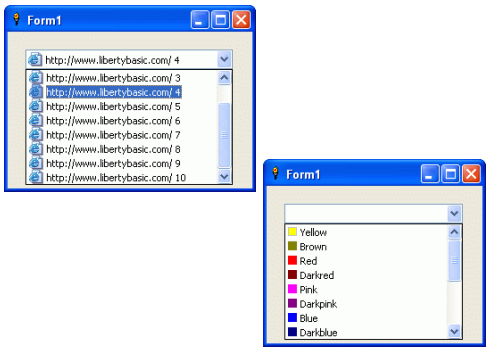

Back when Internet Explorer 3.0 was released a combobox control became available that has the ability to display images. The aim of this article is to show how to use one of these extended comboboxes, called ComboBoxEx, in a Liberty BASIC application *without using a third party DLL, timer or callback.*

Since this control is not native to Liberty BASIC it must be created and managed through the Windows API. This is not difficult but it does impose a certain programming style for the application since some method to monitor the user's actions with the control must be used. One way to monitor the control is to run the application in a scan loop and constantly check for keyboard and mouse input. This is the technique used in this article. If this method is used, the *wait* statement cannot be used anywhere in the application.

This article walks through the steps and code needed to implement and support the ComboBoxEx in LB. Note: the ComboBoxEx control in this article is also referred to as a combobox. This code is not a complete program so please download the zip file at the end of this page for the source code.

Code

To begin with, the ComboBoxEx requires a COMBOBOXEXITEMA struct to supply information about each item in its dropdown list.

```
'COMBOBOXEXITEMA
Struct cbi,
    mask As Ulong, _
    iItem As Long, _
    pszText As Ptr, _
    cchTextMax As Long, _
    iImage As Long, _
    iSelectedImage As Long, _
    iOverlay As Long, _
    iIndent As Long, _
    lParam As Ulong
```

We want this combobox to display both text and images so the mask element of the struct must be set with the value that specifies this.

```
cbi.mask.struct = 7 'CBEIF_TEXT or CBEIF_IMAGE or CBEIF_SELECTEDIMAGE
```

Two more struct elements can be set at this time.

```
cbi.cchTextMax.struct = 100
'Limits the list items to 100 characters each.
cbi.iItem.struct = -1
'Items will be added to the end of the list.
```

The next requirement is to load the ComboBoxEx32 class by calling the InitCommonControlsEx API function. Again a small struct is required.

```
'INITCOMMONCONTROLSEX
Struct iccex, _
    dwSize As Ulong,_
    dwICC As Ulong

'Indicate the size of this struct.
iccex.dwSize.struct = Len(iccex.struct)

'Specify the ComboBoxEx32 class.
iccex.dwICC.struct = hexdec("200") 'ICC_USEREX_CLASSES.

'Load ComboBoxEx32 class.
Calldll #comctl32, "InitCommonControlsEx", iccex As Struct, r As Long
```

The combobox has one more requirement. It needs an Image List containing the image or images that will be used. You can create your own bitmap or icons and create an image list for these. This article takes the easy route and uses the Windows system image list. The code download that accompanies this article contains a second program that shows how to create an image list and use your own bitmap for the images.

The combobox in this code will display urls like Internet Explorer and the icon associated with urls is contained in the system image list. All that needs to be done to use this icon is to discover it's position within the image list, and to get the handle of the image list. Windows provides a function called SHGetFileInfo that will do this. Another struct is required for this too.

```
'SHFILEINFO
Struct shfi, _
    hIcon As Ulong, _
```

```
    iIcon As Long, _
    dwAttributes As Ulong, _
    szDisplayName As Char[260], _
    szTypeName As Char[80]

'Get the size of this struct.
lenSHFI = Len(shfi.struct)
```

SHGetFileInfo will need to know what kind of icon is wanted so the flags must be set for this. The flags are SHGFI_USEFILEATTRIBUTES or SHGFI_SYSICONINDEX or SHGFI_ICON or SHGFI_SMALLICON and this value is 16657.

With these flags SHGetFileInfo will find the proper icon if you tell it the file extension that you are looking for. In this case it is *.url.

```
Calldll #shell32, "SHGetFileInfo", "*.url" As Ptr, 0 As Long, _
    shfi As Struct, lenSHFI As Long, 16657 As Ulong
, hSysImgList As Ulong
```

If the function call succeeds the url icon index is returned in shfi.iIcon.struct and the handle to the image list is returned in hSysImgList.

After opening the application window the combobox is created with the CreateWindowExA API function.

```
hWndParent = hwnd(#main)
Calldll #user32, "GetWindowLongA", hWndParent As Ulong, _
    _GWL_HINSTANCE As Long, hInst As Ulong
class$ = "ComboBoxEx32"
style = _WS_VISIBLE or _WS_CHILD or _WS_VSCROLL or
 _CBS_AUTOHSCROLL or _CBS_DROPDOWN

Calldll #user32, "CreateWindowExA", 0 As Long, class$ As Ptr, _
    "" As Ptr, style As Long, 20 As Long, 20 As Long, _
    400 As Long, 150 As Long, hWndParent As Ulong, 0 As Long, _
    hInst As Ulong, 0 As Long, hCbe As Ulong
'hCbe contains the handle of the new combobox
```

The next step is to tell the combobox which image list to use by sending a message with the proper command and the handle to the system image list that was obtained earlier.

```
CBEM.SETIMAGELIST = (_WM_USER+2)
Calldll #user32, "SendMessageA", hCbe As Ulong, CBEM.
SETIMAGELIST As Long, _
```

```
    0 As Long, hSysImgList As Ulong, r As Long
```

The last step is to load the combobox with the text to display and the image to go along with it. A different image may be specified for each item in the list. The index of the url icon is used to specify the image in cbi.iImage.struct, and also in cbi.iSelectedImage.struct. This next code loads 10 url strings and sets the image for each one.

```
CBEM.INSERTITEMA = (_WM_USER+1)
For i = 1 To 10
    'Set the image for this item to the url icon.
    cbi.iImage.struct = shfi.iIcon.struct
    cbi.iSelectedImage.struct = shfi.iIcon.struct
    'Set the text.
    cbi.pszText.struct = "http://www.libertybasic.com/ "+ str$(i)
    'Add the item to the end of the list.
    Calldll #user32, "SendMessageA", hCbe As Ulong, _
    CBEM.INSERTITEMA As Long, 0 As Long, cbi As Struct, r As Long
Next i
```

At this point the combobox has been created and displays the url image for each item in the dropdown list. It would be nice if this was all there was to it but we still need to know when an item is selected from the list and when the user types in a new url and presses the Enter key. A ComboBoxEx is actually a window that contains three controls, a regular combobox along with it's list and edit control. In order to monitor the user's actions with the combobox you need to have the handles to the edit control and the dropdown list. The Windows API provides a function named GetComboBoxInfo that is supposed to give these handles to you. However, in my experience it often fails to return the handle of the edit control when used with an extended combobox. The following sub uses a combination of functions to get the job done.

```
Sub GetCboExInfo hCboEx32, Byref hwndEdit, Byref hwndList
    'In: hCboEx32. The handle of a ComboBoxEx.
    'Out: The edit control and listbox handles of a ComboBoxEx.
    struct cboInfo, cbSize As Long, pad As Char[44], hwndList As Long
    cboInfo.cbSize.struct = Len(cboInfo.struct)
    'Get the handle of the combobox in hCboEx32.
    Calldll #user32, "GetWindow", hCboEx32 As Ulong
, _GW_CHILD As long, hwndCombo As Ulong
    'Get the handle of the edit control in hwndCombo.
    Calldll #user32, "GetWindow", hwndCombo As Ulong
, _GW_CHILD As Long, hwndEdit As Ulong
    'Get the handle of the listbox in hwndCombo.
    Calldll #user32, "GetComboBoxInfo", hwndCombo As Ulong
, cboInfo As Struct, r As Long
    hwndList = cboInfo.hwndList.struct
end sub
```

Just call this sub and the edit and list handles are returned in hCbeEdit and hCbeList.

```
'Get the listbox and edit control handles of a ComboBoxEx32.
Call GetCboExInfo hCbe, hCbeEdit, hCbeList
```

Now we have everything needed to use the ComboBoxEx in a Liberty BASIC application except for the code to monitor it for user actions. A pair of subs and a couple of supporting functions handle this quite well. You can throw these away and write your own, but if you look closely at these routines you'll notice that they can be used with a variety of different API created controls with little change. The sub GetEvent is the work horse. It tells you what control the mouse cursor is over when the left mouse button is pressed down, and also when the left mouse button is released after being pressed. It also tells you what control has the focus when the enter key is pressed and also when it's released after being pressed. It's very fast since it only uses one API function to get every virtual keystate in the whole keyboard. Two additional supporting functions are also used, GetMouseFocus() and GetFocus(). To make it easy to add additional keystate information a struct is used for returned values instead of global variables or Byref arguments.

```
Struct kb,_
    hDn As Ulong,_
'Handle of window or control with focus when a key down occurs, or
    _
'whose client area is under the mouse cursor when a left button
    _                    'down occurs.
    hUp As Ulong,_       'Same as above, but when a release occurs.
    lBtnDn As Short,_    'Left mouse button is down.
    lBtnClick As Short,_
'Left mouse button has been clicked or pressed and released.
    retDown As Short,_   'Enter key is down.
    retUp As Short       'Enter key has been pressed and released.



Sub GetEvent 'Returns information to kb.struct.

'Left mouse: button down, button click, handle of window whose client
    'area is under the mouse cursor.

'Enter key:  key down, key up (after key down), handle of window with
focus.
    keyBuf$ = Space$(256)
    Calldll #user32, "GetKeyboardState", keyBuf$ As Ptr, r As long
    '_VK_LBUTTON
    If Not(kb.lBtnDn.struct) Then
        If Asc(Mid$(keyBuf$,2))>127 Then
            kb.lBtnDn.struct = 1: kb.hDn.struct = GetMouseFocus()
```

```
        End If
    Else
        If Asc(Mid$(keyBuf$,2))<127 Then
            kb.lBtnClick.struct = 1: kb.lBtnDn.struct = 0: kb.hUp.
struct = GetMouseFocus()
        End If
    End If

    '_VK_RETURN
    If Not(kb.retDown.struct) Then
        If Asc(Mid$(keyBuf$,14))>127 Then
            kb.retDown.struct = 1: kb.hDn.struct = GetFocus()
        End If
    Else
        If Asc(Mid$(keyBuf$,14))<127 Then
            kb.retUp.struct = 1: kb.retDown.struct = 0: kb.hUp.
struct = GetFocus()
        End If
    End If
End Sub
```

This next sub examines the event information collected by GetEvent and acts upon it.
This sub can also be modified to handle events for different API created controls. Only the events for the combobox are handled in this code.

```
Sub HandleEvent 'Uses kb.struct for arguments.
    Select Case
        Case kb.lBtnClick.struct
            If kb.hDn.struct = hCbeList Then
                'allow time for the combobox to update it's display.
                s = Time$("ms"): While Time$("ms") < s+50: Wend
                Print GetWindowText$(hCbe,255)
            End If
            kb.lBtnClick.struct = 0: kb.hDn.struct = 0: kb.hUp.
struct = 0

        Case kb.retUp.struct
            If kb.hUp.struct = hCbeEdit Then
                txt$ = GetWindowText$(hCbe,255)
                If txt$ <> "" Then Print "Enter pressed -> ";txt$
            End If
            kb.retUp.struct = 0: kb.hDn.struct = 0: kb.hUp.struct = 0
    End Select
End Sub
```

GetEvent and HandleEvent are the routines to call in your application loop in order to monitor the mouse and keyboard events for the ComboBoxEx.

This next function is used by GetEvent. It's purpose is to return the handle of the window whose *client* area is under the mouse cursor when it's called.

```
Function GetMouseFocus()
    'Returns the handle of the window or control whose
    'client area is under the mouse cursor.
    Struct GMFpoint, x As Long, y As Long
    Struct GMFrc, left As Long, top As Long, right As Long
, bottom As Long
    Calldll #user32, "GetCursorPos", GMFpoint As Struct, ret As Void
    X = GMFpoint.x.struct
    Y = GMFpoint.y.struct
    Calldll #user32, "WindowFromPoint", X As Long, Y As Long
, hWnd As Ulong
    Calldll #user32, "ScreenToClient", hWnd As Ulong
, GMFpoint As Struct, ret As Void
    x = GMFpoint.x.struct
    y = GMFpoint.y.struct
    Calldll #user32, "GetClientRect", hWnd As Ulong, GMFrc As
 Struct, r As long
    Calldll #user32, "PtInRect", GMFrc As Struct, x As Long, _
        y As Long, PointOnClient As long
    If PointOnClient Then GetMouseFocus = hWnd
End Function
```

GetFocus(), also used by GetEvent, is just a wrapper for the Win32 function of the same name.

```
Function GetFocus()
    Calldll #user32, "GetFocus", GetFocus As Ulong
End Function
```

That pretty well covers it, other than a couple of function wrappers and the code for the window itself. Download the source and try it for yourself.

Source code

cboex32.zip

- Details

- Download
- 6 KB

For more information on using the ComboBoxEx see http://msdn.microsoft.com/library/default.asp?url=/library/en-us/shellcc/platform/commctls/comboex/comboex.asp or use your favorite search engine and look for "ComboBoxEx Controls".

-
DennisMcK Apr 30, 2006