Visual Studio Magazine Online
Classic VB Corner

Hooking the Mouse

ONLINE ONLY

Subclass multiple controls with a single class module to add the missing MouseEnter and MouseLeave events.

July 28, 2009 · by Karl E. Peterson

Subclassing in Classic VB has always been a bit of a pain. In fact, it wasn't even possible until VB5 offered the AddressOf operator to facilitate system callbacks. Recently, I wrote about the SetWindowSubclass API which offers an incredibly slick way to hook into any in-process window's message stream. In this article, I'll show you how to build a class that can provide message processing for multiple controls no matter their type. If you haven't read the previous article, please do before continuing here as this article will assume that as pretext.

To illustrate the utility of this method, I chose to fill in a widely recognized gap in functionality: the lack of MouseEnter and MouseLeave events. Oddly enough, Windows offers both WM_MOUSELEAVE and WM_MOUSEHOVER notifications, but doesn't offer a WM_MOUSEENTER message. So it's up to us to recognize this event ourselves, which can be done by monitoring WM_MOUSEMOVE. When that message arrives for the first time, we know that the mouse has just entered the hWnd of interest. A static variable remembers this handle, and is reset when the mouse later leaves the window.

The only remaining trick is enabling the leave and hover notifications. For this, we call the TrackMouseEvent API when we first detect the mouse entering a watched window. We can use this call to tell Windows we want to be notified when the mouse leaves the window, as well as when it first comes to rest for a given "hover" interval. Raising a MouseHover event in the latter case allows such things as providing informational materials, but only after the user brings the mouse to a rest. Here's the whole message processing scheme:

```
Private Function IHookXP_Message(ByVal hWnd As Long, _
   ByVal uiMsg As Long, ByVal wParam As Long, _
   ByVal lParam As Long, ByVal dwRefData As Long) As Long

   Static hWndLast As Long

   ' Special processing for messages we care about.
   Select Case uiMsg
      Case WM_MOUSEMOVE
         If hWnd <> hWndLast Then
            hWndLast = hWnd  'Set flag
            RaiseEvent MouseEnter(hWnd)
            ' Start tracking for Leave event.
            StartTrackingMouse hWnd, TME_LEAVE Or TME_HOVER
         End If

      Case WM_MOUSEHOVER
```

```
         RaiseEvent MouseHover(hWnd)

      Case WM_MOUSELEAVE
         hWndLast = 0  'Clear flag
         RaiseEvent MouseLeave(hWnd)
   End Select

   ' Pass back to default message handler.
   IHookXP_Message = HookDefault(hWnd, uiMsg, wParam, lParam)
End Function

Private Function StartTrackingMouse _
   (ByVal hWnd As Long, ByVal Flags As Long) As Long
   Dim tme As TRACKMOUSEEVENT_STRUCT
   ' Wrap all the mess into a tidy little procedure.
   With tme
      .cbSize = Len(tme)
      .dwFlags = Flags
      If .dwFlags And TME_HOVER Then
         .dwHoverTime = m_HoverTime
      End If
      .hWndTrack = hWnd
   End With
   ' Return results.
   StartTrackingMouse = TrackMouseEvent(tme)
End Function
```

Notice that the raised events are passing an hWnd parameter back to the client. As I mentioned earlier, this class is designed to hook messages for any number of controls. In fact, here's the code I use to set up the class in Form_Load event of the demo on my site:

```
Private WithEvents m_ME As CHookMouseEvents

Private Sub Form_Load()
   ' Start watching for mouse events.
   Set m_ME = New CHookMouseEvents
   m_ME.HoverTime = 1000 'milliseconds
   m_ME.Add Me
   m_ME.Add Text1
   m_ME.Add HScroll1
   m_ME.Add VScroll1
   m_ME.Add Combo1
   m_ME.Add Option1
   m_ME.Add Check1
   m_ME.Add List1
   m_ME.Add Drive1
   m_ME.Add Dir1
   m_ME.Add File1
End Sub
```

Yeah, it's a form with every one of the visible intrinsic controls included. I've built the CHookMouseEvents class to function somewhat similarly to a Collection on steroids. In fact, it's using a native Collection object to store references to all the hooked windows. The code is simple, and allows great ease of setup:

```
Private Sub Class_Initialize()
   ' Set defaults
   m_Enabled = defEnabled
   m_HoverTime = defHoverTime
   Set m_Objects = New Collection
End Sub

Public Function Add(obj As Object) As Boolean
   On Error Resume Next
   m_Objects.Add obj, FmtHex(obj.hWnd, 8)
   If Err.Number = 0 Then 'success
      If HookSet(obj.hWnd, Me) Then
         Add = True
      End If
   Else
      Debug.Print Err.Number, Err.Description
   End If
End Function

Public Function Count() As Long
   Count = m_Objects.Count
End Function

Public Function Item(ByVal hWnd As Long) As Object
   On Error Resume Next
   Set Item = m_Objects.Item(FmtHex(hWnd, 8))
End Function

Public Function Remove(obj As Object) As Boolean
   On Error Resume Next
   Remove = UnhookOne(obj.hWnd)
End Function
```

Obviously, this will only work for objects that expose an hWnd property. The HookSet procedure called in the Add method was discussed in detail in the previous article. It's simply a call to SetWindowSubclass that uses an ObjPtr to the IHookXP interface passed to it as the ultimate destination for the hooked messages. To remove an object from message processing, we just remove it from the collection and call HookClear (see previous article) which is a shortcut to RemoveWindowSubclass.

No clean-up is necessary in the form that's hosting all the controls and sinking the messages raised by CHookMouseEvents. The class handles all the dirty work by watching for WM_NCDESTROY messages in the IHookXP_Message callback method, and calling UnhookOne just as it would in the Remove method. The class also takes care to call UnhookAll in its own Terminate event:

```
Private Sub Class_Terminate()
   ' Tear down
   Call UnhookAll
   Set m_Objects = Nothing
End Sub

Private Sub UnhookAll()
   Dim obj As Object
   ' Clear existing hook.
   For Each obj In m_Objects
      Call HookClear(obj.hWnd, Me)
```

```
   Next obj
End Sub
```

And that's really all there is to it. So, to add this processing to your project, you simply drop in the three modules I supply (CHookMouseEvents.cls, MHookXP.bas, IHookXP.cls), declare an instance of CHookMouseEvents using WithEvents, and hand it the objects you'd like to be notified of mouse activity for.

Adding support for the extra two buttons on five-button mice is also incredibly simple. Just plug in handlers for three more notifications in your message processor:

```
Private Function IHookXP_Message(ByVal hWnd As Long, _
   ByVal uiMsg As Long, ByVal wParam As Long, _
   ByVal lParam As Long, ByVal dwRefData As Long) As Long

   Static hWndLast As Long

   ' Special processing for messages we care about.
   Select Case uiMsg
      Case WM_XBUTTONDOWN
         If m_Enabled Then
            RaiseEvent XButtonDown(hWnd, _
               WordHi(wParam), WordLo(lParam), WordHi(lParam))
         End If

      Case WM_XBUTTONUP
         If m_Enabled Then
            RaiseEvent XButtonUp(hWnd, _
               WordHi(wParam), WordLo(lParam), WordHi(lParam))
         End If

      Case WM_XBUTTONDBLCLK
         If m_Enabled Then
            RaiseEvent XButtonDblClick(hWnd, _
               WordHi(wParam), WordLo(lParam), WordHi(lParam))
         End If
   End Select

   ' Pass back to default message handler.
   IHookXP_Message = HookDefault(hWnd, uiMsg, wParam, lParam)
End Function
```

For each of these notifications, the Button value is stored in the high word of wParam, and the X/Y mouse coordinates are stored in the low and high words (respectively) of lParam. I've expanded the HookXP sample on my site to include this new demo of mouse event processing. To me, the fascinating aspect of this class is how it handles hooking multiple windows, while taking care of all the "dirty housekeeping" with what's actually very minimal code.

As always when subclassing with native code, be safe. Unhandled errors can be deadly. Save before running.

About the Author

Karl E. Peterson wrote Q&A, Programming Techniques, and various other columns for VBPJ and VSM from 1995 onward, until Classic VB columns were dropped entirely in favor of other languages. Similarly, Karl was a Microsoft BASIC MVP from 1994 through 2005, until such community contributions were no longer deemed valuable. He is the author of VisualStudioMagazine.com's new Classic VB Corner column. You can contact him through his Web site if you'd like to suggest future topics for this column.

1105 Redmond Media Group
Copyright 1996-2009 1105 Media, Inc. View our Privacy Policy.