Verifying Internet Access

Click & Retrieve Source CODE!

by Karl E. Peterson

Q Confirming Internet Connections

My app needs to know whether it has access to the Internet. Is there a simple way to find out? Someone suggested using the Remote Access Services (RAS) functions as one possible approach, but those look incredibly messy.

A The RAS functions are convoluted and inappropriate for making this determination. The WinInet functions provide a more reliable answer. Unfortunately, WinInet is also ill-documented, and Microsoft has not published what specific functionality ships with each version of Internet Explorer (IE). A lot of WinInet capability is packed into IE3, but to unlock it all, you must require IE4—or Win98—to be installed on your users' machines. (If it *really* were part of the operating system, this library would be available as part of a service pack, wouldn't it?)

I'm glad this question is asked occasionally, because it gives me a chance to go beyond the answer in a recent Ask the VB Pro column [*VBPJ* April 1998]. If I haven't scared you away yet, let me show you what's involved. WinInet provides the InternetGetConnectedState function, which quickly returns a value supposedly telling you whether you're connected to the Internet. This function also returns a flags parameter detailing the type of connection that's active. Simple. Until you attempt to make sense of these values, anyway.

A problem that's come to my attention since the previous mention of InternetGetConnectedState: It tells you only the state of the *default connectoid*—the dial-up networking connection. If a machine is configured with multiple ISPs, and the nondefault connectoid is in use, the flags parameter indicates that the modem is "busy" but you are connected. The confusion arises when you ask, "Connected to what?" This function, regardless of its name, does not guarantee an Internet connection, because a connectoid can connect you to virtually anything. The same cautions apply when InternetGetConnectedState tells you your connection is through either the LAN or a proxy server.

In fact, the only way to tell if you're connected to the Internet is to attempt contact with something "out there." InternetGetConnectedState is reliable when it tells you you're not connected, so this function is still useful as a first test and can save time trying to open an Internet resource when no connection is in place. My preference is to open the URL of a known reliable Web site; I do this by iterating through a collection that contains several highly reliable URLs. When InternetGetConnectedState hints that I might be connected, I confirm that by opening an Internet session using InternetOpen, then by attempting to open a known resource with InternetOpenUrl. If the latter call succeeds, I know the connection is alive and well.

I've wrapped all this logic up into a class module, CNetConnect (download the code from the free, Registered Level of The Development Exchange). I've also included a demo that illustrates just how easy it is to use. The demo is available on the free, Registered Level of DevX (see the Download Free Code box for details).

Q Copy Files to the Clipboard

I'm writing an Explorer-type file system browser, and would like to offer users the ability to "copy" files to the clipboard so they can use Paste functions in Explorer or other apps. How can I do this?

A Explorer fills the clipboard with a half-dozen data formats when you highlight a file and select Copy. Five of these are custom formats, and are difficult to use from VB. However, one of the formats, CF_HDROP, is readily usable—in fact, it's all you need to place files on the clipboard. This format, also known as a "Dropped Filelist," has a handle you can use with several drag-and-drop functions to retrieve the name of each file in the list.

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 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.

VB4/32, VB5, VB6 | Slinging Files On/Off the Clipboard

```
Option Explicit

' Required data structures
Private Type POINTAPI
    x As Long
    y As Long
End Type

' Clipboard Manager Functions
Private Declare Function EmptyClipboard Lib _
    "user32" () As Long
Private Declare Function OpenClipboard Lib _
    "user32" (ByVal hWnd As Long) As Long
Private Declare Function CloseClipboard Lib _
    "user32" () As Long
Private Declare Function SetClipboardData Lib _
    "user32" (ByVal wFormat As Long, ByVal hMem _
    As Long) As Long
Private Declare Function GetClipboardData Lib _
    "user32" (ByVal wFormat As Long) As Long
Private Declare Function IsClipboardFormatAvailable Lib _
    "user32" (ByVal wFormat As Long) As Long

' Other required Win32 APIs
Private Declare Function DragQueryFile Lib _
    "shell32.dll" Alias "DragQueryFileA" (ByVal hDrop _
    As Long, ByVal UINT As Long, ByVal lpStr As String, _
    ByVal ch As Long) As Long
Private Declare Function GlobalAlloc Lib "kernel32" _
    (ByVal wFlags As Long, ByVal dwBytes _
    As Long) As Long
Private Declare Function GlobalFree Lib "kernel32" _
    (ByVal hMem As Long) As Long
Private Declare Function GlobalLock Lib "kernel32" _
    (ByVal hMem As Long) As Long
Private Declare Function GlobalUnlock Lib "kernel32" _
    (ByVal hMem As Long) As Long
Private Declare Sub CopyMem Lib "kernel32" Alias _
    "RtlMoveMemory" (Destination As Any, Source _
    As Any, ByVal Length As Long)

' Predefined Clipboard Formats
Private Const CF_HDROP = 15

' Global Memory Flags
Private Const GMEM_MOVEABLE = &H2
Private Const GMEM_ZEROINIT = &H40
Private Const GHND = (GMEM_MOVEABLE Or GMEM_ZEROINIT)

Private Type DROPFILES
    pFiles As Long
    pt As POINTAPI
    fNC As Long
    fWide As Long
End Type

Public Function clipCopyFiles(Files() As String) _
    As Boolean
    Dim data As String
    Dim df As DROPFILES
    Dim hGlobal As Long
    Dim lpGlobal As Long
    Dim i As Long

    ' Open and clear existing crud off clipboard.
    If OpenClipboard(0&) Then
      Call EmptyClipboard

      ' Build double-null terminated list of files.
      For i = LBound(Files) To UBound(Files)
        data = data & Files(i) & vbNullChar
      Next i
      data = data & vbNullChar

      ' Allocate and get pointer to global memory,
      ' then copy file list to it.
      hGlobal = GlobalAlloc(GHND, Len(df) + Len(data))
      If hGlobal Then
        lpGlobal = GlobalLock(hGlobal)

        ' Build DROPFILES structure in global memory.
        df.pFiles = Len(df)
        Call CopyMem(ByVal lpGlobal, df, Len(df))
        Call CopyMem(ByVal (lpGlobal + Len(df)), _
          ByVal data, Len(data))
        Call GlobalUnlock(hGlobal)

        ' Copy data to clipboard, and return success.
        If SetClipboardData(CF_HDROP, hGlobal) Then
          clipCopyFiles = True
        End If
      End If

      ' Clean up
      Call CloseClipboard
    End If
End Function

Public Function clipPasteFiles(Files() As String) As _
    Long
    Dim hDrop As Long
    Dim nFiles As Long
    Dim i As Long
    Dim desc As String
    Dim filename As String
    Dim pt As POINTAPI
    Const MAX_PATH As Long = 260

    ' Ensure desired format is there, and open
    ' clipboard.
    If IsClipboardFormatAvailable(CF_HDROP) Then
      If OpenClipboard(0&) Then

        ' Get handle to Dropped Filelist data, and
        ' number of files.
        hDrop = GetClipboardData(CF_HDROP)
        nFiles = DragQueryFile(hDrop, -1&, "", 0)

        ' Allocate space for return and working
        ' variables.
        ReDim Files(0 To nFiles - 1) As String
        filename = Space(MAX_PATH)

        ' Retrieve each filename in Dropped Filelist.
        For i = 0 To nFiles - 1
          Call DragQueryFile(hDrop, i, filename, _
            Len(filename))
          Files(i) = TrimNull(filename)
        Next i

        ' Clean up
        Call CloseClipboard
      End If

      ' Assign return value equal to number of
      ' files dropped.
      clipPasteFiles = nFiles
    End If
End Function
```

Listing 1 Use this pair of functions to copy a list of files to the clipboard and paste it into another application. In reality, the clipboard contains only a simple list; it's up to the pasting application to retrieve the contents of the original files.

The clipPasteFiles routine deciphers the data, returning an array containing the file names on the clipboard.

IsClipboardFormatAvailable checks for data availability.

GetClipboardData retrieves the HDROP handle.

The DragQueryFile API accepts an HDROP, and returns either the number of files dropped or the name of one of the dropped files, depending on the value of the second parameter.

The clipCopyFiles routine accepts an array of file names, which it copies to the clipboard.

The clipCopyFiles routine accepts an array of file names, which it copies to the clipboard (see Listing 1). The first task: Open and empty the clipboard using OpenClipboard and EmptyClipboard APIs. If this succeeds—it won't if another app currently has the clipboard open—then the file list is prepared. The HDROP data format requires a double-null terminated, single-null separated list of files. A quick loop builds a string by appending each file name successively, with a Chr(0) in between and an extra Chr(0) at the end.

The CF_HDROP format can be used with several drag-and-drop functions to retrieve the names of each file in a list.

At this point, you're ready to construct a DROPFILES structure entirely in global memory. You can't use a standard VB structure, because the clipboard retains ownership of the data after you hand it over. Use GlobalAlloc to set aside enough space for a DROPFILES structure *plus* enough to tack the constructed file-list string onto the end. Call GlobalLock to obtain a pointer to this memory. Assign the structure length to the first element of a local DROPFILES structure—the other elements aren't needed in this case—to indicate the file list will immediately follow the structure in memory. Then copy the structure to the address returned by GlobalAlloc and the file list to the offset indicated in the last step. Finally, use GlobalUnlock to release the locked memory, send it to the clipboard with ClipboardSetData, and call CloseClipboard to finish up.

You'll see that the Paste option on Explorer's Edit menu is enabled. In addition, Explorer and other apps have access to the file list you placed on the clipboard. The clipPasteFiles routine illustrates deciphering this data by returning an array containing the file names on the clipboard (see Listing 1). IsClipboardFormatAvailable checks for data availability, OpenClipboard is called, and GetClipboardData retrieves the HDROP handle. The DragQueryFile API accepts an HDROP, and returns either the number of files dropped or the name of one of the dropped files, depending on the value of the second parameter. 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. Based in Vancouver, Wash., he's also an independent programming consultant who specializes in ActiveX controls and contributes to various journals. Karl coauthored **Visual Basic 4 How-To**, from Waite Group Press. Online, he's a Microsoft MVP, and a section leader several **VBPJ** online forums. Find more of Karl's VB samples at http://www.mvps.org/vb.

DOWNLOAD FREE CODE

Download the code for this issue of **VBPJ free** from http://www.vbpj.com.

To get the free code for this entire issue, type **VBPJ1298** into the Locator+ field at the top right of the VBPJ home page. (You first need to register, for free, on DevX.) The free code for this article includes all code listings, plus the CNetConnect class module and Internet connection demo.

✪ To get the bonus code for this article, available to DevX Premier Club members, type **VBPJ1298AP** into the Locator+ field. The bonus code includes all the free code described above, plus a demo showing how to copy and paste files to/from the clipboard, and a VB-based clipboard viewer application.