Visual Studio Magazine Online
Classic VB Corner

No Such Thing as a Windowless VB App

All Classic VB apps have at least one top-level window. Normally out of reach, you can put them to good use if you know how to get to them.

November 16, 2009 · by Karl E. Peterson

Every Classic VB app has a hidden, top-level window. I suppose after nearly 20 years, stating that ranks right up there with "the sun rises in the east." It's a fact that nearly every VB programmer must be aware of, if only in passing. VB uses this hidden window to receive notifications and events from the system, and generally play traffic cop with all the other windows in your application.

Why should you care? Because hooking into those system message streams can provide information that simply isn't available, or at least isn't available simply, using other methods.

For example, consider a long-running console application that needs to be aware when Windows is trying to shutdown, so that it can avoid potentially corrupting the data it's working with. The classic answer would be to monitor for WM_QUERYENDSESSION and WM_ENDSESSION messages, which are sent to every top-level window in the system, and react appropriately.

If a developer didn't know about the window already attached to the process, the temptation to create a window (or add a form) just for this purpose would be the natural reaction. But simply enumerating all the windows in the current process, looking for the magic classname, will provide a ready handle to cast your hook upon.

The introduction of AddressOf in VB5 opened up a wealth of new opportunities, among them the ability to call the Enum* API functions. To initiate the sequence of callbacks from Windows, once for each window in our thread, we'll use the EnumThreadWindows API. This call accepts three parameters: the ThreadID offered by VB's App object, the address of the callback routine, and another Long value that can be useful in the callback routine itself.

You can use that last parameter to perform a neat little trick. Pass the pointer to the calling function's return value. This allows you to set the ultimate return value directly from the callback routine. In a BAS module, enter the following code:

```
Private Declare Function EnumThreadWindows Lib "user32" _
   (ByVal dwThreadId As Long, ByVal lpfn As Long, _
   ByVal lParam As Long) As Long

Public Function FindHiddenTopWindow() As Long
   ' This function returns the hidden toplevel window
   ' associated with the current thread of execution.
   Call EnumThreadWindows(App.ThreadID, _
      AddressOf EnumThreadWndProc, VarPtr(FindHiddenTopWindow))
```

```
End Function
```

In this case, EnumThreadWndProc is the routine in the same module that Windows will call once for each window on the thread, or until we tell it to stop. EnumThreadWndProc is passed two parameters -- the first a handle to the window being enumerated, and the second our lucky Long value passed to the EnumThreadWindows API.

```
Private Declare Function GetWindowLongA Lib "user32" _
   (ByVal hWnd As Long, ByVal nIndex As Long) As Long

Private Declare Sub CopyMemory Lib "kernel32" _
   Alias "RtlMoveMemory" _
   (Destination As Any, Source As Any, ByVal Length As Long)

Private Const GWL_HWNDPARENT As Long = -8&

Private Function EnumThreadWndProc(ByVal hWnd As Long, _
   ByVal lpResult As Long) As Long
   Dim nStyle As Long
   Dim Class As String

   ' Assume we will continue enumeration.
   EnumThreadWndProc = True

   ' Test to see if this window is parented.
   ' If not, it may be what we're looking for!
   If GetWindowLongA(hWnd, GWL_HWNDPARENT) = 0 Then
      ' This rules out IDE windows when not compiled.
      Class = Classname(hWnd)
      ' Version agnostic test.
      If InStr(Class, "Thunder") = 1 Then
         If InStr(Class, "Main") = (Len(Class) - 3) Then
            ' Copy hWnd to result variable pointer,
            Call CopyMemory(ByVal lpResult, hWnd, 4&)
            ' and stop enumeration.
            EnumThreadWndProc = False
         End If
      End If
   End If
End Function
```

Inside the callback, we can perform any tests we want to, using the passed window handle. This is where you need to know a thing or three about the window you're looking for. We're seeking one that has no parent/owner window, and has a classname that starts with "Thunder" and ends with "Main".

The latter test is a bit complicated if you want portable code. Running under the IDE, the hidden window will have a classname of "ThunderMain", but running as an EXE it will be either "ThunderRT5Main" or "ThunderRT6Main", depending on which version of VB it was built under.

As with many of the enumeration callbacks Windows offers, this particular enumeration will continue as long as the called procedure returns TRUE. If we find the window we're looking for, we can abort the enumeration by returning FALSE.

The really cool trick here is how we tell the procedure that initiated the callback which window we've settled on based on our tests. To do that, I use RtlMoveMemory to copy the window handle value directly into the memory address used for the original procedure's return value.

To make this a standalone column, no download is necessary -- the only piece missing is the routine that determines the classname for any given window handle. So, here's one you can drop in anywhere:

```
Private Declare Function GetClassname Lib "user32" _
   Alias "GetClassNameA" _
   (ByVal hWnd As Long, ByVal lpClassName As String, _
   ByVal nMaxCount As Long) As Long

Public Function Classname(ByVal hWnd As Long) As String
   Dim nRet As Long
   Dim Class As String
   Const MaxLen As Long = 256

   ' Retrieve classname of passed window.
   Class = String$(MaxLen, 0)
   nRet = GetClassname(hWnd, Class, MaxLen)
   If nRet Then Classname = Left$(Class, nRet)
End Function
```

So, if you're ready now to hook those WM_ENDSESSION messages, all you need to do is use whatever your favorite subclassing technique may be, along with the result returned from the FindHiddenTopWindow routine above. If you don't have a favorite subclassing method, I'd highly recommend you grab the SysInfo sample from my Web site, for examples of this and much more.

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.