Visual Studio Magazine Online
Classic VB Corner

# Listening to ThunderMain

*Windows provides an ongoing stream of general system notifications that you can fairly easily hook into.*

**December 17, 2009 · by Karl E. Peterson**

Last time around, I showed you how to find the hidden top-level window that's present in all Classic VB applications. The name of this window is a bit different depending on what version of VB you're using and whether the application is compiled or running within the IDE. For simplicity, I'll just refer to it as "ThunderMain" as it's called in uncompiled applications.

Windows sends all sorts of interesting notification messages to all top-level windows, and it's a mystery why Microsoft chose not to create general purpose events out of some of them. I've already mentioned how WM_QUERYENDSESSION and WM_ENDSESSION are two of these useful notifications. As it happens, those two messages are reflected in every form's QueryUnload event. You can monitor the UnloadMode parameter of that event, watching for vbAppWindows which indicates that the reason the event being triggered is that Windows is shutting down. This is the equivalent of WM_QUERYENDSESSION, and you have the opportunity to cancel the shutdown at that point.

But what if you don't have any forms in your application? Would you want to add one just to watch for this possibility? Probably not. If you hook ThunderMain using my subclassing technique , you can easily create an application global class capable of firing these events to whatever component is interested in listening.

```
Public Event EndSession(ByVal EndingInitiated As Boolean, _
   ByVal Flag As Long)
Public Event QueryEndSession(ByVal Flag As EndSessionFlags, _
   Cancel As Boolean)

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

   Dim Cancel As Boolean
   Dim EatIt As Boolean

   ' Special processing for messages we care about.
   Select Case uiMsg
      Case WM_ENDSESSION
         ' wParam confirms end, lParam provides reason flag.
         ' If the session is being ended, wParam is TRUE; the
         ' session can end any time after all applications have
         ' returned from processing this message. Otherwise,
         ' it is FALSE.
         RaiseEvent EndSession(CBool(wParam), lParam)
```

```
      Case WM_QUERYENDSESSION
         ' lParam provides flag that indicates reason for ending.
         RaiseEvent QueryEndSession(lParam, Cancel)
         ' Applications should respect the user's intentions and
         ' return TRUE. By default, the DefWindowProc function
         ' returns TRUE for this message.
         ' If shutting down would corrupt the system or media that
         ' is being burned, the application can return FALSE.
         IHookXP_Message = Abs(Not Cancel)
         EatIt = True
   End Select

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

What other sorts of cool stuff does VB know about, but chooses not to tell you? Lots! How often might it be useful to know that the user had shifted focus between your application and another? You can watch for WM_ACTIVATEAPP and key off wParam to tell whether your application is gaining or losing activation.

Do you want to perform some action every so often? If the period is longer than a minute, the most common method is to fire a periodic timer event, and compare the current time with the time you last performed that action. It would be useful to know if the system clock had been modified, wouldn't it? WM_TIMECHANGE lets you know just that.

Are you perhaps writing an application that needs to interact with the display as a whole? Maybe it's capturing the entire screen, drawing directly on it, positioning items in specific places, emulating non-client window features, or any number of other such actions. If so, you can use WM_DISPLAYCHANGE to learn when the screen resolution or image depth of the screen changes. A pair of messages, WM_SYSCOLORCHANGE and WM_THEMECHANGED , will let you know when the system palette or theme changes, so you can react appropriately.

Then there's the kitchen sink message, WM_SETTINGCHANGE, which tells you that "something else" has changed. It could be any number of things, and takes a little investigation to understand. Most often, this message fires in reaction to some application calling SystemParametersInfo to change some system setting, but can also fire following a policy modification.

Do you offer your users a list of fonts to work with in your application? You can key off the WM_FONTCHANGE to update your list. These are just a sampling of the sorts of useful information available to all top-level windows. You can extend the Select case shown above to account for these other messages, like this:

```
   ' Special processing for messages we care about.
   Select Case uiMsg
      Case WM_ACTIVATEAPP
         ' wParam indicates active status.
         RaiseEvent ActivateApp(CBool(wParam))

      Case WM_DISPLAYCHANGE
         ' wParam is BPP, lParam is X/Y.
         RaiseEvent DisplayChange(wParam, _
            LoWord(lParam), HiWord(lParam))

      Case WM_ENDSESSION
         ' (see above)
      Case WM_FONTCHANGE
         ' wParam and lParam are both reserved, unused.
         RaiseEvent FontChange

      Case WM_QUERYENDSESSION
         ' (see above)

      Case WM_SYSCOLORCHANGE
         ' wParam and lParam are both reserved, unused.
         RaiseEvent SysColorChange

      Case WM_SETTINGCHANGE
         ' wParam is source Flag, lParam is source Name.
         ' Uses MMsgLookup.bas module for demo purposes!
         RaiseEvent SettingChange(SettingName(wParam, lParam), _
            wParam)

      Case WM_THEMECHANGED
         ' wParam and lParam are both reserved, unused.
         RaiseEvent ThemeChanged

      Case WM_TIMECHANGE
         ' No parameters.
         RaiseEvent TimeChanged
   End Select
```

The corresponding event declarations could look something like this:

```
Public Event ActivateApp(ByVal Active As Boolean)
Public Event DisplayChange(ByVal BitsPerPixel As Long, _
   ByVal PixelsX As Long, ByVal PixelsY As Long)
Public Event FontChange()
Public Event SettingChange(ByVal Setting As String, _
   ByVal Flag As Long)
Public Event SysColorChange()
Public Event ThemeChanged()
Public Event TimeChanged()
```

In my next column, we'll take a look at monitoring the power status of the device your application is running on. In the meantime, if you'd like to snag a functional example of ThunderMain message hooking, take a look at the SysInfo sample on my Web site . I wonder if anyone reading this recognizes the namesake for this sample? Let me know, below, if you do.

## 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.*