Call the Right Function

by Karl E. Peterson

Click & Retrieve Source CODE!

Q Dealing with Versionitis

As time goes on, the capabilities of the machines my applications run on become harder to predict. I'd like to take advantage of new functions offered by the DLLs that ship with Internet Explorer and other Windows updates, but I can do that only if my user has installed the required packages. I've seen folks error-trap DLL calls when it's questionable whether the correct DLL version will exist on the user's machine. Is there a way that's more robust and generic than error trapping to determine whether I can safely make the enhanced calls, or whether "graceful degradation" is the better strategy?

A Yes, there is a better way. Two API functions, rarely useful to VB programmers, come in handy here. Languages that support true function pointers can load a library, retrieve the address of the desired function, then invoke it by address. Although in VB we're left wanting at the critical third step, the first two steps tell you all you need to know.

Use the LoadLibrary API to map a DLL into your process's address space. Then call GetProcAddress, passing the name of the desired routine, to obtain a pointer to that function. If GetProcAddress succeeds, then you know the function is exported and callable from VB (see Listing 1). Don't worry about including Declare statements for API functions that might not be exported, because these functions alone won't generate errors. VB objects only when you attempt to call a nonexistent function.

Interestingly, you'll find that a great many of the common DLLs are already mapped into your process's address space. For this reason, it's not a bad idea to first call GetModuleHandle on the desired library. If this call succeeds, you don't have to make the LoadLibrary call. If GetModuleHandle fails, and you do call LoadLibrary, make sure you call FreeLibrary after determining whether the function you're interested in is exported. If the function you're testing is one that will be called many times, it makes sense to test only once whether it's available, and set a flag accordingly.

VB4/32, VB5, VB6 | Determine Function Availability at Run Time

```
Option Explicit

Private Declare Function GetModuleHandle Lib _
   "kernel32" Alias "GetModuleHandleA" ( _
   ByVal lpModuleName As String) As Long
Private Declare Function LoadLibrary Lib _
   "kernel32" Alias "LoadLibraryA" ( _
   ByVal lpLibFileName As String) As Long
Private Declare Function GetProcAddress Lib _
   "kernel32" (ByVal      hModule As Long, _
   ByVal lpProcName As String) As Long
Private Declare Function FreeLibrary Lib _
   "kernel32" (ByVal hLibModule As Long) As Long

Public Function Exported(ByVal ModuleName As _
   String, ByVal ProcName As String) As Boolean
   Dim hModule As Long
   Dim lpProc As Long
   Dim FreeLib As Boolean

   ' check first to see if the module is already
   ' mapped into this process.
   hModule = GetModuleHandle(ModuleName)
   If hModule = 0 Then
      ' need to load module into this process.
      hModule = LoadLibrary(ModuleName)
      FreeLib = True
   End If

   ' if the module is mapped, check procedure
   ' address to verify it's exported.
   If hModule Then
      lpProc = GetProcAddress(hModule, ProcName)
      Exported = (lpProc <> 0)
   End If

   ' unload library if we loaded it here.
   If FreeLib Then Call FreeLibrary(hModule)
End Function
```

Listing 1 As DLLs are upgraded, new functions are often added. Using this routine, you can quickly determine whether any given function is exported from the DLL found on the machine where your application is running. This strategy is much easier than comparing the DLL's version against known function lists.

VB4/32, VB5, VB6 | Accurately Determine Available Space on Large Drives

```
' used to determine drive free space.
Private Declare Function GetDiskFreeSpace Lib _
   "kernel32" Alias "GetDiskFreeSpaceA" ( _
   ByVal lpRootPathName As String, _
   lpSectorsPerCluster As Long, _
   lpBytesPerSector As Long, _
   lpNumberOfFreeClusters As Long, _
   lpTotalNumberOfClusters As Long) As Long
Private Declare Function GetDiskFreeSpaceEx Lib _
   "kernel32" Alias "GetDiskFreeSpaceExA" ( _
   ByVal lpRootPathName As String, _
   lpFreeBytesAvailableToCaller As Currency, _
   lpTotalNumberOfBytes As Currency, _
   lpTotalNumberOfFreeBytes As Currency) As Long

Public Function GetDriveFreeSpace(Optional ByVal _
   Drive As String = "") As Variant
   ' use current drive if not specified
   If Drive = "" Then Drive = CurDir$

   ' default to zero in case drive is
   ' empty/nonexistent
   GetDriveFreeSpace = CDec(0)

   ' check if enhanced function is available.
   If Exported("kernel32", _
      "GetDiskFreeSpaceExA") Then
      Dim cAvail As Currency
      Dim cTotal As Currency
      Dim cFree As Currency
      ' return available bytes, as that's more
      ' important to know than total free bytes
      ' if they differ.
      If GetDiskFreeSpaceEx(Drive, cAvail, _
         cTotal, cFree) Then
         GetDriveFreeSpace = CDec(cAvail * 10000)
      End If

   Else   ' enhanced function not exported.
      Dim nSecPerClus As Long
      Dim nBytPerSec As Long
      Dim nFreeClus As Long
      Dim nTotalClus As Long
      ' do the math to return total free bytes.
      If GetDiskFreeSpace(Drive, nSecPerClus, _
         nBytPerSec, nFreeClus, nTotalClus) Then
         GetDriveFreeSpace = CDec(nSecPerClus * _
            nBytPerSec * nFreeClus)
      End If
   End If
End Function
```

Listing 2 Microsoft did not offer a reliable API for obtaining available free space on drives larger than 2 GB until Win95/OSR2 and NT4. When you need this value, check whether GetDiskFreeSpaceEx is exported from kernel32.dll, and call it if it's available. To degrade gracefully, be prepared to call the original GetDiskFreeSpace if the enhanced function hasn't been exported.

Q Free Disk Space on Large Drives

I have an older, VB4-era app that uses SetupKit.dll to determine a disk's free space. Now I notice VB6 no longer includes that DLL, and the function that the new setup kit supplies uses a native Win32 API call that fails with an overflow on some, but not all, hard drives. This call works fine on my 4 GB drive, but bombs on my 8 GB drive. I checked the data types and they appear correct. Any ideas?

A It looks like Microsoft needs to update its setup kit code. It's using the GetDiskFreeSpace API, which was never intended to cope reliably with drives larger than 2 GB. Windows NT4 (and Win95/OSR2) introduced the GetDiskFreeSpaceEx API, and this function is what you need to use for larger drives. This is the perfect opportunity to use the export test function discussed in the previous answer, because GetDiskFreeSpaceEx isn't available if your application is running on the original Win95 (see Listing 2).

It's interesting that you mention data types in your question. Today's large hard-drive capacities can definitely overflow VB's signed Long data type. I chose to write the GetDriveFreeSpace function to return a Variant, the only option allowed to take advantage of the Decimal data type. With 28 decimals of precision, the Decimal data type ought to last at least a few years. GetDiskFreeSpaceEx requires pointers to 8-byte variables for several of its parameters. VB's Currency data type works here, but remember to scale the return value up by 10,000.

Another interesting twist offered by GetDiskFreeSpaceEx is that the enhanced function returns both total free space and available free space. The difference is that Windows 2000, when released, will support user-based disk quotas. Never assume you have access to all the free space on a drive—it's time to start checking only for what the current user has been allocated.

Q Display Full-Length Drop-Down

Is there any way to create a ComboBox using the Dropdown List style, which displays all its items in the drop-down without a scrollbar? I'm thinking of something similar to the Months combo in the Windows Calendar applet.

A Yes, but Windows makes it harder than it really should be. The ComboBox's drop-down listbox is by necessity parented to the desktop window; if it weren't, it wouldn't be able to extend beyond the edges of your form. This makes obtaining the list's window handle extremely difficult, especially given that no version of Windows prior to Windows 2000 provided a way to directly uncover the linkage between the list and its associated combo.

ABOUT THIS COLUMN

Ask the VB Pro provides you with free advice on programming obstacles, techniques, and ideas. Read more answers from our crack VB pros on the Web at www.inquiry.com/thevbpro. You can submit your questions, tips, or ideas on the site, or access a comprehensive database of previously answered questions.

Why am I going on about this particular difficulty? You must obtain the list's window handle to resize it by calling the MoveWindow API. When you subclass the combo, one particular message reveals the hWnd for the list. Hook into the combo's message stream, and watch for WM_CTLCOLORLISTBOX. The lParam parameter of this message is the handle you need.

When you receive the WM_CTLCOLORLISTBOX message, use lParam to obtain the coordinates of the list with GetWindowRect, the number of items in the list with SendMessage(LB_GETCOUNT), and the height of each item with SendMessage(LB_GETITEMHEIGHT). Calculate a new height by multiplying the number of items by each item's height, and adding the width of the borders.

The last consideration is the position of the combo itself, because if it's positioned low on the screen, the enlarged list might drop off the screen bottom. Another call to GetWindowRect retrieves the needed coordinates, to which the new height is added, and you can make a comparison against the screen height. Adjust the desired Top coordinate as needed. Finish by calling MoveWindow to reposition the list, and invoking the default window procedure to allow for default processing.

The example I wrote uses native subclassing in VB5, but you can easily alter the code to work with any subclassing control (download Listing 3 and a complete example from the *VBPJ* Web site; see the Download Free Code box for details). VBPJ

About the Author

Karl E. Peterson is a GIS analyst with a regional transportation planning agency and serves as a member of the **Visual Basic Programmer's Journal** Technical Review and Editorial Advisory Boards. Online, he's a Microsoft MVP and a section leader on several **VBPJ** forums. Find more of Karl's VB samples at www.mvps.org/vb.

DOWNLOAD FREE CODE

Download the code for this issue of **VBPJ free** from www.vbpj.com, part of the DevX family of Web sites (www.devx.com).

To get the free code for this entire issue, click on Locator+, the right-most option on the menu bar at the top of the home page, and type **VBPJ0999** into the box. (You first need to register, for free, on DevX.) The free code for this article includes all code listings, plus the CFullDrop class and MHookMe module used to alter the size of a combo drop-down, and drop-in code modules containing the Exported and GetDriveFreeSpace functions shown in Listings 1 and 2.

✪ To get the bonus code for this article, available to DevX Premier Club members, type **VBPJ0999AP** into the Locator+ field. The bonus code includes all the free code described above, plus a project that provides a wide array of information (including available, free, and total space; volume name; serial number; the file system in use; and compression information) about any chosen drive via a CDriveInfo class.