# VB ~~Can't~~ Can Do That!

Click & Retrieve Source CODE!

by Karl E. Peterson

## Q Always Display Correct Name

When you place a regular TextBox on a form at design time, the TextBox automatically places the Name of the control, as well as its unique integer, into the Text property—Text1 or Text2, for example. I created an extended-function TextBox control. I would like to duplicate the naming process, but I can't seem to get the full name that VB5 assigns to the control during the InitProperties event. I get only the Name of the control, not its unique, VB-assigned integer.

A The initial release of VB5 had a problem in this regard. If not exactly as you describe, it was similar and perhaps related. If a usercontrol instance was copied to the clipboard, pasted back onto the form, and you answered "No" to creating a control array, the Extender.Name property continued to reflect the original control's Name value rather than the newly assigned value. I reported this as a bug in VB5 late in the beta, and there wasn't time to address it prior to release, but it has been fixed in a subsequent service pack. So, if you haven't applied SP3 to VB5, it's well past time. With this patch, Extender.Name always returns the correct display name during the InitProperties event.

At other times, you might need to update your display name and add some techniques to your controls. You can instruct the Caption (or Text) property to be updated with each user keystroke through the Procedure Attributes dialog. To access this dialog, open the code window for your usercontrol, then select Procedure Attributes from the Tools menu. Select your Caption property from the Name drop-down, press the Advanced button to reveal more options, and select Caption from the Procedure ID drop-down. After you apply this modification, your Caption property Let procedure is called each time the user alters this property at design time, which allows you to update the display.

In other situations, you might not have a user-configurable caption. Consider the display of VB's intrinsic ListBox at design time. If you don't pre-assign the List property, the ListBox uses a single dummy item to display the control's Name. I recently had to replicate this behavior when I wrapped up a standard ListBox and added DragList capabilities, allowing the user to rearrange the list contents. You can download this control for free at http://www.mvps.org/ccrp.

The answer is in the little known or used AmbientChanged event. This is one of those oddball events that might or might not fire depending on the environment in which your control is hosted. AmbientChanged alerts you to changes in the Ambient object. This object is provided, in varying extents, by containers to suggest various control behaviors. To detect the suggested display name for your control, add this code:

```
Private Sub _
   UserControl_AmbientChanged( _
   PropertyName As String)
   Select Case PropertyName
      Case "DisplayName"
         Call UpdateDisplayName
      Case Else
         Debug.Print PropertyName
   End Select
End Sub
```

Watch for changes to the "DisplayName" property, and update your control appropriately. (As an interesting learning device, send the PropertyName string to the Immediate window on other calls to this event.) I chose to update the display from a separate routine; otherwise, the same code is duplicated in two other locations. The AmbientChanged event doesn't fire (for "DisplayName") when the control is first sited, so you have to call UpdateDisplayName from both InitProperties and ReadProperties as well:

```
Private Sub UpdateDisplayName()
   '
   ' Set display name into listbox,
   ' but only do this at design time!
   '
   On Error GoTo BailOut
   If Ambient.UserMode = False Then
      List1.List(0) = _
         Ambient.DisplayName
   End If
BailOut:
End Sub
```

You could use this logic in many potential controls, not just in list boxes, to display non-caption-based names at design time. Not all containers expose all Ambient properties, so it's always advisable to employ proper error trapping when querying them.

### 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 http://www.inquiry.com/thevbpro. You can submit your questions, tips, or ideas on the site, or access a comprehensive database of previously answered questions.

## Q Scrollbars Not Painting Correctly

I'm using the VScrollBar and HScrollBar intrinsic controls in VB5 under Win95 and NT. They both work fine, but the scroll areas above or below the thumb appear white in NT, not gray as they do in Win95. How can I fix this so they display properly?

A I first submitted a bug on this behavior during the VB4 beta. Now, two versions later, VB6 still hasn't corrected this seemingly simple problem. But, it's a problem easily addressed with a little subclassing.

Windows sends a WM_CTLCOLORSCROLLBAR message to the parent window of a scrollbar when the control is about to be drawn. This message allows an application to set the background color of the control by returning the desired brush Windows should use when painting the background. VB4, VB5, and VB6 all respond inappropriately to this request from Windows NT by returning the handle of a white background brush (see Figure 1).

To avoid this incorrect response, subclass the form on which you use intrinsic scrollbar(s), and "eat" the WM_CTLCOLORSCROLLBAR message. For simplicity, I'll use the freeware MsgHook control to illustrate the technique:

```
Private Const WM_CTLCOLORSCROLLBAR _
   = &H137

Private Sub Form_Load()
   Msghook.HwndHook = Me.hWnd
   Msghook.Message( _
      WM_CTLCOLORSCROLLBAR) = True
End Sub
```

That's all there is to it. By simply instructing MsgHook to intercept this message, but not reacting in any fashion whatsoever to the receipt of the message, you instruct Windows to use the default colors for these controls. If you want to observe how VB is responding to the message, you can do something like this:

```
Private Sub Msghook_Message(ByVal _
   msg As Long, ByVal wp As Long, _
   ByVal lp As Long, result _
   As Long)
   Debug.Print _
      Hex(Msghook.InvokeWindowProc( _
      msg, wp, lp))
End Sub
```

You'll see that VB is returning a Long value, presumably a brush handle, under Windows NT. Interestingly, it appears that forms running under either Windows 95 or Windows 98 never receive this message, thus explaining why the "deviant" behavior is only observed in NT. MsgHook is a freeware control, written by *VBPJ* contributing editor Zane Thomas, and is widely available on the Internet (including my site at http://www.mvps.org/vb).

## Q PIDL to Safearray

I'm trying to use Navigate2 with a WebBrowser control to navigate to one of the special folders. According to the Knowledge Base article Q167834, this cannot be done in VB because there's no way to represent a pointer to an ID list (PIDL) in VB. I assume the article is outdated, as I've been using PIDLs with VB for a pretty long time without problems. I can't believe that it can be this complicated.

A Right you are! The referenced KB article states, "Note that because it is not currently possible to represent a PIDL in Visual Basic, it is not possible to call the Navigate2 method from a Visual Basic application." Well, them thar's fightin' words to most VB developers, ain't they? By the time you read this column, I suspect Microsoft will have updated that document so as not to put VB in such a bad light.

The WebBrowser control is implemented by shdocvw.dll, a component shipped with Internet Explorer (IE) 3 and 4. The IE4 version exposes a new Navigate2 method that accepts a PIDL packed into a SAFEARRAY of bytes. PIDLs were introduced in Win95 and NT4 as a way of navigating the shell's namespace. An item ID list can be represented by one or more structures of this composition (the name SHITEMID is short for Shell Item ID):

```
typedef struct _SHITEMID {
   // mkid
    USHORT cb;       _
   // size of identifier,
   // including cb itself
    BYTE   abID[1];  _
   // variable length item identifier
} SHITEMID, * LPSHITEMID;
```

The only trick is translating this list into a data structure VB can work with. Conceptually, the equivalent VB structure would be:

```
Type SHITEMID
  cb As Integer
  abID() As Byte
End Type
```

But what you have is a pointer to multiple structures like these. The secret is to walk through the memory pointed to by the PIDL, examining the length of each list item, and re-creating the list in a byte array through a series of memory copies (see Listing 1). Because there's no way to know how many IDs are in the list, you must enter a potentially infinite loop, scanning for the terminating element. Copy the length of the current element to a size variable, and add this to a running sum that tracks the overall length of the list. On each iteration, add the running sum to the PIDL to obtain the address of the next structure. The loop terminates when the size element is zero.

When you know the overall size, most of the work is done. Redimension a byte array to hold the entire list, in addition to the two extra bytes representing the terminating zero. Call CopyMemory (an alias of RtlMoveMemory) one more time to transfer the entire list into the byte array. Because the WebBrowser's Navigate2 method expects a Variant, the last step is to assign the byte array into a Variant that is then passed to Navigate2.

So, when someone, *even Microsoft*, says "you can't do that in VB," it's always best to critically question such a claim. I'd like to extend a special thanks to Brad Martinez (http://members.aol.com/btmtz/vb/), fellow CCRP control author, for sharing the logic behind this solution. VBPJ

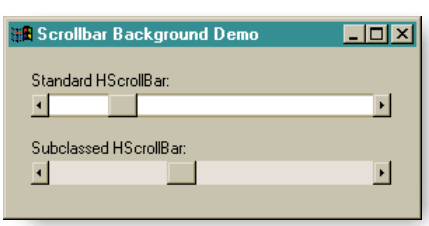


**Figure 1 Scrollbars Paint Incorrectly Under NT.** This sample shows how VB mishandles Windows' request for a background color as scrollbars are drawn. This message appears to occur only under NT, and not at all under either Windows 95 or 98, although it's documented for all three environments. To enforce proper background painting, hook the WM_CTLCOLORSCROLLBAR message and "eat" it as it's sent to your form.

## About the Author

Karl E. Peterson is a GIS analyst with a regional transportation planning agency and serves as a member of the **VBPJ** Technical Review and Editorial Advisory Boards. Based in Vancouver, Wash., he's also an independent programming consultant specializing in ActiveX controls, and contributes to various journals. Karl coauthored **Visual Basic 4 How-To,** from Waite Group Press. Online, he's a Microsoft MVP, and a section leader of several **VBPJ** online forums. In his spare time, he's an official member of the Common Controls Replacement Project (CCRP), a group dedicated to freeware control offerings. Find more of Karl's VB samples at http://www.mvps.org/vb.

## CODE ONLINE

You can find all the code published in this issue of **VBPJ** on The Development Exchange (DevX) at http://www.vbpj.com. For details, please see "Get Extra Code in DevX's Premier Club" in Letters to the Editor.

**VB ~~Can't~~ Can Do That!**
**Locator+ Codes**
Listings for the entire issue, plus a sample project showing a correction of VB's errant response to the WM_CTLCOLORSCROLLBAR message (free Registered Level): VBPJ1098
✪ Listings for this article only, the scrollbar project described above, and a sample project demonstrating use of the WebBrowser's Navigate2 method (subscriber Premier Level): AP1098

**VB4/32 | VB5 | VB6 | Display Any Special Folder in a WebBrowser Control**

```
Private Declare Function _
    SHGetSpecialFolderLocation _
    Lib "shell32.dll" (ByVal hWndOwner As Long, _
    ByVal nFolder As SpecialShellFolderIDs, pidl _
    As Long) As Long
Private Declare Sub CoTaskMemFree Lib _
    "ole32.dll" (ByVal pv As Long)
Private Declare Sub CopyMemory Lib "kernel32" _
    Alias "RtlMoveMemory" (pDest As Any, pSource _
    As Any, ByVal dwLength As Long)

Private Const NOERROR = 0

Public Enum SpecialShellFolderIDs
    CSIDL_DESKTOP = &H0
    CSIDL_INTERNET = &H1
    CSIDL_PROGRAMS = &H2
    CSIDL_CONTROLS = &H3
    CSIDL_PRINTERS = &H4
    CSIDL_PERSONAL = &H5
    CSIDL_FAVORITES = &H6
    CSIDL_STARTUP = &H7
    CSIDL_RECENT = &H8
    CSIDL_SENDTO = &H9
    CSIDL_BITBUCKET = &HA
    CSIDL_STARTMENU = &HB
    CSIDL_DESKTOPDIRECTORY = &H10
    CSIDL_DRIVES = &H11
    CSIDL_NETWORK = &H12
    CSIDL_NETHOOD = &H13
    CSIDL_FONTS = &H14
    CSIDL_TEMPLATES = &H15
    CSIDL_COMMON_STARTMENU = &H16
    CSIDL_COMMON_PROGRAMS = &H17
    CSIDL_COMMON_STARTUP = &H18
    CSIDL_COMMON_DESKTOPDIRECTORY = &H19
    CSIDL_APPDATA = &H1A
    CSIDL_PRINTHOOD = &H1B
    CSIDL_ALTSTARTUP = &H1D            ' // DBCS
    CSIDL_COMMON_ALTSTARTUP = &H1E     ' // DBCS
    CSIDL_COMMON_FAVORITES = &H1F
    CSIDL_INTERNET_CACHE = &H20
    CSIDL_COOKIES = &H21
    CSIDL_HISTORY = &H22
End Enum
```

```
Public Function SHNav2SpecialFolder(ByVal _
    nFolder As SpecialShellFolderIDs, wb _
    As WebBrowser) As Boolean
    Dim pidl As Long
    Dim idSize As Integer
    Dim idlSum As Integer
    Dim Buffer() As Byte
    Dim SafeArray As Variant
    '
    ' Get the pointer to the folder's item ID list
    ' from its specified folder ID.
    '
    If SHGetSpecialFolderLocation(0&, nFolder, pidl) = _
      NOERROR Then
      If pidl Then
        '
        ' Determine size of entire ID list.
        '
        Do
          Call CopyMemory(idSize, ByVal (pidl + _
            idlSum), 2)
          idlSum = idlSum + idSize
          If idSize = 0 Then Exit Do
        Loop
        '
        ' Create buffer with 2 extra bytes for the
        ' zero terminator, and copy ID list into it.
        '
        ReDim Buffer(0 To idlSum + 1)
        CopyMemory Buffer(0), ByVal pidl, idlSum + 2
        '
        ' Pass SAFEARRAY to WebBrowser.
        '
        SafeArray = Buffer
        wb.Navigate2 SafeArray
        '
        ' Free the memory the shell allocated for the
        ' ID list.
        '
        Call CoTaskMemFree(pidl)
        SHNav2SpecialFolder = True
      End If
    End If
End Function
```

**Listing 1** Adding this routine and these declares to a standard code module enables your app to request WebBrowser controls to display any special folder by ID. Simply pass one of the SpecialShellFolderIDs constants, along with a reference to a WebBrowser control, and the SHNav2SpecialFolder function handles the dirty work.