# Chef's Surprise

Click & Retrieve Source CODE!

***This column dishes up a healthy portion of tips to satisfy your VB3 appetite.***

by Karl E. Peterson

Before you start in on this column, I'll run through this month's specials. Because most of this issue of *VBPJ* is about VB4, the editors have asked that I concentrate this column mainly on VB3. As luck would have it, my columns often run over their allotted space, so there are several tips stacked up on the shelf. This presents the opportunity to whip up something creative from the back of the cupboard to serve the VB3 users out there.

I'll begin with two subclassing techniques (for more background, see the September 1995 feature, "Subclass Your Way Around VB's Limitations," by Jonathan Wood and I). Both use the MSGHOOK.VBX written by Zane Thomas, which originally appeared in the second edition of *Visual Basic How-To* (Waite Group Press) but was modified for the subclassing article. The more current *Visual Basic 4 How-To* provides both 16- and 32-bit OCX versions of MsgHook. The new OLE controls offer more options over how messages are handled, so the VBX was updated to use the same syntax and provide the same functionality. The sample code and control discussed in this column is available online in a file called SUBCLS.ZIP. Download the file from *VBPJ's* Development Exchange on the World Wide Web at http://www.windx.com, or from either the *VBPJ* CompuServe Forum or MSN site. For details, see "How to Reach Us" in Letters to the Editor.

Next, I'll show you how to emulate the Win32 Sleep API function with VB3 or 16-bit VB4 code. This technique was employed by the 16-bit demo of the CProgressBar class that I presented in the March 1996 Programming Techniques column, "Making Progress" (download PT0396.ZIP for the complete listing).

Finally, I'll provide a quick look at a new function provided in VB4 that enables you to test for Unicode-induced string corruption, among other things (download PT0596.ZIP for a demo project).

## FORCING AN APP TO REMAIN MINIMIZED

Some applications don't need a windowed user interface. You would typically write these to process large quantities of data in the background, or to sit idly by monitoring and reacting to system events. If you've written such an application, you may decide that it's best for it to remain minimized either as an icon or in the Windows 95 taskbar. However, Visual Basic doesn't provide a convenient means of doing so. If a form's WindowState is reset to Minimized whenever a Form_Resize event occurs, there's still an ugly flash as the form first restores to Normal or Maximized.

Hooking the WM_QUERYOPEN message provides a means for your form to be notified *before* its state is altered (see Listing 1). An application can notify Windows that the icon may be opened by returning a nonzero value, or prevent the icon from opening by returning zero. To implement this technique in Visual Basic, simply return zero in the Result parameter when the Message event is fired. There is no need to invoke the original window procedure.

As a precaution, do not perform any actions that would cause an activation or alter the focus while processing this message. To provide some degree of user interaction with your iconized application, implement the code presented in the September 1995 article to add one or several menu options to the form's system menu. If you download SUBCLS.ZIP, you will find this project saved as MINAPP, and the system menu project saved as SYSCMD.

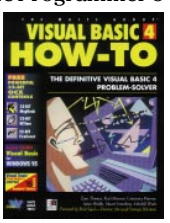
## DETECTING SYSTEM COLOR CHANGES

Often when painting custom elements on your forms, you want to use the default system colors so they will blend in nicely with the appearance of the rest of the screen. Windows offers the GetSysColor API call to retrieve any of the colors that the user can set in Control Panel for items such as buttons, title bars, borders, and so on. But, what if the user alters his or her color preferences while your program is running? For performance reasons, you can store the system colors in an array, allowing you to draw with them immediately rather than having to call the GetSysColor API each time. If you decide to do this, you need to update the contents of the array to reflect the changes in the system's color preferences.


FIGURE 1 ***Trouble Detected!*** *The UniCorruption function could detect corruption occurring as strings are converted back and forth between Unicode and ANSI.*

Hooking WM_SYSCOLORCHANGE provides notification whenever the system colors change. This message is closely followed with a WM_PAINT message that will fire the Form_Paint event in your project. By retrieving each of the system colors during the hooked Message event, the color

*Karl E. Peterson is a GIS analyst with a regional transportation planning agency and a member of the* Visual Basic Programmer's Journal *Technical Review Board. Based in Vancouver, Washington, he's also an independent programming consultant and a writer. Karl coauthored* Visual Basic 4 How-To, *from Waite Group Press. Online, he's a section leader in the* VBPJ *Forum 32-Bit Bucket and a Microsoft MVP in the MSBASIC Forum. Contact Karl in either CompuServe location at 72302,3707.*



array is refreshed prior to the painting (see Listing 2). There is no need to invoke the original window procedure, as VB has never reacted to this message.

Notice how the array of colors is initially filled. Rather than duplicate the looping code, SendMessage is employed to fire MsgHook's Message event during the Form_Load event, by sending the WM_SYSCOLORCHANGE message to the main form.

VB3

```
Option Explicit

' We want to catch WM_QUERYOPEN
Const WM_QUERYOPEN = &H13

Sub Form_Load ()
   '
   ' Ensure minimized state
   ' (could also be set at design
   ' time).  Set MinButton and
   ' MaxButton to False at design
   ' time for cleaner control menu.
   '
   Me.WindowState = 1  'Minimized
   '
   ' Setup MsgHook
   '
   MsgHook.HwndHook = Me.hWnd
   MsgHook.Message(WM_QUERYOPEN) = _
      True
End Sub

Sub MsgHook_Message (Msg As Integer, _
      wParam As Integer, _
      lParam As Long, Result As Long)
   '
   ' Prevent icon from restoring by
   ' returning 0.  No need to
   ' invoke default window procedure.
   '
   If Msg = WM_QUERYOPEN Then
      Result = 0
   End If
End Sub
```

LISTING 1 ***Remain Minimized, Please.*** *You can use MsgHook to prevent an application from being restored to either a normal or maximized state. In a project like this, you'd probably also want to alter the system menu, to allow for some degree of interaction with your users.*

VB3

```
Option Explicit

' Windows message to watch for
Const WM_SYSCOLORCHANGE = &H15

' Win16 API calls
Declare Function GetSysColor Lib _
  "User" (ByVal nIndex As _
  Integer) As Long
Declare Function SendMessage Lib _
  "User" (ByVal hWnd As _
  Integer, ByVal wMsg As Integer, _
  ByVal wParam As Integer, lParam _
  As Any) As Long

' System Colors
Const COLOR_SCROLLBAR = 0
Const COLOR_BACKGROUND = 1
Const COLOR_ACTIVECAPTION = 2
Const COLOR_INACTIVECAPTION = 3
Const COLOR_MENU = 4
Const COLOR_WINDOW = 5
Const COLOR_WINDOWFRAME = 6
Const COLOR_MENUTEXT = 7
Const COLOR_WINDOWTEXT = 8
Const COLOR_CAPTIONTEXT = 9
Const COLOR_ACTIVEBORDER = 10
Const COLOR_INACTIVEBORDER = 11
Const COLOR_APPWORKSPACE = 12
Const COLOR_HIGHLIGHT = 13
Const COLOR_HIGHLIGHTTEXT = 14
Const COLOR_BTNFACE = 15
Const COLOR_BTNSHADOW = 16
Const COLOR_GRAYTEXT = 17
Const COLOR_BTNTEXT = 18
Const COLOR_INACTIVECAPTIONTEXT = 19
Const COLOR_BTNHIGHLIGHT = 20

' Array to hold system colors
Dim SysColor(COLOR_SCROLLBAR To _
  COLOR_BTNHIGHLIGHT) As Long

Sub Form_Load ()
  '
  ' Setup MsgHook control
  '
  MsgHook.HwndHook = Me.hWnd
  MsgHook.Message(WM_SYSCOLORCHANGE)_
     = True
  '
  ' Preload system color array
  '
  Dim nRet As Long
  nRet = SendMessage(Me.hWnd, _
     WM_SYSCOLORCHANGE, 0, 0&)
End Sub

Sub MsgHook_Message (Msg As Integer, _
     wParam As Integer, _
     lParam As Long, Result As Long)
  Dim i As Integer
  If Msg = WM_SYSCOLORCHANGE Then
     '
     ' Update color table.
     '
     For i = COLOR_SCROLLBAR To _
        COLOR_BTNHIGHLIGHT
        SysColor(i) = GetSysColor(i)
     Next i
  End If
End Sub
```

LISTING 2 ***Detect System Color Changes.*** *Use this code with MsgHook to provide a method for your application to store an array of system colors that will be updated whenever the user changes settings in Control Panel. This will automatically coordinate your application with the colors the user selects.*

## ADDING A DELAY

In the March 1996 Programming Techniques column, I presented the CProgressBar class and a demo project that used the (Win32) Sleep API call to add delays while incrementing an imitation progress bar. Here's a quickie trick you can use in VB3 or VB4 to emulate the Sleep function with your 16-bit code. The Win16 API includes a GetTickCount function that retrieves the number of milliseconds that have elapsed since Windows was started. This Sleep subroutine will wait at least the requested number of milliseconds before returning:

```
Declare Function GetTickCount _
   Lib "User" () As Long

Sub Sleep(dwMS As Long)
   Dim AllDone As Long
   '
   ' Enter an idle loop for
   ' dwMS milliseconds.
   '
   AllDone = GetTickCount() + _
      dwMS
   Do While GetTickCount < _
      AllDone
      DoEvents
   Loop
End Sub
```

Two considerations are in order when using this routine. First, the value returned by GetTickCount wraps around to zero after Windows has been running for approximately 49 days (when you're done laughing, remember that your application may be running in NT). The likelihood of this happening is slim, but should be taken into account when writing applications that run 24 hours a day, seven days a week. The other consideration is that the DoEvents in the idle loop may allow other processes to consume the processor, causing the delay to be longer than requested.

## KEYWORD OF THE MONTH: STRCONV

One of the more interesting new keywords in VB4 is StrConv. It does essentially what the name implies—it converts a string. There are a nine different conversions StrConv will perform, but

three stand out as immediately useful. Two conversions duplicate the LCase and UCase functions, and four others perform double-byte/single-byte character set conversions and Katakana/Hiragana conversions. Of the three standouts, the most popular one will probably be vbProperCase, which converts the first letter of each word in a string to upper case, and all other letters to lower case.

But, even more interesting are the two conversions between Unicode and ANSI. These can be useful when you are moving data between Byte arrays and Strings. For example, this code will compress a Unicode string into its ANSI representation within a Byte array:

```
Dim b() As Byte
b = StrConv(Text1.Text, vbFromUnicode)
```

A conversion such as this may be useful when you need to operate on the individual bytes within the string, or simply when you want to save space before storing data. Caution: beware that any time Windows converts between Unicode and ANSI, there is a potential for data corruption. This can occur if there's not a direct correlation between Unicode and ANSI within the default code page in use on the host computer. This will generally not be the case in the United States, but other locales may be less likely to have direct 1:1 mapping between the character sets. VB4/32 forces conversion of Strings from Unicode to ANSI whenever they're passed to DLLs, including such seemingly innocuous activities as placing text in a text box or upon the clipboard. To convert such a compacted Byte array back to a String, just use the related option, vbUnicode, as shown here:

```
Text1.Text = StrConv(b, vbUnicode)
```

VB4

```
Public Function UniCorruption() As Boolean
  Dim b() As Byte
  Dim i As Integer
  Dim s1 As String
  Dim s2 As String
  '
  ' Create string containing all 256 ANSI chars.
  '
  s1 = Space(256)
  For i = 0 To 255
     Mid(s1, i + 1, 1) = Chr(i)
  Next i
  '
  ' Convert from Unicode to ANSI, then back.
  '
  b = StrConv(s1, vbFromUnicode)
  s2 = StrConv(b, vbUnicode)
  '
  ' Test 1: String equivalency.
  '
  If s1 <> s2 Then
     UniCorruption = True
     Exit Function
  End If
  '
  ' Test 2: Proper values in byte array.
  '
  For i = 0 To 255
     If b(i) <> i Then
        UniCorruption = True
        Exit Function
     End If
  Next i
End Function
```

LISTING 3 ***Detecting Unicode Conversion Corruption.*** *This code should detect whether you are likely to experience corruption of your string data due to conversions between Unicode and ANSI.*

Given the potential for corruption, this is still a useful function because it can in fact detect corruption. If you pass a Unicode String to an ANSI Byte array and then convert that back to a different String, comparing the original and the final Strings can give you a good indication if you're in a situation where the threat of data corruption exists. Of course, you won't know for sure unless you create a string containing all possible bytes from zero to 255. Just to ensure "complementary" corruption isn't occurring in both directions, one more test is advised. After converting the first String to a Byte array, test each element of the array, checking for the anticipated values.

Here, in the United States, the UniCorruption function returns False on all the systems I have available to test it on (see Listing 3). I would greatly appreciate hearing from you if this function returns True where you are. Many folks are upset with the possibility of this corruption, and it would be most interesting to find out if it's actually occurring. If I do hear back from anyone on this, I promise to report their findings in a future column. In any event, if your applications are shipping around the globe, it may be wise to include this function and ask your users to notify you if it ever triggers a True response. Hopefully, they'll never see a warning similar to Figure 1. ☒