Let Users Automate File Pasting

by Karl E. Peterson

Click & Retrieve Source CODE!

Q PUT THE FILE WHERE YOU WANT

I wrote a program that loads files, one at a time, into the Windows clipboard. The program loads the first file, and when the user presses the Load Next button, the next file in the series is loaded onto the clipboard. It's up to the user to paste the contents of the clipboard into the appropriate place in another application.

I'd like to add an "Automate" option to let the user configure the program and press the Automate button. The program would paste the first file into the other application, load the next file, wait a predetermined period of time, and then paste the next file, repeating the process until all the files are transferred.

How do I tell my program which application it should paste into which window? When switching focus to the other application, would the cursor stay in the appropriate place so the data is sent to the right place? How do I force the paste? If my program is running the automation, would it know when to load the next file?

—Richard Hendricks, Tewksbury, Massachusetts

A

So many questions! Let's look at the first one and see if the answer resolves the rest. I would turn it around and ask, "How can my user tell my program where to paste the text?" You allow the user to point at the destination window. There really is no other way short of telepathy, is there? This solution is the simplest method, but as easy as it sounds, it raises some tricky issues.

The SetCapture API instructs Windows to direct all mouse input to the designated window. There's just one catch: under Win32, if the user moves the mouse cursor over a window created in another process, mouse input only continues as long as the user holds down the mouse button. To make this workable, you can have your user drag a drop-target to the desired window.

I've implemented a little demo of this method that sets the MousePointer to a custom cursor in a picture box's MouseDown event. To re-create it, start a new project and put a small, icon-sized picture box on Form1. Assign a selection-type icon to the DragIcon property of this picture box (see Figure 1), and start adding code to the form (see Listing 1). A form-level Boolean variable is toggled to indicate that the user is currently selecting a window. Pass the picture box's hWnd property to SetCapture to redirect all mouse movements to the picture box's MouseMove event.

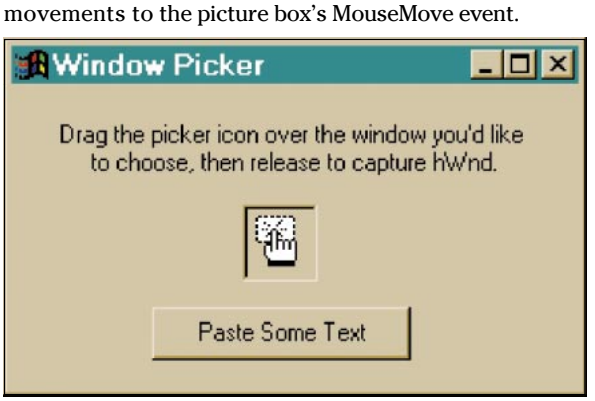

FIGURE 1 ***Drag an Icon to Select Window.*** *This sample applet lets a user drag the icon in the picture box around the screen, capturing the window handle as each window passes under the icon.*

In the MouseMove event, use another form-level variable to track which window the cursor is currently over whenever the user moves the mouse. You can identify the window fairly easily using two API calls: GetCursorPos returns the cursor location in screen coordinates, and WindowFromPoint returns the hWnd of the underlying window. Finally, use the MouseUp event to clean up and signal that you're ready to move on with the task at hand. All that's required is releasing

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, Washington, he's also an independent programming consultant specializing in ActiveX controls. Karl coauthored* Visual Basic 4 How-To *from Waite Group Press and contributes to various journals. Online, he's a Microsoft Developer MVP and a section leader for both* VBPJ *online forums. Contact Karl at karl@rtc.wa.gov.*

This is your forum for addressing the intricacies of Visual Basic. Send your questions, clever tips, and techniques. Visual Basic Programmer's Journal *will pay $25 for any submission, tip, or question we print. If your submission includes code, please send it electronically. Please include your mailing address with your submission. Mail submissions to Q&A Columnists, c/o Fawcette Technical Publications, 209 Hamilton Avenue, Palo Alto, CA, USA, 94301-2500. CompuServe: 74774,305. Internet: vbpjedit@fawcette.com.*

your hold on the mouse with a call to ReleaseCapture, and restoring the MousePointer to its normal state.

Now that you have the handle of the window your user wants to paste the text into, the other details start to fall into place. Passing the stored window handle to SetForegroundWindow transfers focus to the selected window. Now use SendKeys to paste the contents of the clipboard into this window:

```
Private Sub Command1_Click()
   '
   ' Attempt to paste something into
   ' selected window.
   '
   If m_hWnd Then
      Clipboard.Clear
      Clipboard.SetText _
         "Hello from VB5!", vbCFText
      Call SetForegroundWindow(m_hWnd)
      SendKeys "^v", True
   End If
End Sub
```

Once this task is automated, you can loop through all the file pastes required as soon as the user indicates where to paste the text.

Q DETECT SCROLLBAR USE

Using VB3, I need to recognize when the scrollbar of a list box has been used—that is, when the TopIndex value changes. However, there are no events for this user action. Using the scrollbar doesn't generate a MouseDown or any other event. How can I detect when the scrollbar has been used and take action when the TopIndex value changes?
Keith Campbell, received by e-mail

A Yet another example of why subclassing controls have always been popular with VB programmers! You can use two methods to arrive at a solution to this problem. The first is "pure VB," and I think you'll see why I like the second as soon as you hear the first.

One approach is to set a short-duration Timer control, and continuously poll the

VB4 32-bit VB5

```
Option Explicit
'
' Win32 API Declarations
'
Private Declare Function SetCapture Lib "user32" _
  (ByVal hwnd As Long) As Long
Private Declare Function ReleaseCapture Lib "user32" _
  () As Long
Private Declare Function GetCursorPos Lib "user32" _
  (lpPoint As POINTAPI) As Long
Private Declare Function WindowFromPointXY Lib _
  "user32" Alias "WindowFromPoint" (ByVal xPoint As _
  Long, ByVal yPoint As Long) As Long
Private Declare Function SetForegroundWindow Lib _
  "user32" (ByVal hwnd As Long) As Long
'
' Win32 API Structures
'
Private Type POINTAPI
  x As Long
  y As Long
End Type
'
' Form-level member variables
'
Private m_hWnd As Long
Private m_Picking As Boolean

Private Sub Form_Load()
  '
  ' Assign dragging pointer
  '
  Picture1.Picture = Picture1.DragIcon
  Me.MouseIcon = Picture1.DragIcon
End Sub

Private Sub Picture1_MouseDown(Button As Integer, _
  Shift As Integer, x As Single, y As Single)
  '
  ' Clear picture and turn on dragging mousepointer.
  '
  Me.MousePointer = vbCustom
  Set Picture1.Picture = Nothing
  '
  ' Remember that we're currently picking a window.
  '
  m_Picking = True
  '
  ' Capture all mousemovements from this point until
  ' the user releases the mouse button.
  '
  Call SetCapture(Picture1.hwnd)
End Sub

Private Sub Picture1_MouseMove(Button As Integer, _
  Shift As Integer, x As Single, y As Single)
  Static pt As POINTAPI
  Static hwnd As Long
  '
  ' If user is picking a window, check window is under
  ' the cursor whenever it moves. If it's a different
  ' window than previously, update the display to that
  ' effect.
  '
  If m_Picking Then
     Call GetCursorPos(pt)
     hwnd = WindowFromPointXY(pt.x, pt.y)
     If hwnd <> m_hWnd Then
        m_hWnd = hwnd
        Me.Caption = Hex(m_hWnd)
     End If
  End If
End Sub

Private Sub Picture1_MouseUp(Button As Integer, _
  Shift As Integer, x As Single, y As Single)
  '
  ' We're done picking now
  '
  m_Picking = False
  '
  ' Restore dragging icon to picture box,
  ' and return mousepointer to normal.
  '
  Picture1.Picture = Picture1.DragIcon
  Me.MousePointer = vbDefault
  '
  ' Don't need to be notified anymore.
  '
  Call ReleaseCapture
  '
  ' The chosen window is already stored in m_hWnd!
  '
  MsgBox "You picked hWnd: " & Hex(m_hWnd)
End Sub
```

LISTING 1 ***Allow Users to Locate Target Window.*** *This code enables users to point to a destination window for operations, such as pasting information from the clipboard. Apply the same technique in 16-bit VB apps by modifying the API declarations for that platform.*

TopIndex property of your list box. Pretty ugly, huh? The other approach is to hook the WM_VSCROLL message using a subclassing control. All such controls work, but I have historically recommended using Mabry's MsgHook, because it is freeware, works well, and offers a few unique capabilities. A net.search should turn up MsgHook.vbx in quite a few locations. Subscribers to the Premier Level of The Development Exchange can download a combination package that includes a 16-bit VBX and both 16- and 32-bit OCXs (see the Code Online box at the end of this column for details).

Although all subclassing controls use slightly different syntax, they tend to follow the same patterns. You first need to tell the control which window it's subclassing, and which messages you're interested in. In my sample program, I used this code in the Form_Load event:

```
MsgHook1.HwndHook = List1.hWnd
MsgHook1.Message(WM_VSCROLL) = True
```

Now just sit back and wait for the subclassing control to notify you whenever it receives messages you want. MsgHook fires a Message event whenever this happens:

```
Sub MsgHook1_Message (msg As Integer, _
   wparam As Integer, lparam As Long, _
   result As Long)
   '
   ' We're only hooking one message
   ' this time, but others could be
   ' handled here as well.
   '
   Select Case msg
      Case WM_VSCROLL
         '
         ' Pass message along to VB for
         ' default processing, then
         ' update display to show that
         ' scrolling was detected.
         '
         result = InvokeWindowProc_
            (MsgHook1.HwndHook, msg, wparam, lparam)
         Me.Caption = "TopIndex: " & List1.TopIndex
   End Select
End Sub
```

Windows sends a message to the parent control, in this case the list box, whenever an attached scrollbar is used. With these few lines of code, you've intercepted this message. In fact, the scrolling hasn't even occurred yet. Because of this, before reacting to the message, you need to let the default processing occur. MsgHook accomplishes this by calling the InvokeWindowProc function (implemented as a method in the OCX versions). In the Declarations section of your form, declare the InvokeWindowProc function and WM_VSCROLL constants like this:

```
'
' Routine within MsgHook that calls
' default window procedure
'
Declare Function InvokeWindowProc Lib _
   "MsgHook.vbx" (ByVal hWnd As _
   Integer, ByVal msg As Integer, _
   ByVal wparam As Integer, ByVal _
   lparam As Long) As Long
'
' Message hooked to detect scrolling
'
Const WM_VSCROLL = &H115
```

Subclassing is a technique offering capabilities that probably won't rate high enough with developers to be incorporated as VB events. However, when you need them, you *really* need them.

Code Online

You can find all the code published in this issue of* VBPJ *on The Development Exchange (DevX) at http://www.windx.com. All the listings and associated files essential to the articles are available for free to Registered members of DevX, in one ZIP file. This ZIP file is also posted in the Magazine Library of the* VBPJ *Forum on CompuServe. DevX Premier Club members ($20 for six months) can get each article's listings in a separate file, as well as additional code and utilities for selected articles, plus archives of all code ever published in* VBPJ *and* Microsoft Interactive Developer *magazines.

Let Users Automate File Pasting

Locator+ Codes

Listings ZIP file plus sample apps demonstrating the Window-picker and catching WM_VSCROLL messages (free Registered Level): VBPJ0997

Listings ZIP file plus sample apps demonstrating the Window-picker and catching WM_VSCROLL messages, as well as freeware 16-bit VBX and 16- and 32-bit OCX MsgHook controls from Mabry Software (subscriber Premier Level): QA0997P