List Your Higher Windows

by Karl E. Peterson and Phil Weber

Click & Retrieve Source CODE!

ENUMERATING WINDOWS

Q I need to list the top-level windows running in the system. I found that EnumWindows API enumerates all top-level windows, but I don't know how to use it. Is it possible to use this API in VB?

A Yes, with the introduction of VB5's AddressOf operator, it's quite possible to use the EnumWindows API as well as its sibling EnumChildWindows, which enumerates all the children of any given window. In fact, using EnumWindows is the preferred way to cycle the window list. Previously, VB programmers could only set up a loop using the GetWindow API. (See Programming Techniques, *VBPJ* September 1995, for an example.)

Although GetWindow worked well, the window list could change during the loop as windows are created or destroyed. Microsoft documentation implies that the window list doesn't change during an EnumWindows callback sequence, which makes EnumWindows safer. EnumWindows works by passing the handle to each top-level window into a callback procedure within your application. The system makes the callback until it has enumerated all windows or until the callback procedure returns False.

My implementation scheme is to wrap all relevant functions within a single standard code module, often making it reusable if my search criteria are generic. Start by adding a few items to the Declarations section. At minimum, you need to add a declare for EnumWindows and a module-level variable to track the sought-after window handle:

```
Private Declare Function EnumWindows Lib "user32" (ByVal lpEnumFunc As Long, _
   ByVal lParam As Long) As Long
Private m_hWnd As Long
```

You'll probably include other variables, such as a string holding the desired window text to guide the callback procedure in its filtering, as well as whatever other API declarations are required to test each enumerated window to see if it meets your criteria:

```
Private Declare Function GetWindowText Lib "user32" Alias "GetWindowTextA" _
   (ByVal hWnd As Long, ByVal lpString As String, ByVal cch As Long) As Long
Private m_AppTitle As String
```

Next, define the "public" interface to your enumeration routine. Write a function that accepts the parameters by which you filter the enumerated windows. For example, let's say you want to find a window that starts with a given string. Your public routine could look like this:

```
Public Function FindAllWindows() As Long
   m_hWnd = 0
   m_AppTitle = Search
   Call EnumWindows(AddressOf EnumWindowsProc, 0&)
   MyFindWindow = m_hWnd
End Function
```

Note that before calling EnumWindows, you first set the module-level filter variables—initializing the window handle (m_hWnd) to zero, and storing the search string so you can read it from the callback. Finally, write the callback procedure. You can

Karl E. Peterson is an independent programming consultant who specializes in ActiveX controls by night and spends his days as a GIS analyst with a regional transportation planning agency. Karl coauthored Visual Basic 4 How-To *from Waite Group Press. Online, he's a Microsoft MVP and a section leader in several* VBPJ *online forums. Find more of Karl's VB samples at http://www.mvps.org/vb.*

Phil Weber is an independent consultant specializing in Visual Basic and Web site development. He is a Microsoft Certified Solution Developer and Product Specialist. Find more of Phil's VB tips on his Web site: http://www.teleport.com/~pweber.

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 http://www.inquiry.com/thevbpro. You can submit your questions, tips, and ideas on the site, or access a comprehensive database of previously answered questions.

name this procedure anything, but the example uses the common name EnumWindowsProc. This procedure does the real work. The system calls EnumWindowsProc once for each top-level window. You can stop the enumeration at any time by returning False:

```
Private Function EnumWindowsProc _
   (ByVal hWnd As Long, _
   ByVal lParam As Long) As Long
   Static WindowText As String
   WindowText = Space$(256)
   If GetWindowText(hWnd, WindowText, _
      Len(WindowText)) Then
      If Instr(WindowText, m_AppTitle) _
         = 1 Then
         m_hWnd = hWnd
      End If
   End If
   EnumWindowsProc = (m_hWnd = 0)
End Function
```

Here the callback procedure calls GetWindowText against each passed window handle. It then checks whether this window text starts with the search string. Your filter might be much more elaborate, but this example demonstrates the point. If the procedure finds the desired window, it uses the module-level variable to store its handle. It sets the return value for EnumWindowsProc to the logical result (m_hWnd = 0), which means that the callback procedure returns True until it finds the window it's looking for, at which point m_hWnd has a value other than zero and the logical test results in False.

But I'm afraid I went further than you asked. If you simply want to build a list of all the top-level windows, rather than a single m_hWnd variable, declare an array of them. With each callback into EnumWindowsProc, stash the new handle in the array:

```
Private m_hWnd() As Long
Private m_Windows As Long
Public Function MyFindWindow _
   (Search As String) As Long
   m_Windows = 0
   Redim m_hWnd(1 to 100) As Long
   Call EnumWindows(AddressOf _
      EnumWindowsProc, 0&)
   ' Do something with the array
End Function
Private Function EnumWindowsProc _
   (ByVal hWnd As Long, _
   ByVal lParam As Long) As Long
   If (m_Windows + 1) > UBound(m_hWnd) _
      Then
      ReDim Preserve m_hWnd _
         (1 To m_Windows + 100)
   End If
   m_Windows = m_Windows + 1
   m_hWnd(m_Windows) = hWnd
   EnumWindowsProc = True
End Function
```

In both these examples, execution doesn't resume after the EnumWindows call until all top-level windows have been enumerated or—as in the first example—the filtering criteria have been met. On occasion, you can pass a value into each EnumWindowsProc by specifying an lParam in the EnumWindows call. This value can be a flag that instructs EnumWindowsProc how to deal with the incoming window handles, or just about anything you'd find handy, potentially eliminating one module-level variable. —*K.E.P.*

SOUNDS COMPLICATED

Q I'm trying to get the Media Control Interface (MCI) control to play a WAV file without user intervention. I can't find a way to get it to play without clicking on the Play button on the control. I've tried calling the PlayClick event procedure, but it doesn't work. How do I get this to work?

A Bail on the control altogether. Seriously. Playing a WAV file involves a grand total of one API call. You can wrap it up in a nice, simple function quite cleanly (see Listing 1). The PlaySound function requires three parameters—the file name, an instance handle that isn't used except with resource files, and a flag that tells it you're requesting a file to be played. Optionally, you can play the sound asynchronously by tossing in a SND_ASYNC on the flag parameter, or you can let the sound finish before execution resumes by simply using SND_FILENAME.

But wait, there's more! With a few more lines of code, you can also play any of the predefined system sounds. You used to be able to find these in Win.ini, but they are now located in the registry. Take a look at \HKEY_CURRENT_USER\AppEvents\EventLabels to get a sense of the possibilities.

VB4 32-bit VB5

```
Private Declare Function PlaySound Lib "winmm.dll" _
  Alias "PlaySoundA" (ByVal lpszName As String, _
  ByVal hModule As Long, ByVal dwFlags As Long) As Long
Private Const SND_ASYNC = &H1
Private Const SND_ALIAS = &H10000
Private Const SND_FILENAME = &H20000
Public Enum SystemSounds
  ssSystemAsterisk = 0
  ssSystemExclamation = 1
  ssSystemExit = 2
  ssSystemHand = 3
  ssSystemQuestion = 4
  ssSystemStart = 5
End Enum
Public Sub PlaySoundFile(ByVal FileName As String, _
  Optional ByVal Wait As Boolean = False)
  If Wait Then
    Call PlaySound(FileName, 0&, SND_FILENAME)
  Else
    Call PlaySound(FileName, 0&, SND_ASYNC Or _
      SND_FILENAME)
  End If
End Sub

Public Sub PlaySoundSystem(ByVal WhichSound As _
  SystemSounds)
  Dim SoundAlias As String
  Select Case WhichSound
    Case ssSystemAsterisk
      SoundAlias = "SystemAsterisk"
    Case ssSystemExclamation
      SoundAlias = "SystemExclamation"
    Case ssSystemExit
      SoundAlias = "SystemExit"
    Case ssSystemHand
      SoundAlias = "SystemHand"
    Case ssSystemQuestion
      SoundAlias = "SystemQuestion"
    Case ssSystemStart
      SoundAlias = "SystemStart"
    Case Else 'play default sound
      SoundAlias = "Gobbledygook"
  End Select
  Call PlaySound(SoundAlias, 0, SND_ASYNC Or _
    SND_ALIAS)
End Sub
```

LISTING 1 ***Name That Tune.*** *You must wonder why anyone bothered to write a control around playing WAV files, because playing them is so simple. These two routines demonstrate how to play any WAV file by name, or any system-defined sound by alias. You can extend the Select Case block considerably by scouring the registry for more system sound names. To use these routines in VB4, replace the Enum with hard-coded constants.*

VB4 32-bit VB5

```
Public Function PageSetup(ByVal hWndOwner As Long, _
  rtMargin As RECT) As Boolean
  Dim iNull As Integer
  Dim lpDM As Long
  Dim lResult As Long
  Dim sDevName As String
  Dim prn As Printer
  Dim dm As DEVMODE
  Dim psd As PSDTYPE
  ' Initialize Page Setup dialog with
  ' values for system default printer
  With psd
     .lStructSize = Len(psd)
     .flags = PSD_RETURNDEFAULT
  End With
  lResult = PageSetupDlg(psd)
  If lResult Then
     ' Display Page Setup dialog
     With psd
        .hWndOwner = hWndOwner
        .flags = PSD_INTHOUSANDTHSOFINCHES
     End With
     lResult = PageSetupDlg(psd)
     If lResult Then
        ' Copy API's DevMode structure into VB variable
        lpDM = GlobalLock(psd.hDevMode)
        Call RtlMoveMemory(dm, ByVal lpDM, Len(dm))
        lpDM = GlobalUnlock(psd.hDevMode)

        ' Get device name from DevMode
        sDevName = dm.dmDeviceName
        iNull = InStr(sDevName, vbNullChar)
        If iNull Then
           sDevName = Left$(sDevName, iNull - 1)
        End If
        ' Search Printers collection for
        ' matching device name
        For Each prn In Printers
           If prn.DeviceName = sDevName Then
              Set Printer = prn
              Exit For
           End If
        Next
        ' Set Printer's properties to match
        ' user's selections in dialog
        With Printer
           .PaperSize = dm.dmPaperSize
           .PaperBin = dm.dmDefaultSource
           .Orientation = dm.dmOrientation
        End With
        ' Copy desired margins into rtMargin
        rtMargin = psd.rtMargin
     End If
  End If
  PageSetup = lResult
End Function
```

LISTING 2 ***Use Common Page Setup Dialog.*** *Windows 95 and NT provide a common dialog for setting page properties, such as margins, orientation, and paper size. This function displays the dialog and sets the properties of VB's Printer object according to the user's selections. Due to space limitations, this listing omits the API function declarations and constant definitions. The complete code is available on the DevX Web site (http://www.windx.com); see the Code Online box at the end of this column for details.*

Realize that not all sounds are likely to be on all machines, but most of the common ones are. To play a system sound, pass the name found in the registry, and the flag SND_ALIAS. Toss in a SND_ASYNC if it suits you.

If you happen to pass an alias that isn't registered, the system default sound plays. This is the same sound VB's Beep statement causes. —*K.E.P.*

Q USE STANDARD PAGE SETUP

The WordPad utility in Windows95 and NT includes a cool age Setup dialog that allows the user to set margins, paper size, orientation, and other properties. Is there any way I can use this dialog in my VB programs?

A You betcha! You can call the PageSetupDlg API function display the dialog. That's the easy part; the challenge is to get the user's choices after the user closes the dialog. The PAGESETUPDLG structure that you pass to the function includes a field named hDevMode. This is a handle to a DEVMODE structure that contains the user's choices. To access this structure, pass the hDevMode handle to the GlobalLock function to get a pointer. Then call the RtlMoveMemory function to copy the data from this address into a VB variable. Finally, call GlobalUnlock to unlock the memory containing the original DEVMODE structure.

I've written a handy PageSetup function to handle all these details for you (see Listing 2). The function accepts the hWnd of whatever window you want to own the Page Setup dialog—you may pass zero in this parameter if you don't want the dialog to "belong" to any particular window—and a RECT structure that returns the user's margin settings in thousandths of an inch. The PageSetup function displays the Page Setup dialog, and if the user clicks on the OK button, the function retrieves the user's choices and applies them to VB's Printer object. All you need to do then is set the Printer's CurrentX and CurrentY properties to match the margins in the RECT structure and print away. —*P.W.*

CORRECTION

The April 1998 Ask the VB Pro column incorrectly implied that if you use Microsoft's Axdist.exe to install Wininet.dll along with your application, you can use the InternetAutodial and InternetAutodialHangup functions to initiate and terminate an Internet connection programmatically.

Unfortunately, Axdist.exe installs the Internet Explorer 3.0 version of Wininet.dll, but the InternetAutodial and InternetAutodialHangup functions—as well as the InternetGetConnectedState function discussed in the May 1998 Ask the VB Pro column—were not introduced until IE4. This means an app cannot use these functions unless the user has IE4, Windows 98, or NT5 installed on his or her machine.

If you don't want your app to require IE4, you can use the Remote Access Services (RAS) API to programmatically connect to and disconnect from the Internet. You can call the RAS functions directly from VB, as demonstrated in the Microsoft Knowledge Base (http://support.microsoft.com/download/support/mslfiles/vb32ras.exe), or use a third-party control, such as Mabry Software's RAS Dialer Control (ftp://ftp.mabry.com/ras.exe), to simplify the process. ☒

Code Online

You can find all the code published in this issue of* VBPJ *on The Development Exchange (DevX) at http://www.windx.com. For details, please see "Get Extra Code in DevX's Premier Club" in Letters to the Editor.

List Your Higher Windows Locator+ Codes

Listings for the entire issue, plus the complete code modules for the PageSetup and PlaySound functions (free Registered Level): VBPJ0698

★ Listings for this article only, plus the code listed above, as well as a complete sample app demonstrating custom FindWindowPartial and AppActivePartial routines used for finding and activating a window whose captions start with or contain a given string, with a reusable FindPart.bas module (subscriber Premier Level): AP0698