The ABCs of APIs Lesson 9
DLLs To Be Opened

Table of Contents

In Lesson 8 we discussed the standard DLLs that Liberty BASIC recognizes natively. They can be used by referring to their handles. One example: user32.dll can be called with the handle "#user32".

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.)*

All DLLs that are not on that list must be opened for use and closed when they are no longer needed, with the **OPEN** and **CLOSE** commands. This is true for Windows DLLs not in the list and for third-party DLLs.

DLLs and Open

DLLs can be opened each time one of their API functions is to be called, and closed afterwards, or they can be opened at the start of the program and remain open until the program ends.

The syntax for opening a DLL is as follows:

```
OPEN "filename.dll" for dll as #handle
```

DLLs and Close

DLLs can be opened each time they are to be called and closed afterwards. If many API calls are to be made to a DLL, it can be opened at the start of the program and closed when the program ends. Failure to close a DLL when the program ends generates an error.

The syntax for closing an open DLL is as follows:

```
close #handle
```

Calling Opened DLLs

Once a DLL has been opened with the **OPEN** statement, it is called in exactly the same way as the standard Windows DLLs in earlier lessons in this series.

Here is some sample code that calls on "advapi32.dll". It is part of Windows, but Liberty BASIC does not have a ready-made handle for it, so it must be opened for use. This program retrieves the computer user's username.

```
    'open the DLL
    Open "advapi32" For DLL As #ad

    'create a string buffer
    buf$ = Space$(100 )+ Chr$(0)

    'create a struct to hold the size of the string
    struct size, L As Long
```

```
'fill struct with value for length of string
size.L.struct  = 100

'make API call
CallDLL #ad,"GetUserNameA",_
   buf$ As Ptr,_    'string buffer to receive info
   size As struct,_ 'struct to hold size of returned string
   re As Long

'close the DLL
Close #ad

'the function fills the struct
'with the length of the string it returns
lenReturn = size.L.struct

'truncate string buffer to retrieve username
UserName$=left$(buf$, lenReturn)

'print result
print "Username is ";UserName$
```

Did you notice the use of the struct called "size"? It's used to pass a number byref, as explained in ABCs of APIs Lesson 7 - Passing Numbers by Reference.

Third Party DLLs

There are many add-on DLLs made available by third parties. Such DLLs are called in exactly the same way as Windows DLLs. They must be opened with the **OPEN** statement and given a handle. After this, the functions in the DLLs can be called. When they are no longer needed, they must be closed with the **CLOSE** statement.

Location of DLLS

Windows DLLs are located in the Windows Directory or in the Windows System Directory. Windows knows to look for the DLLs in this directories and in the directory that contains the running program. When you distribute your programs that require an add-on DLL, you can install it in the program's directory or in the Windows or Windows System directory as described in the next section.

Here is a small demo that uses the add-on JPEG.DLL to load images that are not natively supported by Liberty BASIC. Add-on DLLs extend the abilities of Liberty BASIC.

```
'jpeg.dll by Alyce Watson, 2003
nomainwin

filter$="*.jpg;*.bmp;*.ico;*.emf;*.wmf;*.gif"
filedialog "Open Image",filter$,jname$

'if user doesn't choose an image, end program
if jname$="" then end

open "jpeg.dll" for dll as #j

open "JPEG.DLL Test" for graphics_fs as #1
    #1 "trapclose [quit]"
    #1 "down; fill lightgray;flush"
    hW=hwnd(#1)   'graphics window handle

    calldll #j, "LoadImageFile",_
    hW as ulong,_     'graphics window handle
    jname$ as ptr,_   'filename of image
    hPic as ulong     'handle of image in memory

    #1 "cls;fill lightgray"

    'load image with LOADBMP
    loadbmp "demo",hPic
    hDemo=hbmp("demo")

    'display with DRAWBMP
    #1 "drawbmp demo 0 0 ;flush"

wait

[quit]
    unloadbmp "demo"
    close #1:close #j:end
```

The JPEG.DLL can be downloaded at the following link, along with a larger sample program and documentation.

jpegdll.zip

- Details
- Download
- 5 KB

When you use add-on DLLs, you must refer to the documentation provided to discover which functions are available and how to call them. Lesson 10 will discuss the way to translate documentation for DLLs from other languages into Liberty BASIC syntax.

Dynamic-Link Library Search Order

This is an extract from the MSDN Library article Dynamic-Link Library Search Order If the link is broken, search MSDN for "Dynamic-Link Library Search Order."

A system can contain multiple versions of the same dynamic-link library (DLL). Applications can control the location from which a DLL is loaded by specifying a full path, using DLL redirection, or by using a manifest. If none of these methods are used, the system searches for the DLL at load time as described in this topic.

Standard Search Order

1. The directory from which the application loaded.
2. The current directory. (Only in latter versions of Windows and only if SafeDLLSearchMode is disabled. Seach MSDN for more info.)
3. The system directory. Use the GetSystemDirectory function to get the path of this directory.
4. The 16-bit system directory. There is no function that obtains the path of this directory, but it is searched.
5. The Windows directory. Use the GetWindowsDirectory function to get the path of this directory.
6. The directories that are listed in the PATH environment variable. Note that this does not include the per-application path specified by the App Paths registry key.

What's Next?

Lesson 10 will discuss using Winstring() with pointers to text data.

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