The ABCs of APIs Lesson 8

Windows API in Liberty BASIC

Table of Contents

DLLs Recognized by Liberty BASIC

Liberty BASIC recognizes most of the standard Windows DLLs automatically. The list is as follows:

- #user32 - *window and control management functions*
- #kernel32 - *memory, computer info, timing, kernel info*
- #gdi32 - *graphics*
- #winmm - *multimedia*
- #shell32 - *Windows shell, executing programs, folder dialogs*
- #comdlg32 - *common dialogs (printer, font, file, color)*
- #comctl32 - *common controls (treeview, progressbar, tabstrip, etc.)*

Because Liberty BASIC recognizes these DLLs, they can be used in **CALLDLL** without being opened first. This example is from Lesson 1 . **#user32** is called with **CALLDLL** to discover if a button is enabled.

```
CallDLL #user32, "IsWindowEnabled",_
   hButton as uLong,_   'handle of window or control
   result As Long       'nonzero = enabled
```

DLLs That Must Be Opened

Any DLL that is not listed above must receive an **OPEN** command before it can be used. It also must be closed with a **CLOSE** command when it is no longer needed, or when the program ends. Lesson 9 will discuss both Windows and add-on DLLs that require "open for DLL".

Windows Constants

Windows Constants are simply numbers. They are used in API calls to send messages. They take this form in the documentation for languages like Visual Basic and C:

SW_HIDE
SW_SHOW

The first two or three letters define a category. In the example above, "SW" stands for "ShowWindow" because these are constants for use in the "ShowWindow" API call. The prefix is followed by an underscore character, then a descriptive word or two.

Windows constants are declared in other languages and given a value. The two constants in the example above have values as follows:

SW_HIDE = 0
SW_SHOW = 5

The actual values can be used in an API call. Constants are used because it is easier for the programmer to remember that "SW_SHOW" will cause a window to be visible, than it is to remember that the number 5 will do it.

Both examples below work in exactly the same way:

```
CallDLL #user32, "ShowWindow",hWnd as uLong, 5 As Long, r As Long
CallDLL #user32, "ShowWindow",hWnd as uLong, _SW_SHOW As Long, r As
Long
```

If you look at the second line in this example, you'll see an additional underscore character prepended to the name of the constant, "_SW_SHOW". See the next section to learn why that is so.

Windows Constants Defined by Liberty BASIC

Many Windows constants are defined by Liberty BASIC. An initial underscore character is prepended to the name of Windows constants to signal Liberty BASIC that this is, in fact, a Windows constant. Liberty BASIC then substitutes the proper value whenever that constant is used in a program.

SW_SHOW
In Liberty BASIC code, is written as:
_SW_SHOW

The following example causes a widow to be minimized with the "ShowWindow" API call. Values for some of the "SW" constants are included in the comments.

```
'SW_HIDE = 0
'SW_NORMAL = 1
'SW_SHOWMINIMIZED = 2
'SW_MAXIMIZE = 3
'SW_SHOWNOACTIVATE = 4
'SW_SHOW = 5
'SW_MINIMIZE = 6
'SW_SHOWMINNOACTIVE = 7
'SW_SHOWNA = 8
'SW_RESTORE = 9

hMain = hwnd(#main)

CallDLL #user32, "ShowWindow",hMain as uLong,
 _SW_MINIMIZE As Long, r As Long
```

The following demo opens a window, then mimizes it.

```
nomainwin
Open "My Window" for window as #main
#main "trapclose [quit]"

hMain = hwnd(#main)

CallDLL #user32, "ShowWindow",_
hMain as uLong,_        'handle of window
_SW_MINIMIZE As Long,_  'message to minimize window
 r As Long

wait
```

```
[quit] close #main:end
```

Additional Windows Constants

Liberty BASIC does not recognize all of the Windows constants. For unrecognized constants, you may use the actual value, as found in the documentation for the API call in question, or you may create your own variable that stands in for a Windows constant. When you create such a variable, do not use an underscore as one of the characters. Liberty BASIC expects variable names containing underscores to be Windows constants that it recognizes. If you use an underscore as part of a variable name, Liberty BASIC will give you an "undefined windows constant" error.

Incorrect format for defining your own Windows constant:

MY_CONSTANT

Some alternatives:

MY.CONSTANT
MyConstant

How can you know which Windows constants are defined by Liberty BASIC? It's easy! Try printing them. The following code generates the "undefined Windows constant" error, because Liberty BASIC does not know the value for this contant.

```
    print _CSIDL_PROGRAMS
```

The following demo creates a variable that stands in for the unrecognized Windows constant and uses it to retrieve the location of the user's program folder.

```
CSIDL.PROGRAMS = 2

struct IDL,cb As Long, abID As short

calldll #shell32, "SHGetSpecialFolderLocation",_
    0 as long, CSIDL.PROGRAMS as long, IDL as struct, ret as long

if ret=0 then
    Path$ = Space$(512)
    id=IDL.cb.struct
    calldll #shell32, "SHGetPathFromIDListA",id as long, Path$ as
```

```
ptr, ret as long
    GetSpecialfolder$ = Left$(Path$, InStr(Path$, Chr$(0)) - 1)
else
    GetSpecialfolder$ = "Error"
end if

print GetSpecialfolder$
```

Here is the same demo, but with one difference. Instead of creating a variable to hold the value of the Windows constant, the value itself is used.

```
struct IDL,cb As Long, abID As short

calldll #shell32, "SHGetSpecialFolderLocation",_
    0 as long, 2 as long, IDL as struct, ret as long

if ret=0 then
    Path$ = Space$(512)
    id=IDL.cb.struct
    calldll #shell32, "SHGetPathFromIDListA",id as long, Path$ as
ptr, ret as long
    GetSpecialfolder$ = Left$(Path$, InStr(Path$, Chr$(0)) - 1)
else
    GetSpecialfolder$ = "Error"
end if

print GetSpecialfolder$
```

Multiple Values and OR

Some API functions include arguments that convey multiple messages in a single argument. You can put multiple Windows constants together to form a single message with the **BITWISE OR** operator. It looks like this:

value = CONSTANT.ONE OR CONSTANT.TWO OR CONSTANT.THREE
value = 4 or 7 or 23

The first method uses named constants, and the second uses actual values. Both methods work in the same way.

The following demo uses multiple values in a single argument. It creates a Windows messagebox. A single argument tells Windows which combination of icons and buttons to use for this messagebox.

Some possible icons and buttons are as follows:

Icon Sets:
_MB_ICONASTERISK (same as _MB_ICONINFORMATION) 'a lower case "i"
_MB_ICONEXCLAMATION 'an exclamation point
_MB_ICONHAND (same as _MB_ICONSTOP) 'circle with X [Win95/98]
_MB_ICONQUESTION 'a question mark

Push Button Sets:
_MB_OK
_MB_OKCANCEL
_MB_YESNO
_MB_YESNOCANCEL
_MB_RETRYCANCEL
_MB_ABORTRETRYIGNORE

To create a messagbox containing the red X icon and buttons with captions "Retry" and "Cancel, put the values together with **OR** like this:

```
wtype = _MB_ICONSTOP or _MB_RETRYCANCEL
```

Note that we've assigned the **OR'd** value to a variable. You cannot use **OR** in an API argument.

Incorrect usage:

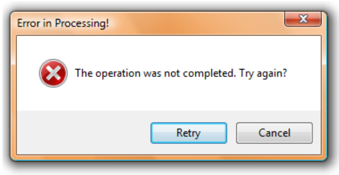
```
calldll #dll, "MyFunction",_
MY.ONE OR MY.TWO as long, result as long
```

Correct usage:

```
value = MY.ONE OR MY.TWO

calldll #dll, "MyFunction",_
value as long, result as long
```

The demo creates a messagebox that looks like this:


Error in Processing!

The operation was not completed. Try again?

```
Title$ = "Error in Processing!"
Message$ = "The operation was not completed. Try again?"
wtype = _MB_ICONSTOP or _MB_RETRYCANCEL

calldll #user32, "MessageBoxExA",_
h as ulong,_          'window handle can be 0
Message$ as ptr,_     'desired message text
Title$ as ptr,_       'desired titlebar caption
wtype as long,_       'flag for icon and buttons
language as word,_    'language identifier
result as long        'returns action code

'possible values for result:
'1 = OK was clicked
'2 = Cancel was clicked
'3 = Abort was clicked
'4 = Retry was clicked
'5 = Ignore was clicked
'6 = Yes was clicked
'7 = No was clicked

if result = 4 then
print "Retrying now."
end if

if result = 2 then
print "Canceling operation."
end if
```

What's Next?

Lesson 9 will discuss both Windows and add-on DLLs that require "open for DLL".

Written by Alyce Watson. For more on APIs, see:
APIs for Liberty BASIC