Visual Studio Magazine Online
Classic VB Corner

Take Control of Window Movements

ONLINE ONLY

Hook the window message stream to snap your main window to the screen edges as the user moves it about.

April 14, 2008 · by Karl E. Peterson

I was writing a simple little utility last month that was meant to merely sit onscreen and convey some information that varied as time went on. The idea was for it to be as obscure as possible, yet visible -- sort of like Google Desktop's sidebar.

I'd seen a feature in Winamp that I wanted to emulate: the ability to "snap" to the screen edge as the user dragged it close. This would allow the user to easily position my little dialog as close to the edge or into the corner as possible without going over the edge. It's a subtle but (at least personally) highly appreciated nod to the user that can add a professional touch to many apps.

First thing I do is scan the Internet to see what sorts of examples may be out there. As common as this is, surely someone must've covered it already, right? Well, the best I could find was at the CodeProject Web site. The basic ideas were there, but with lots of omissions. The CodeProject sample worked a lot harder than it should have to detect where the taskbar was, assumed the taskbar was visible, didn't even consider the case of multiple monitors and was just generally quirky. Clearly, a better example was called for. (And what's better than ClassicVB!?)

The general idea for implementing this technique is to hook into (subclass) your window's message stream and watch for the WM_WINDOWPOSCHANGING notification. When the user drags your window around the screen, Windows notifies your app by sending a series of these notifications, followed by a WM_WINDOWPOSCHANGED notification when the user releases the mouse button. Both notifications include a pointer to a WINDOWPOS structure in lParam. If you hook these messages, and don't pass them along to the default window procedure, you can control how your window responds to the user. The basic replacement hook procedure looks like this:

```
Private Function IHookSink_WindowProc( _
   hWnd As Long, msg As Long, wp As Long, lp As Long) As Long
   Dim pos As WINDOWPOS
   ' Handle each message appropriately.
   Select Case msg
      Case WM_WINDOWPOSCHANGING
         ' Snag copy of position structure, process it,
         ' and pass back to Windows any changes.
         Call CopyMemory(pos, ByVal lp, Len(pos))
         Call SnapToDesktopEdge(pos, hWnd)
         Call CopyMemory(ByVal lp, pos, Len(pos))

      Case Else
```

```
         ' Just allow default processing for everything else.
         IHookSink_WindowProc = _
            InvokeWindowProc(hWnd, msg, wp, lp)
   End Select
End Function
```

IHookSink is an interface I provide with the HookMe sample on my Web site. It provides the signature for this callback, which allows you to sink messages for a window in any object that can implement the interface. Upon receiving WM_WINDOWPOSCHANGING, you can retrieve the current window coordinates by dereferencing the lParam pointer with a CopyMemory (RtlMoveMemory) call. You are now free to modify these coordinates however you'd like, then copy them back to the address Windows expects (lParam). That's all there is to it! Since the message hook isn't restricted to just this one message, it's your obligation to ensure that every other message gets routed through the default window procedure. This is easily accomplished using my HookMe module.

Snapping to the edge of the current monitor involves just a few simple tasks. First, you need to determine the work area (absolute coordinates) of the monitor on which the bulk of your window is currently displayed. Traditionally, this meant a call to SystemParametersInfo, asking for SPI_GETWORKAREA. But the increasing prevalence of multi-monitor setups means you should also check to see if there are more than one on the current machine, and adjust appropriately. GetSystemMetrics will tell us how many monitors are active. In cases of two or more, MonitorFromWindow retrieves a handle to the monitor on which a given window (mostly) exists, and GetMonitorInfo supplies information about that monitor including its work area.

```
Private Function GetWorkArea(ByVal hWnd As Long) As RECT
   Dim hMonitor As Long
   Dim mi As MonitorInfo

   ' Default to using traditional method, as fallback for
   ' cases where only one monitor is being used or new
   ' multimonitor method fails.
   Call SystemParametersInfo(SPI_GETWORKAREA, _
                             0&, GetWorkArea, 0&)

   ' Use newer multimonitor method when needed.
   If GetSystemMetrics(SM_CMONITORS) > 1 Then
      ' Get handle to monitor that has bulk of window within it.
      hMonitor = MonitorFromWindow(hWnd, _
                                   MONITOR_DEFAULTTONEAREST)
      If hMonitor Then
         mi.cbSize = Len(mi)
         Call GetMonitorInfo(hMonitor, mi)
         GetWorkArea = mi.rcWork
      End If
   End If
End Function
```

At this point, completing the "snap" is just a matter of math. Compare each edge of your window (using the WINDOWPOS structure coordinates) with the work area rectangle retrieved from the routine above. If you find that an edge is less than some predetermined snap tolerance, adjust as desired. The WINDOWPOS structure makes

this incredibly easy, because it stores the width and height rather than the right and bottom coordinates, so you only have to adjust the left and/or top elements:

```
Private Sub SnapToDesktopEdge(pos As WINDOWPOS, _
                              ByVal hWnd As Long)
   Dim mon As RECT            ' monitor work area coords
   Const SnapGap As Long = 15 ' snap tolerance, in pixels

   ' Get coordinates for main work area.
   mon = GetWorkArea(hWnd)

   ' Snap X axis
   If Abs(pos.x - mon.Left) <= SnapGap Then
      pos.x = mon.Left
   ElseIf Abs(pos.x + pos.cx - mon.Right) <= SnapGap Then
      pos.x = mon.Right - pos.cx
   End If

   ' Snap Y axis
   If Abs(pos.y - mon.Top) <= m_SnapGap Then
      pos.y = mon.Top
   ElseIf Abs(pos.y + pos.cy - mon.Bottom) <= SnapGap Then
      pos.y = mon.Bottom - pos.cy
   End If
End Sub
```

I've bundled up the SnapDialog sample, complete with API declarations, and made it freely available on my Web site as a drop-in ready class module. It should be easily modifiable for any subclassing scheme you already have in place, or you can use the native one included in the sample which I've found useful in probably every application I've written over the last decade. Enjoy!

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.