# Repair Distorted, Shrunken Icons

Click & Retrieve Source CODE!

by Karl E. Peterson

## Q Where's That Small Icon?

I was disappointed by the way VB uses StretchBlt to shrink the 32x32 pixel icons assigned to my forms to the 16x16 pixel representations required for the titlebar. So I meticulously designed my own 16x16 representations, built dual-resource ICO files, and assigned these files to the forms' Icon properties. To my dismay, VB continued to display the distorted image of the shrunken 32x32 resource. How can I persuade VB to use the proper image in the proper circumstances?

A Initially, the results you describe weren't reproducible on my machine. But after asking around I found that you were far from alone in your observations. Something was clearly amiss. After further testing, and consultation with Microsoft, it became clear that VB has a bug that exhibits itself under certain circumstances. There is no simple way to assign a dual-resource icon to a form, and expect it to be properly displayed.

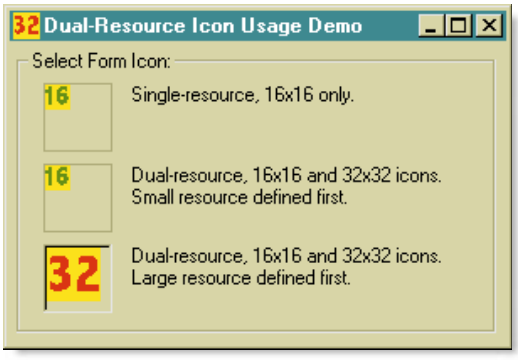


**Figure 1 Bug Leads to Improper Resource Selection.** VB incorrectly assigns a 32x32 pixel icon resource, even given a dual-resource ICO file, to a form's titlebar in most circumstances. Shown here are test results under True Color—the best possible scenario, because VB then uses the smaller resource if it is definied first in the ICO file. This bug began at least as early as VB3. Assigning a single-resource ICO, stored in an Image control, to the form's Icon property at run time is the only viable workaround.

The bug is not as noticeable when your display is set to True Color. In this situation, a dual-resource icon might display properly if constructed in a certain manner. I found that using the Microsoft Image Editor—imagedit.exe and imagedit.hlp are found in the \COMMON\TOOLS\VB\IMAGEDIT folder on the VB6 CD-ROM—a dual-resource icon could work if you define the smaller image first, and then the larger image.

If the icon images are defined in the opposite order—large first, small second—the larger icon will always be used in the titlebar. Again, these results were reproducible only under True Color settings. If a lower color depth is used by the display, you can't convince VB to display the 16x16 icon from a dual-resource ICO file (see Figure 1). Interestingly, and probably not coincidentally, VB's intrinsic Image and PictureBox controls display similar behaviors. By the way, this isn't a new bug—I found it as far back as VB3.

So, what's the answer? How can you ensure that your specially designed icons are used properly? Continue, as before, to assign the dual-resource icon to the form's Icon property at design time. This setting allows Explorer to display the desired image as needed. Then create a new ICO file that only contains the 16x16 resource. Place an Image control on your form, set its Visible property to False, and assign this single-resource ICO to the form's Icon property during the Form_Load event:

```
Private Sub Form_Load()
   Set Me.Icon = Image1.Picture
End Sub
```

With this approach, the titlebar and the taskbar display the small image, but the Alt-Tab task switcher displays the large image because that's what was assigned to the EXE at compilation. This bug warrants Microsoft's attention in VB7.

### ABOUT THIS COLUMN

Ask the VB Pro provides you with free advice on programming obstacles, techniques, and ideas. Read more answers from our crack VB Pros on the Web at www.inquiry.com/thevbpro. You can submit your questions, tips, or ideas on the site, or access a comprehensive database of previously answered questions.

### RESOURCES

Universal Coordinated Time is what used to be called Greenwich Mean Time. It's abbreviated UTC to correspond with the French translation. Some additional references:

ww2.cis.temple.edu/isworld/all_about_time.htm

www.msmango.com/time.html

www.kavouras.com/UTC.htm

www.atmos.washington.edu/utc.html

pisolo.cstv.to.cnr.it/toi/uk/utctime.html

## Q Exposing Global Objects

I'm writing a DLL that exposes some objects. I hoped to expose an object that worked something like the global Clipboard and Printer objects, which are used as a class instance but do not need to be instantiated. However, now I'm stuck with two inferior options: I can expose a normal class that can be instantiated and used, or make it global, in which case it is not used like a class but methods can be invoked simply as global functions. Is there a way to emulate objects such as the intrinsic Clipboard?

A Yes. You need to create an object, setting its Instancing property to GlobalMultiUse, and expose your global objects from this class. To demonstrate, I've created a project that replicates and extends the functionality of VB's intrinsic Clipboard object.

Start a new ActiveX DLL project. Name the default class CGlobalObjects and set its Instancing property to GlobalMultiUse. Add a new class to the project, name it CClipboard, and set its instancing to PublicNotCreatable. During the Initialize event of CGlobalObjects, instantiate an instance of CClipboard. Add a public property, named ClipboardEx, to CGlobalObjects which exposes the CClipboard object (see Listing 1).

The CClipboard class offers all the standard properties and methods of the intrinsic Clipboard object. Most are mapped directly to the intrinsic object, but one point of this exercise was to extend or otherwise modify the default behavior, offering new and more useful routines to the client application. So the new object extends the intrinsic by adding Owner and Formats properties (see Listing 2). Owner returns the window handle of the current clipboard owner, and Formats returns a read-only collection object that can be used to enumerate all the formats currently on the system clipboard. The complete code for this example can be found on The Development Exchange (see the Download Free Code box for details).

**VB5, VB6 | Expose a Global Object**

```
Option Explicit

Private m_Clip As CClipboard

' ************************************************
'  Initialization and Termination
' ************************************************
Private Sub Class_Initialize()
  Set m_Clip = New CClipboard
End Sub

Private Sub Class_Terminate()
  Set m_Clip = Nothing
End Sub

' ************************************************
'  Exposed Objects
' ************************************************
Public Property Get ClipboardEx() As CClipboard
  Set ClipboardEx = m_Clip
End Property
```

**Listing 1** This is the complete listing of a CGlobalObjects class, which in turn exposes an internal instance of the CClipboard class. Instancing for CGlobalObjects is set to GlobalMultiUse, so the ClipboardEx property is seen as a global object within VB's namespace for any project that references the resulting DLL. Client projects don't need to refer to the CGlobalObjects class.

## Q Finding Time Zone Information

How can I find the active time zone offset—from Universal Coordinated Time (UTC)—to which the computer clock is set? This value changes when you switch from Standard to Daylight

**VB5, VB6 | Create an Enhanced Global ClipboardEx Object**

```
Option Explicit

' Private variables
Private m_Fmts As CClipFormats 'PublicNotCreatable

' ************************************************
'  Initialization and Termination
' ************************************************
Private Sub Class_Initialize()
  Set m_Fmts = New CClipFormats
End Sub

Private Sub Class_Terminate()
  Set m_Fmts = Nothing
End Sub

' ************************************************
'  Extended Properties
' ************************************************
Public Property Get Formats() As CClipFormats
  ' Expose "New and Improved!" enumeration object.
  m_Fmts.Refresh
  Set Formats = m_Fmts
End Property

Public Property Get Owner() As Long
  ' Return window handle of Clipboard owner.
  ' Useful(?) if conflicts occur.
  Owner = GetClipboardOwner()
End Property

' ************************************************
'  Public Methods
' ************************************************
Public Function Clear() As Boolean
  ' Could map directly to the standard object,
  ' but here's the API equivalent.
  If OpenClipboard(0&) Then
    Clear = CBool(EmptyClipboard)
    Call CloseClipboard
  End If
End Function

Public Function GetFormat(ByVal Format As Long) _
  As Boolean
```

**Continued on page 88.**

**Listing 2** This is the complete listing of the CClipboard class, which is exposed by CGlobalObjects, and provides new and enhanced functionality over VB's intrinsic Clipboard object. Most methods were directly mapped to the intrinsic equivalent. Two methods—Clear and GetFormat—were coded directly using APIs. And two new properties, Formats and Owner, were added to provide new features.

Savings Time. I have looked in the Win.ini file and my API reference, but can find nothing.

**A** The GetTimeZoneInformation API offers the information you need, and more. Pass a variable of the TIME_ZONE_INFORMATION type to GetTimeZoneInformation, and inspect the Bias member, the first one returned, for your specific need. All translations between UTC and local time are based on this formula:

```
UTC = local time + bias
```

The bias is the difference, in minutes, between UTC and local time. Additionally, you'll want to factor in either the StandardBias—0 in most locations—or DaylightBias (–60 in most locations that observe Daylight Savings Time), depending on whether it's currently Standard or Daylight Savings Time. The return value from GetTimeZoneInformation should alert you to

**Continued from page 86.**

```
  ' Could map directly to the standard object,
  ' but here's the API equivalent.
  If OpenClipboard(0&) Then
    GetFormat = _
      CBool(IsClipboardFormatAvailable(Format))
    Call CloseClipboard
  End If
End Function

' ************************************************
'  Mapped Public Methods
' ************************************************
Public Function GetData(Optional ByVal Format As Long = _
  vbCFBitmap) As Picture
  ' Map directly to intrinsic object.
  On Error Resume Next
  Set GetData = Clipboard.GetData(Format)
End Function

Public Function SetData(ByVal NewVal As Picture, _
  Optional ByVal Format As Long = _
  vbCFBitmap) As Boolean
  ' Map directly to intrinsic object.
  On Error Resume Next
  Clipboard.SetData NewVal, Format
  SetData = (Err.Number = 0)
End Function

Public Function GetText(Optional ByVal Format As Long = _
  vbCFText) As String
  ' Map directly to intrinsic object.
  On Error Resume Next
  GetText = Clipboard.GetText(Format)
End Function

Public Function SetText(ByVal NewVal As String, _
  Optional ByVal Format As Long = _
  vbCFText) As Boolean
  ' Map directly to intrinsic object.
  On Error Resume Next
  Clipboard.SetText NewVal, Format
  SetText = (Err.Number = 0)
End Function
```

**VB4/32, VB5, VB6 | Understanding the GetTimeZoneInformation API**

```
Option Explicit

Private Declare Function GetTimeZoneInformation _
  Lib "kernel32" (lpTimeZoneInformation As _
  TIME_ZONE_INFORMATION) As Long

Private Type SYSTEMTIME
  wYear As Integer
  wMonth As Integer
  wDayOfWeek As Integer
  wDay As Integer
  wHour As Integer
  wMinute As Integer
  wSecond As Integer
  wMilliseconds As Integer
End Type

Private Type TIME_ZONE_INFORMATION
  Bias As Long
  StandardName(0 To 63) As Byte
  StandardDate As SYSTEMTIME
  StandardBias As Long
  DaylightName(0 To 63) As Byte
  DaylightDate As SYSTEMTIME
  DaylightBias As Long
End Type

Private Const TIME_ZONE_ID_INVALID = &HFFFFFFFF
Private Const TIME_ZONE_ID_UNKNOWN = 0
Private Const TIME_ZONE_ID_STANDARD = 1
Private Const TIME_ZONE_ID_DAYLIGHT = 2

Private Sub Form_Paint()
  Dim nRet As Long
  Dim tz As TIME_ZONE_INFORMATION

  Me.CurrentX = 0
  Me.CurrentY = 0
  nRet = GetTimeZoneInformation(tz)
  If nRet <> TIME_ZONE_ID_INVALID Then
    Select Case nRet
      Case TIME_ZONE_ID_UNKNOWN
        Me.Print "Time Zone Unknown!"
      Case TIME_ZONE_ID_STANDARD
        Me.Print "Standard Time..."
      Case TIME_ZONE_ID_DAYLIGHT
        Me.Print "Daylight Savings Time..."
    End Select

    Me.Print "UTC Bias: "; tz.Bias / 60; " hrs."
    Me.Print " ST Zone: "; _
      TrimNull(CStr(tz.StandardName))
    Me.Print " ST Date: "; tzDate(tz.StandardDate)
    Me.Print " ST Bias: "; tz.StandardBias; " mins."
    Me.Print " DT Zone: "; _
```

**Continued on page 90.**

**Listing 3** This single API call returns a lot of information. It can be somewhat difficult to interpret, however. Although much of it is straightforward, extracting the time and date of switches from Daylight to Standard, and vice versa, typically involves code to find the first or last Sunday of a given month in a given year.

**Continued from page 88.**

```
      TrimNull(CStr(tz.DaylightName))
    Me.Print " DT Date: "; tzDate(tz.DaylightDate)
    Me.Print " DT Bias: "; tz.DaylightBias; " mins."
  End If
End Sub

Private Function tzDate(st As SYSTEMTIME) As Date
  Dim i As Long
  Dim n As Long
  Dim d1 As Long
  Dim d2 As Long

  ' This member supports two date formats. Absolute
  ' format specifies an exact date and time when
  ' standard time begins. In this form, the wYear,
  ' wMonth, wDay, wHour, wMinute, wSecond, and
  ' wMilliseconds members of the SYSTEMTIME structure
  ' are used to specify an exact date.
  If st.wYear Then
    tzDate = _
      DateSerial(st.wYear, st.wMonth, st.wDay) + _
      TimeSerial(st.wHour, st.wMinute, st.wSecond)

  ' Day-in-month format is specified by setting the
  ' wYear member to zero, setting the wDayOfWeek member
  ' to an appropriate weekday, and using a wDay value in
  ' the range 1 through 5 to select the correct day
  ' in the month. Using this notation, the first Sunday
  ' in April can be specified, as can the last Thursday
  ' October (5 is equal to "the last").
  Else
    ' Get first day of month
    d1 = DateSerial(Year(Now), st.wMonth, 1)
    ' Get last day of month
    d2 = DateSerial(Year(d1), st.wMonth + 1, 0)

    ' Match weekday with appropriate week...
    If st.wDay = 5 Then
      ' Work backwards
      For i = d2 To d1 Step -1
        If WeekDay(i) = (st.wDayOfWeek + 1) Then
          Exit For
        End If
      Next i
    Else
      ' Start at 1st and work forward
      For i = d1 To d2
        If WeekDay(i) = (st.wDayOfWeek + 1) Then
          n = n + 1  'incr week value
          If n = st.wDay Then
            Exit For
          End If
        End If
      Next i
    End If

    ' Got the serial date!  Just format it and
    ' add in the appropriate time.
    tzDate = i + _
      TimeSerial(st.wHour, st.wMinute, st.wSecond)
  End If
End Function
```

which setting is in effect.

If, in addition, you'd like to obtain the date/time for switching between Standard and Daylight Savings Times, these can be extracted from the returned TIME_ZONE_INFORMATION structure. Embedded within this structure are two SYSTEMTIME substructures. Windows can use either of two different encoding methods to return these dates, depending on local tradition.

The absolute method is straightforward. If the wYear member is not zero, the structure specifies the exact date and time of the switch from Daylight to Standard (or vice versa) Time. If the wYear member is zero, the wDayOfWeek member contains the weekday of the switch, and the wDay member tells the relative week within the month. In other words, if wDayOfWeek is 0, and wDay is 1, then you need to determine the date for the first Sunday of the month specified in the wMonth element. When wDay is 5, this is interpreted to mean the "last" occurrence of the month. So if wDayOfWeek is 1 and wDay is 5, your task is to find the last Monday of the month.

I've wrapped this logic up in a function, tzDate, that accepts a SYSTEMTIME structure and returns the date/time encoded within it as a serial-date value (see Listing 3). The one thing to notice as you browse this routine is that the system and VB use different constant values for the day of the week. Although VB starts numbering weekdays by assigning 1 to Sunday, 2 to Monday, and so on, the system constants evaluate to one less—0 is Sunday; 1 is Monday. VBPJ

**About the Author**

Karl E. Peterson is a GIS analyst with a regional transportation planning agency and serves as a member of **Visual Basic Programmer's Journal** Technical Review and Editorial Advisory Boards. Online, he's a Microsoft MVP and a section leader on several **VBPJ** online forums. Find more of Karl's VB samples at www.mvps.org/vb.

**DOWNLOAD FREE CODE**

Download the code for this issue of **VBPJ free** from www.vbpj.com.

To get the free code for this entire issue, click on Locator+, the right-most option on the menu bar at the top of the VBPJ home page, and type **VBPJ0499** into the box. (You first need to register, for free, on DevX.) The free code for this article includes all code listings, plus the complete ClipboardEx (global objects) project and a test applet that uses the enhanced ClipboardEx object.

✪ To get the bonus code for this article, available to DevX Premier Club members, type **VBPJ0499AP** into the Locator+ field. The bonus code includes all the free code described above, plus fully coded demos showing interpretation of time zone data and a workaround for the 16x16 icon bug.