Visual Studio Magazine Online
Classic VB Corner

Honoring Startup Requests

Windows provides numerous ways to tell an application how to size and position itself on startup. Here's how you can honor those requests.

May 4, 2010 · by Karl E. Peterson

You've seen that dropdown in Shortcut property dialogs that offers to tell the target application whether to start as a Normal Window, Minimized or Maximized, right? Have you also noticed that your Classic VB apps don't pay any attention whatsoever to how that property is set? If you'd like to find out how to do just that, and more, read on.

At first, it may seem like a bug of sorts, that those properties aren't automatically honored. But if they were, this column would probably be on how to override the settings, rather than follow them. There are always good reasons to want to perform in a given way, but in the absence of such reasons it's certainly incumbent on you to run exactly as the user desires. This means you'll want to ask the OS how to optimally start up.

Windows offers the GetStartupInfo API function for just that purpose, as it returns a STARTUPINFO structure filled with instructions on how to present your application:

```
Private Type StartupInfo
   cb As Long
   lpReserved As Long
   lpDesktop As Long
   lpTitle As Long
   dwX As Long
   dwY As Long
   dwXSize As Long
   dwYSize As Long
   dwXCountChars As Long
   dwYCountChars As Long
   dwFillAttribute As Long
   dwFlags As Long
   wShowWindow As Integer
   cbReserved2 As Integer
   lpReserved2 As Long
   hStdInput As Long
   hStdOutput As Long
   hStdError As Long
End Type
```

Call GetStartupInfo by first declaring a STARTUPINFO variable and setting its cb element equal to its length:

```
Private m_si As StartupInfo

Private Sub Class_Initialize()
   ' This never changes, so just grab it once.
   m_si.cb = Len(m_si)
   Call GetStartupInfo(m_si)
End Sub
```

The fields returned in STARTUPINFO fall into three rough categories -- most often applying either just to GUI apps or just to console apps. A couple fields (lpDesktop, lpTitle) are more general in nature. And one field, dwFlags, tells you which of the rest you need to pay attention to.

For example, if dwFlags includes STARTF_USEPOSITION, you will want to position the Left/Top of your main window at dwX/dwY. And if dwFlags includes STARTF_USESIZE, then your main window should be dwXSize in Width and dwYSize in Height. Also, you should use the wShowWindow element to set your initial WindowState, if dwFlags includes STARTF_USESHOWWINDOW.

I've wrapped all these fields up in a drop-in ready class module that you can add to your projects. While this is a very straight-forward API function to call and it's equally easy to interpret, there are various checks and tweaks you would want to routinely apply to use its results in Classic VB. Wrapped up in my CStartupInfo class module, it's as easy to use as this:

```
Public Sub Main()
   Dim si As CStartupInfo
   Dim frm As FStartupDemo

   Set si = New CStartupInfo
   Set frm = New FStartupDemo

   ' Check for requested size/position.
   If (si.Left <> 0) And (si.Top <> 0) Then
      frm.Move si.Left, si.Top
   End If
   If (si.Width <> 0) Then
      frm.Width = si.Width
   End If
   If (si.Height <> 0) Then
      frm.Height = si.Height
   End If

   ' Set to requested WindowState and show.
   frm.WindowState = si.WindowState
   frm.Show
End Sub
```

As I said, there are reasons to put these structure elements behind class properties. For example, the position elements are returned in pixels, and you'll most likely want them in twips for direct assignment to your Form properties. And they shouldn't be used at all if the proper flags aren't set, in which case CStartupInfo returns 0 for the requested property. Here's how I handled the Left and Width properties, allowing the option to avoid the conversion to twips on request:

```
Public Property Get Left(Optional Pixels As Boolean) As Long
   ' If dwFlags specifies STARTF_USEPOSITION:
   ' The x-offset of the upper left corner, in pixels.
   If m_si.dwFlags And STARTF_USEPOSITION Then
      If Pixels Then
         Left = m_si.dwX
      Else
         Left = m_si.dwX * Screen.TwipsPerPixelX
      End If
   End If
End Property

Public Property Get Width(Optional Pixels As Boolean) As Long
   ' If dwFlags specifies STARTF_USESIZE:
   ' The width of a new window, in pixels.
   If m_si.dwFlags And STARTF_USESIZE Then
      If Pixels Then
         Width = m_si.dwXSize
      Else
         Width = m_si.dwXSize * Screen.TwipsPerPixelX
      End If
   End If
End Property
```

The WindowState property offered by CStartupInfo requires a bit more interpretation, reducing the numerous potential wShowWindow values down to just three. This same return element is also used to interpret the Visible property:

```
Public Property Get WindowState() As FormWindowStateConstants
   ' If dwFlags specifies STARTF_USESHOWWINDOW:
   ' Can be any of the values that can be specified in the nCmdShow
   ' parameter for the ShowWindow function, except for SW_SHOWDEFAULT.
   If m_si.dwFlags And STARTF_USESHOWWINDOW Then
      Select Case m_si.wShowWindow
         Case SW_SHOWMINIMIZED, SW_MINIMIZE, _
              SW_SHOWMINNOACTIVE, SW_FORCEMINIMIZE
            WindowState = vbMinimized
         Case SW_SHOWMAXIMIZED, SW_MAXIMIZE
            WindowState = vbMaximized
         Case Else
            WindowState = vbNormal
      End Select
   End If
End Property

Public Property Get Visible() As Boolean
   ' If dwFlags specifies STARTF_USESHOWWINDOW:
   ' Synthesized based on SW_* flags
   If m_si.dwFlags And STARTF_USESHOWWINDOW Then
      Visible = Not (m_si.wShowWindow = SW_HIDE)
   End If
End Property
```

You may be wondering where all these properties could possibly be set. After all, the Shortcut property dialog only offers a WindowState analog. Well, the place I've run into them most often is with programmatic application spawning, either using CreateProcess or VB's own Shell function.

Shell offers more than the three simple WindowState values we're most accustomed to, also allowing us to do other interesting things like spawn a new app invisibly. This can be very useful when firing up background processes that shouldn't distract the user at all. CreateProcess takes this a step further in allowing us to not only specify a ShowWindow value, but to also exactly position the new app. I've used that capability in the past to tile multiple spawned instances across a display.

As always, you can download the complete StartupInfo sample on my website. In this column, I've focused mostly on the most common properties -- those that relate to GUI apps. But GetStartupInfo also supplies a number of hints to console apps that some of you will find useful too.

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.