Spool, Shell, and Hook

by Karl E. Peterson and Phil Weber

Click & Retrieve Source CODE!

Q SENDING PRINT JOBS DIRECTLY TO SPOOLER

I need to send print jobs as preformatted disk images directly to Windows' print spooler, bypassing both VB's Printer object and the printer driver entirely. My app generates and stores PostScript output, but I'd like to offer users the opportunity to spool any job already created. How can I output directly to a printer under Win32 without shelling and issuing a DOS copy command?
—*Anonymous, received by e-mail*

A

Microsoft's Knowledge Base contains an article (Q119113) that offers three methods to send data or a file directly to printers, bypassing the printer driver. Unfortunately, one method doesn't work with all printers, another doesn't work under Windows NT, and the third doesn't work under Windows 95 *or* NT. We'll show you a series of Win32 API calls that, when used together, lets you print directly under any version, with any printer (see Listing 1). Although this might look like a lot of API calls for what should be a simple procedure, it's a logical approach.

To call the SpoolFile routine, simply pass the names of the file to print and which printer to spool to. Optionally, include an AppName variable to help identify the source of the print job if viewed from the Windows print queue. The printer name parameter accepts the same strings returned by the DeviceName property of VB's Printer object. A sample app demonstrating the spooling of any disk-based print file to any installed printer is available on the free, Registered Level of The Development Exchange (see the Code Online box at the end of the column for details).

Your app must make three API calls to prepare the printer to accept direct output. OpenPrinter tells Windows which printer you want to print to, and it returns a printer handle to use in upcoming calls. StartDocPrinter passes a name to display in the print queue viewer, and it specifies that you send raw data directly to the printer. Finally, StartPagePrinter informs the spooler that a "page" is about to be sent. Your data can actually contain multiple pages, so this call is simply a formality that prepares the spooler to accept data.

Now you can call WritePrinter with whatever data you need to send. Call WritePrinter as many times as needed to finish the job. In the sample, a loop reads 16K blocks of data from the file and passes them to WritePrinter. The loop must not read past the end of the file because it will spool random garbage. To prevent that, adjust the buffer size for the last pass if the file isn't an even multiple of 16K bytes.

Shut down the spooling process by reversing the startup. Finish the job with successive calls to EndPagePrinter, EndDocPrinter, and ClosePrinter, and signal the spooler to begin downloading data to the printer. Always keep in mind that by taking this route, *you assume all responsibility of the printer driver.* Your output must conform precisely to the language used by the printer being spooled to.

Q WHAT RETURNS FROM A 32-BIT VB SHELL?

I'm trying to determine the handle of the startup window in a 32-bit VB4 app that another VB4 app is executing using the Shell command. I have read numerous resources on doing this, and I am now completely confused between IDs and handles, and between processes, threads, instances, and tasks.

The 16-bit environment was simpler because it always returned an instance ID, and you could find the corresponding window handle with that Instance ID by looping through all top-level windows using the GetWindow API and the GetWindowWord API with the GWW_INSTANCE parameter. However, VB4/32 introduces concepts such as threads and processes, making it much more complex.

Karl E. Peterson is a GIS analyst with a regional transportation planning agency. Based in Vancouver, Washington, he's also an independent programming consultant specializing in ActiveX controls and contributes to various journals. Online, he's a Microsoft MVP and a section leader in both VBPJ *online forums. Find more of Karl's samples and tips on the Web 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 at 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, or ideas on the site, or access a comprehensive database of previously answered questions.

What exactly does the 32-bit VB Shell return—an ID or a handle? If it's an ID, is it for a process, a thread, or a task? If it's a handle, is it for a process or an instance? Is there a definitive code segment I could use to get a window handle from a shelled app in 32-bit VB? I tried the 16-bit technique and, of course, it doesn't work.
—Mark Cohen, Winnipeg, Manitoba, Canada

A As you've discovered, Win32 ain't Kansas anymore! 32-bit versions of Visual Basic return a Process ID from the Shell function. You can still use a routine much like the one you used in Win16, but you need to call the GetWindowThreadProcessID API while looping through the top-level windows in Win32.

Start by shelling the desired application. Then, depending on whether you're coding in VB4 or VB5, you can take two different approaches. To use the approach that works in both VB4 and VB5, begin by calling FindWindow, passing NULL as both the class name and window title (see Listing 2). This fairly unknown trick obtains the handle of the first window following the desktop in the window list. Then enter a loop, checking each window against your criteria. In this case, calling GetWindowThreadProcessID returns the Process ID for each window, and compares that value to what Shell returned. Although it might not always be necessary, I toss in a test to ensure that each considered window is parentless. Visual Basic has been known to use some bizarre parent-child-owner relationships in the windows it creates.

If you're coding in VB5, you might want to acquaint yourself with the EnumWindows API. Because it requires a function pointer to call back into, this API was previously off limits to VB programmers. However, Microsoft says that calling EnumWindows is preferable to a GetWindow loop, because you're "guaranteed" that the window list doesn't change while iterating. I've presented the VB4 approach here for maximum portability, but I highly recommend using EnumWindows if you're coding in VB5 exclusively (see Keith Pleas's "Drilling Down on VB 5.0," in the May 1997 issue of *VBPJ*).

Due to space limitations, I omitted the API and constant declarations from the code shown in Listing 2. To obtain a complete listing, download a demo applet that shows four different Win32 Shell techniques from the free, Registered Level of DevX. One technique is the routine presented here, while the other three are new twists on the old "shell and wait" problem.

VB4 32-bit VB5

```
Private Declare Function OpenPrinter Lib _
  "winspool.drv" Alias "OpenPrinterA" (ByVal _
  pPrinterName As String, phPrn As Long, pDefault As _
  Any) As Long
Private Declare Function StartDocPrinter Lib _
  "winspool.drv" Alias "StartDocPrinterA" (ByVal hPrn _
  As Long, ByVal Level As Long, pDocInfo As _
  DOC_INFO_1) As Long
Private Declare Function StartPagePrinter Lib _
  "winspool.drv" (ByVal hPrn As Long) As Long
Private Declare Function WritePrinter Lib _
  "winspool.drv" (ByVal hPrn As Long, pBuf As Any, _
  ByVal cdBuf As Long, pcWritten As Long) As Long
Private Declare Function EndPagePrinter Lib _
  "winspool.drv" (ByVal hPrn As Long) As Long
Private Declare Function EndDocPrinter Lib _
  "winspool.drv" (ByVal hPrn As Long) As Long
Private Declare Function ClosePrinter Lib _
  "winspool.drv" (ByVal hPrn As Long) As Long

Private Type DOC_INFO_1
  pDocName As String
  pOutputFile As String
  pDatatype As String
End Type

Public Sub SpoolFile(sFile As String, PrnName As _
  String, Optional AppName As String = "")
  Dim hPrn As Long
  Dim Buffer() As Byte
  Dim hFile As Integer
  Dim Written As Long
  Dim di As DOC_INFO_1
  Dim i As Long
  Const BufSize As Long = &H4000

  ' Extract filename from passed spec, and build job
  ' name. Fill remainder of DOC_INFO_1 structure.
  If InStr(sFile, "\") Then
     For i = Len(sFile) To 1 Step -1
        If Mid(sFile, i, 1) = "\" Then Exit For
        di.pDocName = Mid(sFile, i, 1) & di.pDocName
     Next i
  Else
     di.pDocName = sFile
  End If
  If Len(AppName) Then
     di.pDocName = AppName & ": " & di.pDocName
  End If
  di.pOutputFile = vbNullString
  di.pDatatype = "RAW"

  ' Open printer for output to obtain handle.
  ' Set it up to begin recieving raw data.

  Call OpenPrinter(PrnName, hPrn, vbNullString)
  Call StartDocPrinter(hPrn, 1, di)
  Call StartPagePrinter(hPrn)

  ' Open file and pump it to the printer.

  hFile = FreeFile
  Open sFile For Binary Access Read As hFile
     ' Read in 16K buffers and spool.
     ReDim Buffer(1 To BufSize) As Byte
     For i = 1 To LOF(hFile) \ BufSize
        Get #hFile, , Buffer
        Call WritePrinter(hPrn, Buffer(1), _
           BufSize, Written)
     Next i
     ' Get last chunk of file if it doesn't
     ' fit evenly into a 16K buffer.

     If LOF(hFile) Mod BufSize Then
        ReDim Buffer(1 To (LOF(hFile) Mod _
           BufSize)) As Byte
        Get #hFile, , Buffer
        Call WritePrinter(hPrn, Buffer(1), _
           UBound(Buffer), Written)
     End If
  Close #hFile

  ' Shut down spooling process.
  Call EndPagePrinter(hPrn)
  Call EndDocPrinter(hPrn)
  Call ClosePrinter(hPrn)
End Sub
```

LISTING 1 ***Spool Files and Data Directly to a Printer.*** *Adding these declares and code to a standard module in your 32-bit VB app gives it the ability to spool any disk-based file directly to a printer. You can use the same technique to send any sort of data directly, by simply altering what you pass with the WritePrinter calls. Be aware that by using this technique, you assume all the responsibility of a printer driver, in that the data must be formatted precisely for whatever printer you're sending to.*

Q RESIZING A COMBO DROP-DOWN

Is there any way to specify how many items are visible in the list portion of a drop-down combo box? VB seems to display either eight items or the actual number of items, whichever is less. In Microsoft Access, combo boxes have a ListRows property that allows you to set the maximum number of visible items, but I can't find a corresponding property in VB.
—*Robert Stockdale, received by e-mail*

A Unfortunately, VB offers no simple way to change the height or width of a combo box's drop-down list. You could use a third-party control that offers this feature, but what fun is that? As with most challenges in programming, there *is* a way to do what you want; it's just a question of how dirty you want to get your hands.

Combo boxes consist of up to three separate windows: the combo box itself, an edit control, and a list box. Combo boxes of Style 2, vbComboDropdownList, don't have an associated edit control, because they don't allow editing. Users must select an item from the list. If you could get the window handle of the list box portion, you could easily resize it with a call to the MoveWindow or SetWindowPos API. But the handle returned by a drop-down combo's hWnd property is that of the combo box itself; the list box is a different window.

VB4 32-bit VB5

```
Public Function hWndShell(ByVal JobToDo As String, _
  Optional ExecMode) As Long
  Dim ProcessID As Long
  Dim PID As Long
  Dim hProcess As Long
  Dim hWndJob As Long

  ' Shells a new process and returns the hWnd
  ' of its main window.
  If IsMissing(ExecMode) Then
     ExecMode = vbMinimizedNoFocus
  Else
     If ExecMode < vbHide Or ExecMode > _
        vbMinimizedNoFocus Then
        ExecMode = vbMinimizedNoFocus
     End If
  End If

  On Error Resume Next
     ProcessID = Shell(JobToDo, CLng(ExecMode))
     If Err Then
        hWndShell = 0
        Exit Function
     End If
  On Error GoTo 0

  hWndJob = FindWindow(vbNullString, vbNullString)
  Do While hWndJob <> 0
     If GetParent(hWndJob) = 0 Then
        Call GetWindowThreadProcessId(hWndJob, PID)
        If PID = ProcessID Then
           hWndShell = hWndJob
           Exit Do
        End If
     End If
     hWndJob = GetWindow(hWndJob, GW_HWNDNEXT)
  Loop
End Function
```

LISTING 2 ***Use the ProcessID to Find Shelled Window.*** *Obtaining the main window handle of a shelled app in Win32 is more involved than it ever was in Win16, but the basic idea is the same. Shell returns a Process ID, which can be compared with the return value from GetWindowThreadProcessId while looping through a list of top-level windows.*

"Fine," I hear you saying, "I'll just use the GetWindow API to find the first child window of the combo box whose class is *ComboLBox*." Think again: the drop-down list is a child of the desktop, not of the combo box. This relationship is necessary for the list to be able to extend outside the application window. Otherwise, the list box would be clipped at the border of its parent window. No sure-fire way exists to use the FindWindow or GetWindow API to locate the list box associated with a specific combo box.

YOUR APP MUST MAKE THREE API CALLS TO PREPARE THE PRINTER TO ACCEPT DIRECT OUTPUT.

But all is not lost: it turns out that when Windows sends certain messages to the combo box, it passes the handle of the list box portion as one of the parameters. So if you *subclass* the combo box and intercept those messages, you can manipulate the drop-down list as you wish. VB5's *AddressOf* function allows you to subclass without using a third-party control. For more information, see Jonathan Wood's COMponent Builder column, "Create a Subclassing Control" [*VBPJ* May 1997]. You need such a control if you're using an earlier version of VB; if you don't already have one, you can download Zane Thomas's MsgHook control from this Web site for free: http://www.teleport.com/~pweber.

In 32-bit programs, you must hook the WM_CTLCOLORLISTBOX message (download Listing 3 from DevX; see the Code Online box for details). When Windows sends that message, it passes the list box's hWnd as the last parameter (lParam). Armed with the elusive window handle, you can get the height of a single item, calculate the height necessary to display the desired number of items, and resize the drop-down list.

Use a similar, though slightly less straightforward, approach in 16-bit programs (download Listing 4 from DevX; see the Code Online box for details). Windows sends a single message (WM_CTLCOLOR) to all the constituents of a combo box, passing a special value in the high word of lParam to indicate which type of window this message is for. First you must check the upper 16 bits of lParam to make sure they contain the value CTLCOLOR_LISTBOX. If so, get the list box's hWnd from the lower 16 bits of lParam, and proceed as indicated.

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.

Spool, Shell, and Hook
Locator+ Codes

Listings for the entire issue, plus a demo app that submits files directly to the print spooler under Windows 95 or NT, and another demo app with four different shell techniques under Win32 (free Registered Level): VBPJ0298
Listings for this article only, plus the files described above (subscriber Premier Level): AP0298