.g "when leftButtonDouble RunApp"

    hDC = GetDC(hwnd(#desk.g))
    call PaintDesktop hDC
    call GetIniInfo 'get and draw user icons, flush desktop
    wait

Sub AddApp handle$, mx, my
    for index = 1 to appMax
        if apps$(index,1) = "" then
            filedialog "Add App", "*.exe", exe$
```

```
            if exe$ <> "" then
                apps$(index,1) = exe$
                newX = val(apps$(index-1,2)) + gridWidth 'next column
in this row
                newY = val(apps$(index-1,3)) 'use same row
                if newX > DisplayWidth - gridHeight then
                    newX = gridWidth         'beginning of new row
                    newY = newY + gridHeight 'move to next row
                end if
                apps$(index,2) = str$(newX)
                apps$(index,3) = str$(newY)
                hIcon = ExtractIcon(hwnd(#desk.g), apps$(index,1))
                apps$(index,4) = str$(hIcon)
                'draw icon on desktop at designated location
                result = DrawIcon(hDC, val(apps$(index,4)), val(apps$(
index,2)), val(apps$(index,3)) )
            end if
            exit for
        end if
    next
    call DoFlush
    if index = appMax then  'all slots are filled
        notice "All slots are filled. Cannot add more apps."
    end if
    end sub

Sub RunApp handle$, mx, my
    for i = 1 to appMax
        x = val(apps$(i,2))
        y = val(apps$(i,3))
        print "xy is ";x;" ";y
        if (mx>x) and (mx<x+gridWidth) then
            if (mx>y) and (my<y+gridHeight) then
                run apps$(i,1)
                exit for
            end if
        end if
    next
    end sub

Sub GetIniInfo
    for i = 1 to appMax
        exe$ = GetIniFile$("My Desktop Shell", "appName"+str$(i),"none
","mydesktopshell.ini")
        if exe$ = "none" then exit for
        appX$ = GetIniFile$("My Desktop Shell", "appX"+str$(i),"0","my
```

```
desktopshell.ini")
        appY$ = GetIniFile$("My Desktop Shell", "appY"+str$(i),"0","my
desktopshell.ini")
        apps$(i,1) = exe$   'full path to app exe
        apps$(i,2) = appX$  'x location on desktop of app icon
        apps$(i,3) = appY$  'y location on desktop of app icon
        'get handle of icon extracted from exe:
        hIcon = ExtractIcon(hwnd(#desk.g), apps$(i,1))
        apps$(i,4) = str$(hIcon)
        'draw icon on desktop at designated location
        result = DrawIcon(hDC, val(apps$(i,4)), val(apps$(i,2)), val(a
pps$(i,3)) )
    next
    call DoFlush
    end sub

Sub WriteIniInfo
    for i = 1 to appMax
        if apps$(i,1) = "" then exit for
        call WriteIniFile "My Desktop Shell", "appName"+str$(i), apps$
(i,1), "mydesktopshell.ini"
        call WriteIniFile "My Desktop Shell", "appX"+str$(i), apps$(i,
2), "mydesktopshell.ini"
        call WriteIniFile "My Desktop Shell", "appY"+str$(i), apps$(i,
3), "mydesktopshell.ini"
    next
    end sub

Sub DoFlush
    #desk.g "delsegment flushMe"
    #desk.g "getbmp desk 0 0 ";DisplayWidth;" ";DisplayHeight
    #desk.g "drawbmp desk 0 0; flush flushMe"
    end sub

Sub Quit handle$
    call ReleaseDC hwnd(#desk.g), hDC
    call WriteIniInfo
    for i = 1 to appMax
        if apps$(i,1) = "" then exit for
        hIcon = val(apps$(i,4))
        calldll #user32, "DestroyIcon", hIcon as ulong, re as long
    next
    close #handle$:end
    end sub

Sub DoExit handle$
```

```
    call Quit "#desk"
    end sub

'*************
' API WRAPPERS
' api wrappers are copied from http://alycesrestaurant.com/workshop.ht
m
'*************

sub PaintDesktop hwndDC
    calldll #user32, "PaintDesktop", hwndDC as ulong,_
    re as long
    end sub

Sub WriteIniFile lpAppName$, lpKeyName$, lpString$, lpFileName$
    lpFileName$=DefaultDir$+"\"+lpFileName$
    CallDLL #kernel32, "WritePrivateProfileStringA", _
        lpAppName$ As ptr, _    'section name
        lpKeyName$ As ptr, _    'key name
        lpString$ As ptr, _     'key value
        lpFileName$ As ptr, _   'ini filename
        result As long       'nonzero = success
    end sub

Function GetIniFile$(lpAppName$, lpKeyName$,lpDefault$,lpFileName$)
    lpFileName$=DefaultDir$+"\"+lpFileName$
    nSize=100
    lpReturnedString$=Space$(nSize)+Chr$(0)
    CallDLL #kernel32, "GetPrivateProfileStringA", _
        lpAppName$ As ptr, _'section name
        lpKeyName$ As ptr, _'key name
        lpDefault$ As ptr, _'default string returned if there is no en
try
        lpReturnedString$ As ptr, _ 'destination buffer
        nSize As long, _            'size of destination buffer
        lpFileName$ As ptr, _       'ini filename
        result As ulong     'number of characters copied to buffer

    GetIniFile$=Left$(lpReturnedString$,result)
    end function

Function ExtractIcon(hW, file$)
    hInst=GetWindowLong(hW, _GWL_HINSTANCE)
    CallDLL #shell32, "ExtractIconA",hInst as uLong,_
    file$ As Ptr, 0 As Long, ExtractIcon as uLong
    End Function
```

```
Function DrawIcon(hdc,hIcon,x,y)
    CallDLL #user32, "DrawIcon",hdc as uLong, x As Long,_
    y As Long, hIcon as uLong, DrawIcon As Long
    End Function

Function GetWindowLong(hW, type)
    CallDLL #user32, "GetWindowLongA",hW as uLong,_
    type As Long,GetWindowLong As Long
    End Function

Function GetDC(hWnd)
    CallDLL #user32, "GetDC",hWnd as uLong,GetDC as uLong
    End Function

Sub ReleaseDC hWnd, hDC
    CallDLL#user32,"ReleaseDC",hWnd as uLong,_
    hDC as uLong,result As Long
    End Sub
```

Table of Contents