Welcome to the Ninth Edition of the *VBPJ* Technical Tips Supplement!

These tips and tricks were submitted by professional developers using Visual Basic 3.0 through 6.0, Visual Basic for Applications (VBA), and Visual Basic Script (VBS). The editors at *Visual Basic Programmer's Journal* compiled the tips. Special thanks to *VBPJ* Technical Review Board member Karl E. Peterson for testing all the code. Instead of typing the code published here, download the tips for free from the Code section on *VBPJ*'s Web site at www.vbpj.com.

If you'd like to submit a tip to *VBPJ*, please send it to User Tips, Fawcette Technical Publications, 209 Hamilton Ave., Palo Alto, California, USA, 94301-2500. You can also fax it to 650-853-0230 or send it electronically to vbpjedit@fawcette.com. Please include a clear explanation of what the technique does and why it's useful, and indicate if it's for VBA, VBS, VB3, VB4 16- or 32-bit, VB5, or VB6. Please limit code length to 20 lines. Don't forget to include your e-mail and mailing address. If we publish your tip, we'll pay you $25 or extend your *VBPJ* subscription by one year.

VB5, VB6

Level: Advanced

Restore Errant Focus to RichTextBox

If a RichTextBox control has the focus in a Multiple Document Interface (MDI) child form, it doesn't properly regain the focus after your application loses and regains focus. To fix this, you must subclass the MDI form and watch for the WM_ACTIVATE event. Set a Public variable equal to the window handle of the RichTextBox in its GotFocus event, and set that variable to zero in its LostFocus event. Use the SetFocus API function to force the focus to the RichTextBox:

```
'Module Code:
Private Declare Function CallWindowProc Lib _
   "user32" Alias "CallWindowProcA" (ByVal _
   lpPrevWndFunc As Long, ByVal hWnd As Long, _
   ByVal Msg As Long, ByVal wParam As Long, _
   ByVal lParam As Long) As Long
Private Declare Function SetFocusAPI Lib _
   "user32" Alias "SetFocus" (ByVal hWnd As Long) As Long
Public Declare Function SetWindowLong Lib _
   "user32" Alias "SetWindowLongA" (ByVal hWnd _
   As Long, ByVal nIndex As Long, ByVal _
   dwNewLong As Long) As Long
Private Declare Sub CopyMem Lib "kernel32" Alias _
   "RtlMoveMemory" (Destination As Any, Source _
   As Any, ByVal Length As Long)
Public Const WM_ACTIVATE = &H6
Public Const WA_INACTIVE = 0
Public Const GWL_WNDPROC = (-4)
Public origWndProc As Long
Public lFocusHandle As Long

Public Function AppWndProc(ByVal hWnd As Long, _
   ByVal Msg As Long, ByVal wParam As Long, _
   ByVal lParam As Long) As Long
   Select Case Msg
      Case WM_ACTIVATE
         If WordLo(wParam) <> WA_INACTIVE Then
            If lFocusHandle Then SetFocusAPI _
               lFocusHandle
         End If
   End Select
   AppWndProc = CallWindowProc(origWndProc, _
      hWnd, Msg, wParam, lParam)
End Sub

Private Function WordLo(LongIn As Long) As Integer
   Call CopyMem(WordLo, ByVal VarPtr(LongIn), 2)
End Function

'MDI Form code:
Private Sub MDIForm_Load()
   origWndProc = SetWindowLong(Me.hWnd, _
      GWL_WNDPROC, AddressOf AppWndProc)
End Sub

Private Sub MDIForm_Unload(Cancel As Integer)
   SetWindowLong Me.hWnd, GWL_WNDPROC, origWndProc
End Sub

'MDIChild Form code:
Private Sub RichTextBox1_GotFocus()
   lFocusHandle = RichTextBox1.hWnd
End Sub

Private Sub RichTextBox1_LostFocus()
   lFocusHandle = 0
End Sub
```

—Matt Hart, Tulsa, Oklahoma

VB3 and up

Level: Beginning

☆☆☆☆☆ Five Star Tip

Close All MDI Children Simply

This code allows you to close all the MDI child forms in an MDI form at once. First, create a menu item in the MDI form, then paste in this code:

```
Private Sub mnuCloseAll_Click()
   Screen.MousePointer = vbHourglass
   Do While Not (Me.ActiveForm Is Nothing)
      Unload Me.ActiveForm
   Loop
   Screen.MousePointer = vbDefault
End Sub
```

Once the user clicks on that menu item, the MDI child forms will close.

—Tai Choo Tack, Port Klang, Selangor, Malaysia

VB4 16/32, VB5, VB6
Level: Beginning

Consolidate Global Data
We all know that using global variables in our projects is not considered good programming practice. Nevertheless, everyone does it. Sometimes you need to introduce public variables for storing initialization (INI) file names, Registry keys, shared constant values, and other common needs. Well, you can consolidate all your public variables through one public class, which encapsulates your variables. Store global data inside this class as module-level variables and constants or public Enums:

```
Option Explicit

Private m_sINIFile As String
Private Const QUERY_TIMEOUT As Long = 120

Private Sub Class_Initialize()
   m_sINIFile = App.Path & "\" & App.Title & ".ini"
End Sub

Public Property Get INIFile() As String
   INIFile = m_sINIFile
End Property

Public Property Get REGKey() As String
   REGKey = "\SOFTWARE\MyKey"
End Property

Public Property Get QueryTimeOut() As Long
   QueryTimeOut = QUERY_TIMEOUT
End Property
```

Set the class's Instancing property to Private, so it's not visible to outside projects. After you expose the variables as properties or enumerations of a class, you can create this class on a module level or on a project level and use the data stored in this class:

```
Dim objData As CProjectData
Set objData = New CProjectData

If Dir$(objData.INIFile, vbNormal) = "" Then
   MsgBox objData.INIFile _
      & " - INI file not found!"
End If
```

By doing this, you don't have to memorize the names of those public variables. As you type the name of your data class, VB's IntelliSense helps you pick the class properties you need.

—Oleg Melnikov, Los Angeles, California

VB4 32, VB5, VB6
Level: Intermediate

☆☆☆☆☆ Five Star Tip

Show the Standard File Properties Dialog
If your program has an Explorer shell-style interface, you probably want to supply the standard File | Properties dialog. Do this by using the ShellExecuteEx API function:

```
Private Type SHELLEXECUTEINFO
   cbSize As Long
   fMask As Long
   hWnd As Long
   lpVerb As String
   lpFile As String
   lpParameters As String
   lpDirectory As String
   nShow As Long
   hInstApp As Long
   lpIDList As Long
   lpClass As String
   hkeyClass As Long
   dwHotKey As Long
   hIcon As Long
   hProcess As Long
End Type

Private Declare Function ShellExecuteEx Lib _
   "shell32" (lpSEIAs SHELLEXECUTEINFO) As Long
Private Const SEE_MASK_INVOKEIDLIST = &HC

Private Sub Form_Click()
   Call ShowFileProperties( _
      "c:\windows\system\msvbvm50.dll")
End Sub

Private Sub ShowFileProperties(ByVal aFile As String)
   Dim sei As SHELLEXECUTEINFO
   sei.hWnd = Me.hWnd
   sei.lpVerb = "properties"
   sei.lpFile = aFile
   sei.fMask = SEE_MASK_INVOKEIDLIST
   sei.cbSize = Len(sei)
   ShellExecuteEx sei
End Sub
```

—Matt Hart, Tulsa, Oklahoma

VB4 32, VB5, VB6
Level: Intermediate

☆☆☆☆☆ Five Star Tip

Toggle ListView Headers Between Flat and 3-D
Whenever you want a ListView control with flat, nonclickable headers, use this code to toggle the header style. Flat headers don't give users the impression of sortability that 3-D headers do:

```
Private Declare Function GetWindowLong Lib _
   "user32" Alias "GetWindowLongA" (ByVal hWnd _
   As Long, ByVal nIndex As Long) As Long
Private Declare Function SetWindowLong Lib _
   "user32" Alias "SetWindowLongA" (ByVal hWnd _
   As Long, ByVal nIndex As Long, ByVal _
   dwNewLong As Long) As Long

Private Const GWL_STYLE = (-16)
Private Const LVM_FIRST = &H1000
Private Const LVM_GETHEADER = (LVM_FIRST + 31)
Private Const HDS_BUTTONS = &H2

Call ToggleHeader(ListView1.hWnd)

Private Sub ToggleHeader(lsvhWnd As Long)
   Dim hHeader As Long, lStyle As Long
   hHeader = SendMessage(lsvhWnd, _
      LVM_GETHEADER, 0, ByVal 0&)
   lStyle = GetWindowLong(hHeader, GWL_STYLE)
   SetWindowLong hHeader, GWL_STYLE, lStyle Xor HDS_BUTTONS
End Sub
```

—Matt Hart, Tulsa, Oklahoma

VB6
Level: Intermediate

React to Font-Changed Events
Use WithEvents to perform an action when you change any font properties of a specific control or form. Make sure you set the OLE Automation reference in the References dialog:

```
' Declaration section
Private WithEvents fntAny As StdFont

Private Sub fntAny_FontChanged(ByVal PropertyName _
   As String)
   Select Case PropertyName
      Case "Name"
         ' Perform specific action
      Case "Size"
         ' Perform specific action
      Case "Italic"
         ' Perform specific action
      Case "Bold"
         ' Perform specific action
      Case "Underline"
         ' Perform specific action
      '...
      '...
      ' Similarly, you can extend the
      ' functionality for each font property.
   End Select
End Sub
```

You only have to assign any form or control's Font reference to fntAny. For example, if you want to trap the changes in the form's font attributes, add this code to the Form_Load event:

```
Set fntAny = Me.Font
' If a control, then Control.Font
```

—Badari Syam Mysore, Scotch Plains, New Jersey

VB4 32, VB5, VB6
Level: Beginning

Determine Which ListView Column was Checked
When using the ListView control in list mode (Listview.View = lvwList), no property indicates which column the user clicked on within the selected row. The ListView's HitTest method returns only a reference to the ListItem the user clicked on, not the specific subitem. Use the SendMessage API function in the ListView's MouseUp or MouseDown event to provide this information:

```
Private Declare Function SendMessage Lib _
   "user32" Alias "SendMessageA" (ByVal hWnd As _
   Long, ByVal wMsg As Long, ByVal wParam As _
   Long, lParam As Any) As Long

Private Const LVM_SUBITEMHITTEST As Long = 4153

Private Type POINTAPI
   X As Long
   Y As Long
End Type

Private Type LVHITTESTINFO
   pt As POINTAPI
   lngFlags As Long
   lngItem As Long
   lngSubItem As Long
End Type

Private Sub ListView1_MouseUp(Button As Integer, _
   Shift As Integer, X As Single, Y As Single)
   Dim hti As LVHITTESTINFO
   Dim lngRet As Long
   hti.pt.X = X / Screen.TwipsPerPixelX
   hti.pt.Y = Y / Screen.TwipsPerPixelY
   lngRet = SendMessage(ListView1.hWnd, _
      LVM_SUBITEMHITTEST, 0&, hti)
   Debug.Print "Row=" & hti.lngItem,
   Debug.Print "Col=" & hti.lngSubItem
End Sub
```

—Brian Pursley, Cincinnati, Ohio

VB4 32, VB5, VB6
Level: Intermediate

Convert File Size Into Proper Strings
Use this small API from the shlwapi (Shell Windowing API) DLL that ships with Internet Explorer (IE) to help you convert file size in bytes into proper strings such as "1.41 KB" or "1.32 MB." You need IE for this because it calls shlwapi.dll, which is present on all NT4, Windows 95+IE, and Windows 98 systems:

```
Private Declare Function StrFormatByteSize Lib _
   "shlwapi" Alias "StrFormatByteSizeA" (ByVal _
   dw As Long, ByVal pszBuf As String, ByRef _
   cchBuf As Long) As String

Public Function FormatKB(ByVal Amount As Long) As String
   Dim Buffer As String
   Dim Result As String
   Buffer = Space$(255)
   Result = StrFormatByteSize(Amount, Buffer, _
      Len(Buffer))
   If InStr(Result, vbNullChar) > 1 Then
      FormatKB = Left$(Result, InStr(Result, _
         vbNullChar) - 1)
   End If
End Function
```

—Deepu Chandy Thomas, Kerala, India

VB3 and up
Level: Beginning

Create Better Button Arrows
Many times developers use a form's default font arrow characters for To and From buttons between listboxes. To give your app a nicer, more solid look, do what Microsoft does—use the Marlett font, which is added by default to all Windows 95, 98, and NT4 and later installations.

Instead of giving a To button the caption ">," change the font to Marlett and type "3" for the Caption property. Instead of a From button being a skeletal "<," use Marlett and type "4" in the Caption property. For nice, solid Up and Down arrows, use Marlett "5" and "6."

—Robert Smith, Kirkland, Washington

VB3 and up
Level: Beginning

Return Focus After Button Click
Here's an easy way to return the focus to a control after the user clicks on a button on the screen. If you use the keyboard, you can save time this way by not having to tab back to where you were. First, create a module variable of type Control, and call it mCtl. Next, in the GotFocus event of each input control, set mCtl equal to the control that got the focus. When you want to return the focus after the Click event, execute the SetFocus method of mCtl:

```
Private mCtl As Control

Private Sub Command1_Click()
   ' perform normal button routine here, then
   ' return focus to previous control.
   On Error Resume Next
   mCtl.SetFocus
End Sub

Private Sub Text1_GotFocus()
   Set mCtl = Text1
End Sub

Private Sub Text2_GotFocus()
   Set mCtl = Text2
End Sub
```

—Ed Ordorica, Toledo, Ohio

VB4 32, VB5, VB6
Level: Intermediate

Enhance Development With Registry Edits
Here are a couple useful Registry edits to enhance your day-to-day Visual Basic development. Save these Registry lines to a text file with a REG extension, then double-click on the file to merge it with your Registry.

Right-click on DLL register/unregister (make sure RegSvr32.exe is on your path):

```
REGEDIT4
[HKEY_CLASSES_ROOT\dllfile\Shell\Register\command]
@="RegSvr32 \"%1\""
[HKEY_CLASSES_ROOT\dllfile\Shell\Unregister\command]
@="RegSvr32 /u \"%1\""
```

Right-click on OCX register/unregister:

```
REGEDIT4
[HKEY_CLASSES_ROOT\.ocx]
@="ocxfile"
[HKEY_CLASSES_ROOT\ocxfile]
@="OCX"
[HKEY_CLASSES_ROOT\ocxfile\shell\Register\command]
@="RegSvr32 \"%1\""
[HKEY_CLASSES_ROOT\ocxfile\shell\Unregister\command]
@="RegSvr32 /u \"%1\""
```

Right-click on OLEView to open a DLL or OCX in the OLE Viewer utility (edit paths appropriately):

```
REGEDIT4
[HKEY_CLASSES_ROOT\dllfile\Shell\OleView\command]
@="c:\\Visual Studio\\Common\\Tools\\oleview.exe \"%1\""
[HKEY_CLASSES_ROOT\ocxfile\Shell\OleView\command]
@=":\\Visual Studio\\Common\\Tools\\oleview.exe \"%1\""
```

These shell extensions work for previous versions of VB as well; simply change the commands and/or paths appropriately. ***Editor's Note:*** *As with all Registry edits, be sure you either have a current backup or can live with the consequences.*

—Richard Hundhausen, Boise, Idaho

VB4 32, VB5, VB6
Level: Beginning

Catch Every TabStrip Click Event
You might have noticed that if you click on a TabStrip tab, then move the mouse pointer until it's no longer over the TabStrip without releasing the mouse button, the Click event doesn't fire even though the tab is changed. To detect this behavior, use this code:

```
Private Sub TabStrip1_MouseUp(Button As Integer, _
   Shift As Integer, X As Single, Y As Single)
   If X < 0 Or X > TabStrip1.Width Or _
      Y < 0 Or Y > TabStrip1.Height Then
         TabStrip1_Click
   End If
End Sub
```

This calls the Click event code if you do not release the mouse button above the TabStrip. You can't just put the Click handling code in the MouseUp event because then keyboard tab changes wouldn't work.

—Matt Hart, Tulsa, Oklahoma

VB4 32, VB5, VB6
Level: Intermediate

Display Taskbar Icons for Borderless Forms
When you create a borderless form in VB (BorderStyle = 0) and make it visible to the taskbar (ShowInTaskbar = True), only the form's caption shows and the form's icon is invisible. To display the icon, you must add a system menu to the form by modifying the style. This also gives you the ability to right-click on the taskbar button and see a standard system menu, but the items in it won't work unless you subclass the form and handle the system menu Click events yourself.

Add this code to the Declarations section of a form:

```
Private Declare Function GetWindowLong Lib _
   "user32" Alias "GetWindowLongA" (ByVal hWnd _
   As Long, ByVal nIndex As Long) As Long
Private Declare Function SetWindowLong Lib _
   "user32" Alias "SetWindowLongA" (ByVal hWnd _
   As Long, ByVal nIndex As Long, ByVal _
   dwNewLong As Long) As Long

Private Const GWL_STYLE = (-16)
Private Const WS_SYSMENU = &H80000
```

Add this code to the Form_Load event:

```
Dim lStyle As Long
lStyle = GetWindowLong(hWnd, GWL_STYLE) Or WS_SYSMENU
SetWindowLong hWnd, GWL_STYLE, lStyle
```

—Matt Hart, Tulsa, Oklahoma

VB4 16/32, VB5, VB6

Level: Intermediate

Link List Contents to ListIndex in Another List

I recently needed to create two listboxes, where the items displayed in the second listbox depend upon the item selected in the first listbox. After seeing the amount of code it took to do this using standard arrays, I came up with a better solution using array functions that reduced the amount of code by half. As an added benefit, using array functions makes it easy to add items to the listboxes—you don't need to modify any array dimensions, which makes the code much less error-prone. For example, to add the item "Candy Bars" to the first listbox, simply insert the subarray into the main array; nothing else is needed:

```
Array(3, "Candy Bars", "Milky Way", "Baby Ruth", _
   "Almond Joy")
```

The code uses one subarray for each item in the first listbox. The elements of each subarray are the number of items for the second listbox, followed by the item for the first listbox, followed by the items for the second listbox.

To test the code, start a Standard EXE project and put two listboxes on the form, keeping the default names of List1 and List2:

```
Option Explicit

Private varArray As Variant
Private varSubArray As Variant

Private Sub Form_Initialize()
   varArray = Array(Array(4, "Fruit", "Apples", _
      "Oranges", "Peaches", "Pears"), Array(5, _
      "Vegetables", "Peas", "Beans", "Corn", _
      "Beets", "Onions"), Array(3, _
      "Dairy Products", "Milk", "Cream", "Butter"))
End Sub

Private Sub Form_Load()
   Dim intIndex1 As Integer
   With List1
      For intIndex1 = 0 To UBound(varArray)
         varSubArray = varArray(intIndex1)
         .AddItem varSubArray(1)
      Next intIndex1
      .ListIndex = 0
   End With
End Sub

Private Sub List1_Click()
   Dim intIndex2 As Integer
   With List2
      varSubArray = varArray(List1.ListIndex)
      .Clear
      For intIndex2 = 0 To varSubArray(0) - 1
         .AddItem varSubArray(intIndex2 + 2)
      Next intIndex2
      .ListIndex = 0
      .Refresh
   End With
End Sub
```

This code works with VB5 and VB6, and should work with any version that supports the Array function.

—Paul Carlson, Fort Collins, Colorado

VB4 32, VB5, VB6

Level: Intermediate

Strip Comments off Strings Returned by GetPrivateProfileString

Windows doesn't treat comments in INI files the same way VB does in code. Typically, comments must be on a single line of their own. When calling the GetPrivateProfileString function, if the requested entry contains a Rem statement—such as "*; this is a rem'd statement*"—the entire entry, value and comment, will be returned.

If your code anticipates the entry without comment, confusion could result from a user entering a comment following the entry rather than on a separate line. Use this VB code example to properly receive a return using the GetPrivateProfileString function call under any circumstance:

```
Private Declare Function GetPrivateProfileString _
   Lib "kernel32" Alias "GetPrivateProfileStringA" (ByVal _
   lpApplicationName As Any, ByVal lpKeyName As _
   Any, ByVal lpDefault As String, ByVal _
   lpReturnedString As String, ByVal nSize As _
   Long, ByVal lpFileName As String) As Long

Private Sub Form_Click()
   Dim IniString As String
   Dim sDefault As String
   Dim lReturn As Long

   sDefault = "n/a"
   ' allocate sufficient buffer
   IniString = String$(260, 0)
   lReturn = GetPrivateProfileString("DB", _
      "Path", sDefault, IniString, _
      Len(IniString), "c:\test.ini")
   If lReturn > 0 Then
      IniString = Left$(IniString, lReturn)
      Debug.Print IniString

      ' added to strip out trailing comments
      If InStr(IniString, ";") > 0 Then
         IniString = Trim$(Left$(IniString, _
            InStr(IniString, ";") - 1))
         If InStr(IniString, vbTab) > 0 Then
            IniString = Trim$(Left$(IniString, _
               InStr(IniString, vbTab) - 1))
         End If
      End If
   Else
      IniString = sDefault
   End If
   Debug.Print IniString
End Sub
```

—R. Van Volkenburgh, Keller, Texas

VB4 32, VB5, VB6, VBA

Level: Beginning

Sharing Files Between VB and VBA

You can easily share files such as standard code, form, and class modules (BAS, FRM, and CLS files, respectively) between VB and VBA by using the appropriate menu commands. Within VB, select Project, then Add File… to import the desired file into a VB project. No action is necessary to export a VB file from a project because the files are maintained separately in Windows. Within VBA, select Import, then File… from the VB

Editor menu, or right-click on a folder in the Project Explorer and select Import File… from the popup menu. Because these files are stored inside the VBA application's file, such as in an Excel Workbook, you must export these files to a folder before another VB project can import them. To export a file from within VBA, right-click on the file in the Project Explorer and select Export File… from the popup menu. You will then see a prompt to select the folder where you want to put the file.

—John M. Dennis, South Lyon, Michigan

VB6

Level: Beginning

Change the Setup Wizard Background Color

VB6's Setup Wizard generates gradient blue background screens, as do wizards in previous versions of VB. However, an undocumented setting gives you the opportunity to see installation programs with other background colors. Open the SETUP.lst file generated by the Setup Wizard and add this line in the Setup section:

```
[Setup]
Color=N
```

N can be:
0: Black
1: Red background
2: Green
3: Yellow
4: Classical Blue
5: Magenta
6: Light blue
7: Gray
8: Red, but gradient reversed—top of screen is darker
and more ... .

—Pierre Metras, Mont Royal, Quebec, Canada

VB3 and up

Level: Beginning

Clear a Masked Edit Box

To blank out the text in a masked edit box, make sure the string you assign uses underscores matching the mask. It's difficult to maintain code when you have to change a mask; you have to find all the locations where you cleared the masked edit control in your code. To fix this problem, simply get rid of the mask, clear the contents, then restore the mask. Use this subroutine to clear the text of any masked edit box; just pass in the control:

```
Public Sub ClearMaskEdBox(oMaskEdBox As MaskEdBox)
   Dim sTemp As String
   With oMaskEdBox
      sTemp = .Mask
      .Mask = ""
      .Text = ""
      .Mask = sTemp
   End With
End Sub

'Sample Call:
Private Sub cmdClear_Click()
   ClearMaskEdBox MaskEdBox1
   MaskEdBox1.SetFocus
End Sub
```

—E.J. Osis, Sterling Heights, Michigan

VB4 16/32, VB5, VB6

Level: Intermediate

Use UDTs for Irregular Arrays

If you use variable-length strings and/or dynamic arrays in a user-defined type (UDT), the actual data does not become part of the structure. Instead, four-byte pointers are stored in the structure, and the actual data is stored separately. Therefore, it doesn't matter if the strings and/or arrays in variables A and B are different sizes; the type size is the same, assuming A and B are of the same UDT.

It also means that if you create an array with a UDT that itself includes dynamic arrays, any "inside" array inside one element of the "outside" array is physically distinct from the corresponding "inside" array of any other "outside" array element. Therefore, code along these lines is perfectly fine:

```
Type AnotherType
   Something As Integer
   SomethingElse As Long
End Type

Type SomeType
   Something As Integer
   InsideArr() As AnotherType
   SomethingElse As Long
End Type

Sub Test()
   '   set up outside array
   ReDim OutsideArr(1 To 4) As SomeType
   '   set up inside arrays
   ReDim OutsideArr(1).InsideArr(1 To 3)
   ReDim OutsideArr(2).InsideArr(1 To 7)
   ReDim OutsideArr(3).InsideArr(-4 To 0)
   ReDim OutsideArr(4).InsideArr(1 To 3, 2 To 14)
End Sub
```

You can use this code to create psuedo-multidimensional, nonrectangular arrays, or arrays in which more than one dimension is resized using ReDim Preserve. Both are impossible with ordinary multidimmed arrays.

—Robert Alan Gustafson II, Butler, Pennsylvania

VB3 and up

Level: Beginning

Start Up in Your Code Folder

I keep all my projects in a particular directory such as C:\work. When I save or open up a new project, I want the File dialog box to start at the directory C:\work. By default, it starts at the VB install directory, which is something like C:\Program Files\Microsoft Visual Studio\VB98. I don't save any projects in the same directory with the VB executable.

To get your project to open in the desired directory, change the "Start in" textbox in the shortcut for Microsoft Visual Basic to C:\work. It saves a mountain of clicks.

—Christian Gilstrap, Gilbert, Arizona

VB4 32, VB5, VB6

Level: Intermediate

Customize Colors and Fonts for Statusbar Panels

You can easily customize the fonts and colors in individual statusbar panels using a PictureBox control and an API call. Each statusbar panel can display a Picture object, so you can

use an invisible PictureBox control with the background, font, and foreground elements you want and assign that PictureBox image to the Picture object of a statusbar panel. The Panel object exposes a Width property, but not a Height property. The SendMessage API function can retrieve that height. Place a PictureBox on a form and set its Name to picStatus, set AutoRedraw to True, and set Visible to False. Change the Font object of the statusbar to a Panel's preferences before calling the PanelText procedure:

```
Private Type RECT
   Left As Long
   Top As Long
   Right As Long
   Bottom As Long
End Type

Private Declare Function SendMessage Lib "user32" Alias _
   "SendMessageA" (ByVal hWnd As Long, ByVal wMsg _
   As Long, ByVal wParam As Long, lParam As Any) As Long

Private Const WM_USER = &H400
Private Const SB_GETRECT = (WM_USER + 10)

Private Sub PanelText(sb As StatusBar, Index As Long, _
   aText As String, bkColor As Long, fgColor As Long)
   Dim R As RECT
   SendMessage sb.hWnd, SB_GETRECT, Index - 1, R
   With picPanel
      Set .Font = sb.Font
      .Move 0, 0, (R.Right - R.Left + 1) * _
         Screen.TwipsPerPixelX, (R.Bottom - _
         R.Top + 1) * Screen.TwipsPerPixelY
      .BackColor = bkColor
      .Cls
      .ForeColor = fgColor
      picPanel.Print aText
      sb.Panels(Index).Text = aText
      sb.Panels(Index).Picture = .Image
   End With
End Sub

Private Sub Form_Load()
   PanelText StatusBar1, 1, "Panel Message", _
      QBColor(1), QBColor(15)
End Sub
```

—Matt Hart, Tulsa, Oklahoma

VB4 32, VB5, VB6, VBA

Level: Beginning

Create Rich, Colorful Text

Call this sub to insert text of any color at the current insert point of a RichTextBox:

```
Private Sub ColorText(rtb As RichTextBox, Color _
   As Long, Text As String)
   Dim lR As Long, lG As Long, lB As Long

   lR = (Color Mod &H100)
   lG = (Color \ &H100) Mod &H100
   lB = (Color \ &H10000) Mod &H10000
   rtb.SelRTF = "{{\colortbl;\red" & CStr(lR) & "\green" & _
      CStr(lG) & "\blue" & CStr(lB) & ";}" & "{\cf1 " & Text & "}}"
End Sub
```

—Graeme Anderson, Blackburn, Australia

VB4 32, VB5, VB6

Level: Intermediate

Proper MouseLeave Detection

I have read many tips about using MouseMove events to create an Explorer-like coolbar look. The problem is that if your button is located close to the main form's border and you move the mouse too fast, the pointer jumps to the desktop or to another window without firing a Form_MouseMove event, and your button still has the "active" look.

The Command1_MouseMove event allows you to intercept the moment when the mouse pointer moves in or passes over the button, and you can easily assign an "active" picture to the button within this event. But you *cannot* intercept the moment when the pointer leaves the button using Form_MouseMove or any other Control_MouseMove or LostFocus event. To do this, you need to use two Windows APIs: SetCapture and ReleaseCapture. SetCapture directs all mouse events to the button, so you can trap the mouse at any point on the screen. ReleaseCapture releases mouse events, restoring standard behavior. The only thing you have to do with these APIs is check the pointer's coordinates to determine when it leaves the button:

```
Declare Function ReleaseCapture Lib "user32" () As Long
Declare Function SetCapture Lib "user32" (ByVal _
   hWnd As Long) As Long

Private Sub SSCommand1_MouseMove(Button As _
   Integer, Shift As Integer, X As Single, Y As Single)
   Dim ret As Long
   Static flagInside As Boolean

   If X < 0 Or X > SSCommand1.Width Or _
      Y < 0 Or Y > SSCommand1.Height Then
      ' pointer is out
      flagInside = False
      ret = ReleaseCapture()
      SSCommand1.Picture = _
         ImageList1.ListImages("gray").Picture
      SSCommand1.BevelWidth = 0
   Else
      ' pointer is in
      If flagInside = False Then
         flagInside = True
         ret = SetCapture(SSCommand1.hWnd)
         SSCommand1.Picture = ImageList1.ListImages( _
            "color").Picture
         SSCommand1.BevelWidth = 1
      End If
   End If
End Sub
```

As you can see, you don't need to use any other events except SSCommand1_MouseMove.

Use the flagInside variable to prevent the icon from blinking while the pointer passes over the button. While you drag the mouse over the button, the MouseMove event fires repeatedly. You don't need to assign the "active" picture every time if it has already been assigned.

—Alex Klikouchin, Toronto, Ontario, Canada

VB5, VB6
Level: Beginning

Make a Form Stay on Top Redux
Many routines use the SetWindowPos API to always keep a form on top. Most require the user to remember several nonintuitive arguments. I'll not only show you how to simplify the arguments, but I'll also illustrate the usefulness of the new Enum function. Enums have several advantages: Possible argument values are listed for you in the IDE using Microsoft's IntelliSense, the arguments are listed in the Object Browser, and Enums are included automatically in the type library when used in a class module. This all translates into easier programming and more code reuse:

```
'Paste this code into a module
Private Declare Function SetWindowPos Lib _
   "user32" (ByVal hWnd As Long, ByVal _
   hWndInsertAfter As Long, ByVal X As Long, _
   ByVal Y As Long, ByVal cx As Long, ByVal cy _
   As Long, ByVal wFlags As Long) As Long

Private Const SWP_NOMOVE = &H2
Private Const SWP_NOSIZE = &H1
Private Const SWP_SHOWWINDOW = &H40
Private Const SWP_NOACTIVATE = &H10

Public Enum WindowPos
   vbTopMost = -1&
   vbNotTopMost = -2&
End Enum

Public Sub SetFormPosition(hWnd As Integer, _
   Position As WindowPos)
   Const wFlags As Long = SWP_NOMOVE Or _
      SWP_NOSIZE Or SWP_SHOWWINDOW Or SWP_NOACTIVATE
   If Position = vbTopMost or Position = vbNotTopMost Then
      SetWindowPos hWnd, Position, 0, 0, 0, 0, wFlags
   End If
End Sub

'Add two command buttons to a form; then paste in this code
Private Sub Command1_Click()
   'Makes form topmost
   SetFormPosition Me.hWnd, vbTopMost
End Sub

Private Sub Command2_Click()
   'Restore form to normal position
   SetFormPosition Me.hWnd, vbNotTopMost
End Sub
```

—Jared A. Faulkner, Roxana, Illinois

VB6
Level: Advanced

Native Registry Access Fails Under MTS
VB provides an easy way to access the Registry with the built-in GetSetting and SaveSetting functions. These functions read and write information to the HKEY_CURRENT_USER\Software\VB and VBA Program area of the Registry. VB6 makes it easy to create DLLs that can run in Microsoft Transaction Server (MTS) on an NT Server. A problem occurs, however, when you create an ActiveX DLL that uses GetSetting or SaveSetting in a DLL running on an NT Server that has no current user.

In MTS, you can set up a user under the Identity tab of the Property settings for an MTS package. Selecting a user under the Identity tab is supposed to allow your DLL to run even if there is no currently logged-in user on the server. However, even with that user identified in MTS, your DLL still fails to run if it uses GetSetting or SaveSetting. Apparently, the VB DLL is unable to locate the HKEY_CURRENT_USER hive of the Registry if there is no current user. No error is returned to your calling app, while a variety of errors can be generated on the server in the event log, related to errors in MTS. The only workarounds seem to be either not using Registry settings (use hard-coded values or read a text file instead) or avoiding VB's native Registry functions and calling the API Registry functions directly to store and retrieve information in some other part of the Registry, such as HKEY_LOCAL_MACHINE.

Editor's Note: *A third option, for the masochists out there, might be to load a given user's hive programmatically. See the Microsoft Knowledge Base article Q168877 for details.*

—Edward Chicca, La Plata, Maryland

VB5, VB6
Level: Intermediate

Support Enumeration in Your Collection Classes
To create a collection class you can use with the For Each...Next syntax, add a subroutine that looks like this:

```
Private myCollection As Collection

Public Property Get NewEnum() As IUnknown
   Set NewEnum = myCollection.[_NewEnum]
End Property
```

Click on the Tools menu and select Procedure Attributes. Select the NewEnum procedure from the combo box. Click on the "Advanced >>" button. In the ProcedureID field, enter the value -4. Check the "Hide this member" checkbox. Now you can use the For Each...Next syntax with your collection class in the same way as a standard collection.

—Russell Jones, Conroe, Texas

VB4 16/32, VB5, VB6
Level: Intermediate

Speed Up String Operations
It's often faster to perform string operations with byte arrays than with 32-bit VB's native double-byte character strings. Even when using 16-bit VB4's single-byte character strings, it's still often faster to convert to byte arrays before intense processing. Compare the speed at which these two procedures execute and you'll be amazed. Paste this code into the default form of a new project to see the difference:

```
Private Sub Form_Click()
   Dim s As String
   Dim i As Integer
   s = String$(50000, "1")
   Debug.Print Time
   For i = 1 To 100
      Call countCharString(s, Asc("1"))
   Next
   Debug.Print Time
   For i = 1 To 100
      Call countCharByte(s, Asc("1"))
   Next
   Debug.Print Time
```

```
End Sub

Function countCharString(s As String, _
   charASCIIValue As Integer) As Long
   Dim i As Long
   For i = 1 To Len(s)
      If Asc(Mid$(s, i, 1)) = charASCIIValue Then
         countCharString = countCharString + 1
      End If
   Next
End Function

Function countCharByte(s As String, _
   charASCIIValue As Integer) As Long
   Dim b() As Byte
   Dim i As Long
   #If Win32 Then
      b = StrConv(s, vbFromUnicode)
   #Else
      b = s
   #End If

   For i = 0 To UBound(b)
      If b(i) = charASCIIValue Then
         countCharByte = countCharByte + 1
      End If
   Next
End Function
```

This optimization doesn't apply to all string operations. Depending on the nature of your problem, you'll see anything from fantastic improvement to slight degradation. Be sure to test both ways.

—Russell Jones, Conroe, Texas

VB4 32, VB5, VB6

Level: Intermediate

MCI Supports Multiple CD-ROMs

The Media Control Interface (MCI) can easily support multiple CD audio devices. You simply specify the drive letter in the MCI open command. To eject the CD from any drive, first place a listbox on a form. To detect which drives are CD-ROMs, place this code in the form's General Declarations section:

```
Private Declare Function GetDriveType Lib _
   "kernel32" Alias "GetDriveTypeA" (ByVal _
   nDrive As String) As Long
Private Declare Function mciSendString Lib _
   "winmm.dll" Alias "mciSendStringA" (ByVal _
   lpstrCommand As String, ByVal _
   lpstrReturnString As String, ByVal uReturnLength _
   As Long, ByVal hWndCallback As Long) As Long

Private Const DRIVE_CDROM = 5
```

The following code in the Form_Load event fills the listbox with available CD-ROM drives. Note that the code does not detect whether the CD-ROM drive can actually eject the CD. Use the "capabilities can eject" MCI command to determine that:

```
Private Sub Form_Load()
   Dim k As Long
   For k = Asc("A") To Asc("Z")
      If GetDriveType(Chr$(k) & ":") = DRIVE_CDROM Then
         List1.AddItem Chr$(k) & ":"
      End If
   Next
End Sub
```

Place this code in the List1_DblClick event to eject the CD:

```
Private Sub List1_DblClick()
   mciSendString "open " & List1.List(List1.ListIndex) & _
      " type cdaudio alias cdaudio", vbNullString, 0, 0
   mciSendString "set cdaudio door open", vbNullString, 0, 0
   mciSendString "close cdaudio", vbNullString, 0, 0
End Sub
```

—Matt Hart, Tulsa, Oklahoma

VB5, VB6

Level: Beginning

Enumerate a Dictionary Object

Although the Dictionary object does not have an enumerator, it does have an Items method that returns a Variant array. You can use the For...Each construct on the array:

```
Dim vItem As Variant
Dim dict As Scripting.Dictionary

Set dict = New Scripting.Dictionary
dict.Add "Item1", "Item data 1"
dict.Add "Item2", "Item data 2"
dict.Add "Item3", "Item data 3"

For Each vItem In dict.Items
   Debug.Print vItem
Next

Set dict = Nothing
```

Before you can use the Dictionary object in your application, you must set a Project Reference to the Microsoft Scripting Runtime (scrrun.dll).

—Steve Griffs, Niskayuna, New York

VB3 and up

Level: Beginning

Avoid IIf Inefficiencies

The IIf function—which returns one of two values determined by logical test—has this syntax: IIf(Expression, TruePart, FalsePart). At first, it might seem like an ideal shortcut for an If…Else...End If block. However, IIf is designed to execute both the True part and the False part. To verify, copy this into your Debug window and press enter:

```
? IIf(True, MsgBox("True Part"), MsgBox("False Part"))
```

Obviously, it's extremely inefficient—and possibly error-inducing—to place functions in the True and False parts of IIf, because they are both executed. In general, always use a standard If…Else…End If block instead.

—William Wen, New York, New York

VB5, VB6
Level: Beginning

Find the Last Modified Date of a Web Page
Microsoft's Internet Transfer Control (MSInet.ocx) is a great tool to use to automate the Web. However, how do you know you're looking at an updated version of a Web page? Use OpenURL to navigate to a page, then use the GetHeader function to retrieve the entire HTML header, from which you can get the last modified date:

```
Dim x As String
Inet1.OpenURL("www.devx.com")
x = Inet1.GetHeader
Debug.Print x
MsgBox Mid$(x, InStr(1, x, "Date:") + Len("Date:"), 30)
```

Or you can request the desired header: "Date" in this case:

```
Inet1.OpenURL("www.devx.com")
MsgBox Inet1.GetHeader("Date")
```

—William Wen, New York, New York

VB4 32, VB5, VB6
Level: Intermediate

Align Text on a Command Button
If you've ever wanted to align text on a command button and found you can do it only by using spaces in the caption, there are a couple of constants that can help you do this with the SetWindowLong API Call. Add this code to a standard BAS module, then call it at will, passing the command button and the desired combination of vertical and horizontal alignment values:

```
Option Explicit

Private Declare Function GetWindowLong Lib _
   "user32" Alias "GetWindowLongA" (ByVal hWnd _
   As Long, ByVal nIndex As Long) As Long
Private Declare Function SetWindowLong Lib _
   "user32" Alias "SetWindowLongA" (ByVal hWnd _
   As Long, ByVal nIndex As Long, ByVal _
   dwNewLong As Long) As Long

Private Const BS_LEFT As Long = &H100
Private Const BS_RIGHT As Long = &H200
Private Const BS_CENTER As Long = &H300
Private Const BS_TOP As Long = &H400
Private Const BS_BOTTOM As Long = &H800
Private Const BS_VCENTER As Long = &HC00

Private Const BS_ALLSTYLES = BS_LEFT Or BS_RIGHT _
   Or BS_CENTER Or BS_TOP Or BS_BOTTOM Or BS_VCENTER

Private Const GWL_STYLE& = (-16)

Public Enum bsHorizontalAlignments
   bsLeft = BS_LEFT
   bsRight = BS_RIGHT
   bsCenter = BS_CENTER
End Enum

Public Enum bsVerticalAlignments
   bsTop = BS_TOP
   bsBottom = BS_BOTTOM
   bsVCenter = BS_VCENTER
End Enum

Public Sub AlignButtonText(cmd As CommandButton, _
   Optional ByVal HStyle As bsHorizontalAlignments = bsCenter, _
   Optional ByVal VStyle As bsVerticalAlignments = bsVCenter)

   Dim oldStyle As Long

   ' get current style
   oldStyle = GetWindowLong(cmd.hWnd, GWL_STYLE)

   ' clear existing alignment setting(s)
   oldStyle = oldStyle And (Not BS_ALLSTYLES)

   ' set new style and refresh button
   Call SetWindowLong(cmd.hWnd, GWL_STYLE, _
      oldStyle Or HStyle Or VStyle)
   cmd.Refresh
End Sub
```

—Sam Huggill, Colchester, England

VB5, VB6
Level: Beginning

Retrieve Localization Strings
Use this API function wrapper to retrieve localization and personalization information:

```
Private Declare Function GetLocaleInfo Lib _
   "kernel32" Alias "GetLocaleInfoA" (ByVal _
   Locale As Long, ByVal LCType As Long, ByVal _
   lpLCData As String, ByVal cchData As Long) As Long

Public Function WinLocaleInfo(ByVal lnfoType As _
   Long) As String
   Dim sLCData As String
   Dim nRet As Long

   nRet = GetLocaleInfo(0, lnfoType, sLCData, 0)
   If nRet Then
      sLCData = Space$(nRet)
      nRet = GetLocaleInfo(0, lnfoType, _
         sLCData, Len(sLCData))
      If nRet Then
         WinLocaleInfo = Left$(sLCData, nRet)
      End If
   End If
End Function
```

Here are some of the handier parameters you can use for information:

```
   LOCALE_SCURRENCY = &H14
   ' local monetary symbol
   LOCALE_SDATE = &H1D
   ' date separator
   LOCALE_SDAYNAME1 = &H2A
   ' long name for Monday
   LOCALE_SDECIMAL = &HE
   ' decimal separator
```

—Brian Morris, Woonsocket, Rhode Island

VB5, VB6
Level: Beginning

Alpha Resource IDs Can Spell Trouble
Resource files are a great way to store text strings and images, but beware—use of a resource file can override the property setting for your application's icon. When Windows needs an icon to represent your application, it always grabs the first one available.

You can use the VB Resource Editor add-in to store icons. Normally, VB assigns an ID of "1" to the icon you assign as the application default. However, if you assign alphanumeric IDs in the Resource Editor, instead of using only numerics, the icon used for your application will be the first one listed under Icons. I always preface the desired application icon's ID with the string "AAA" to ensure that it remains first alphabetically.

—Al Meadows, Oklahoma City, Oklahoma

VB4 16/32, VB5, VB6
Level: Beginning

Ask the Form Itself Whether it's Loaded
Occasionally you initialize a form but don't load it. You might do this to read in initial application Registry values. After that, any references to control properties on the form automatically cause your Form_Load event to fire, whether or not you intended to fire the Form_Load event. If you want to know programmatically whether your form has been loaded, implement these steps.

Add a private Boolean variable to your form and call it m_bLoaded:

```
Option Explicit

Dim m_bLoaded as Boolean
```

Add Property Let and Get procedures to provide public read and write access to your form's internal m_bLoaded variable:

```
Public Property Get Loaded() As Boolean
   Loaded = m_bLoaded
End Property

Public Property Get Let(ByVal bLoaded As Boolean)
   m_bLoaded = bLoaded
End Property
```

Add this line of code to your form's Load event:

```
Me.Loaded = True
```

To see whether your form is loaded from other modules or other forms, use this code:

```
If  FormName.Loaded = True Then
   ' Do stuff that can only be done if the form
   ' has been loaded (i.e., set control properties)
EndIf
```

Using this code helps prevent you from accidentally firing the Form_Load event.

—Scott McFadden, Oklahoma City, Oklahoma

VB4 32, VB5, VB6
Level: Intermediate

Locate the Temp Folder
In your apps, do the right thing: Use the computer's Temp folder to hold your temporary files. To find that location, paste this code into the Declarations section of a form:

```
Private Declare Function GetTempPath Lib _
   "kernel32" Alias "GetTempPathA" (ByVal _
   nBufferLength As Long, ByVal lpBuffer As _
   String) As Long
Private Const MAX_PATH = 260

Public Property Get TempPathLocation() As String
   Dim sBuffer As String
   sBuffer = Space(MAX_PATH)
   If GetTempPath(MAX_PATH, sBuffer) <> 0 Then
      TempPathLocation = Left$(sBuffer, _
         InStr(sBuffer, vbNullChar) - 1)
   Else
      TempPathLocation = ""
   End If
End Property
```

This routine returns the computer's established Temp folder. If none has been established, an empty string is returned, which may be interpreted as the current directory. Be sure to use Kill on all created temp files when you're through with them.

—Robert Smith, Kirkland, Washington

VB4 32, VB5, VB6
Level: Intermediate

Make Sure MsgBox is on Top
When you implement the Always on Top feature with VB forms using SetWindowPos and HWND_TOPMOST, message boxes are shown below the topmost form. To overcome this, you can use the vbSystemModal constant in the MsgBox function's Style attribute. This forces the message box to show on top of the Always on Top form. This is necessary only with the 32-bit versions of VB.

—Srinivasa S. Sivakumar, Chicago, Illinois

VBA
Level: Beginning

Catalog Your Graphics
I needed a printed catalog of GIF images in a directory. So I wrote a routine that creates a new document, and inserts an image for each GIF file in the directory, along with its name and size. When the routine has finished, you can print or save the document (for the complete code listing, download the RTF file for this supplement under the Code section on *VBPJ*'s Web site, www.vbpj.com). This code does not check whether the image fits on the page; in my case all the images were small, such as Web buttons and rules.

—Richard Gardiner, Chicago, Illinois

VB3 and up
Level: Beginning

Perform Complex Tests in the Immediate Window

If you've ever tried to type a small program into the Immediate window or the Debug window in Access, you might have found that the style of VB programming you use in most of your code didn't work there. For example, this code doesn't work when you type it into the Debug window because one of the statements takes more than a single line:

```
Dim rs As Recordset
Set rs = CurrentDb.OpenRecordset("SELECT * FROM MyTable")
If rs.RecordCount = 0 Then
   Beep
   Debug.Print "Hello"
ElseIf rs.Updatable Then
   DoThis()
   DoThat()
End If
```

You should almost always use Option Explicit when writing regular code, but it is neither necessary nor available in the Immediate window. You never need to declare a variable, as in the first line of the previous snippet. When you refer to a variable name for the first time, it is implicitly created. Remember that Variants of type Object are late-bound, which means you don't see the AutoSyntax box for implicitly created object variables.

Using the colon executes more than one statement on a single line:

```
Debug.Print "A" : Beep
```

For example, this code prints an "A" in the Immediate window and then beeps. By itself, this isn't highly valuable, but using the colon construct in the right place allows you to reformat most VB statements for the Immediate window.

To handle the multiple-line If...ElseIf...Else...End If construct in the Immediate window, use a single-line If...Else...Then construct—nested, if necessary—with the colon construct. The syntax single-line version of the If statement is simple:

```
If <condition> Then <statement> [Else <statement>]
```

Notice there is no End If or ElseIf clause, and the Else clause is optional. Here is the previous snippet, correctly formatted for the Immediate window with ElseIf replaced by a nested If:

```
Set rs = CurrentDb.OpenRecordset("SELECT * FROM MyTable")
If rs.RecordCount = 0 Then Beep : Debug.Print _
   "Hello" Else If rs.Updatable Then DoThis() : DoThat()
```

Editor's Note: *Line continuation characters also aren't allowed in the Immediate window, although they're required in this example to fit the supplement's layout.*

Be careful when converting If statements from multiple-line to single-line. Unusual or complex nesting might require special logic.

The colon also allows you to perform loops in the Immediate window. VB declares the looping variable implicitly:

```
   For Each v In MyCollection : ? v.Description : Next v
```

—Chase Saunders, Bath, Maine

VB3 and up
Level: Beginning

Perform Faster String Manipulations

Are you dealing with strings you have to parse if you want to drop one special character or change it into another? Keep this trick in mind. Even though this code seems to work fine, there is one minor problem that can cause headaches. The time you spend in memory allocation (line 7) increases dramatically in relation to the length of the processed string:

```
Dim m_newtext, m_oldtext As String
Dim i As Integer, m_c As String
m_newtext = ""
For i = 1 To Len(m_oldtext)
   m_c = Mid(m_oldtext, i, 1)
   If ParseTestFunction(m_c) Then
      m_newtext = m_newtext & m_c
   End If
Next i
```

Use this code instead:

```
Dim m_newtext, m_oldtext As String
Dim i As Integer, m_c As String
Dim j As Integer
m_newtext = m_oldtext
j = 1
For i = 1 To Len(m_oldtext)
   m_c = Mid(m_oldtext, i, 1)
   If ParseTestFunction(m_c) Then
      Mid(m_newtext, j, 1) = m_c
      j = j + 1
   End If
Next I
If j > 1 Then
   m_newtext = Left(m_newtext, j - 1)
Else
   m_newtext = ""
End If
```

You can use the same technique when you parse for substrings. The improvement is obvious for large strings. Implement this with a 30K string and see the difference. It looks like a lot more code, but inserting a character into an existing string is always much faster than appending it.

—Sorinel Ticrea, Minneapolis, Minnesota

VB3 and up
Level: Beginning

Take Care When You Declare

This is a common error among C programmers who have recently switched to VB. This code would result in A and B as Variant variables and only C as an Integer variable:

```
Dim a, b, c As Integer
```

Here's the proper way to declare these variables:

```
Dim a As Integer, b As Integer, c As Integer
```

Or:

```
Dim a%, b%, c%
```

—Philip Lee, Foster City, California

VB4 16/32, VB5, VB6

Level: Intermediate

Generate Business Object Classes

I often include a module called Utilities.bas in my projects. It contains routines that are not directly used by the application, but instead are invoked in the Debug window, usually to generate code snippets for the project.

For example, I have a routine that generates the skeleton of a class whose properties are mapped to the fields of a table, view, or stored procedure—as in the Business Object class described in Deborah Kurata's book *Doing Objects in Microsoft Visual Basic 5.0* (pp. 476-479, Que, 1999, ISBN: 1562765779). Writing the code for this class is tedious, especially when the table has many fields. Imagine writing all the Get and Let property subs for a table with 20 or 30 fields (for the complete code listing, download the RTF file for this supplement under the Code section on *VBPJ*'s Web site, www.vbpj.com).

If I want to write a class for the Person table, for example, I go to the Debug window and type:

```
GenTableClass "Person"
```

I then cut and paste the generated code to my target class module.

—Arnel J. Domingo, Hong Kong

VB3 and up

Level: Intermediate

☆☆☆☆☆ Five Star Tip

Add a Document Name to Your Printouts

I wanted to distinguish between different print jobs. I tried to find a way to set the print job name that appears in the printer queue, but even some nasty API calls didn't work well. I found a simple trick, though it works only after compiling to an EXE:

```
Dim sAppTitle As String

sAppTitle = App.Title
App.Title = "What ever Print Job Name you wish"

' Start Printing....

' End Printing
Printer.EndDoc
App.Title = sAppTitle
```

—Yoram Shechori, Netanya, Israel

VB3 and up

Level: Beginning

Set Tab Order in a Hurry

Setting up the control tab order can be unruly—a sore point with many VB developers. In fact, in Jeff Hadfield's recent editorial, "Five Things You Hate About VB6" [Editor's Note, *VBPJ* April 1999], it's listed as one of the top five things people hate most about VB.

I've never understood this; it's never been a big deal to me. But many people are shocked when I show them how I lay out tab orders, and are even embarrassed they didn't think of it themselves. I don't worry about the tab order until I've placed all the controls on the form, then I order them.

Now for the slam in the face: Think backwards. While viewing your form, click on the control you want to be the *last* control in the tab order. Highlight the TabIndex property in the Properties window and change it to zero. Now click on the control that you want to have the focus before that one, and repeat the process. Repeat this process until you reach the control that should have focus when the form opens.

Here's why this works: VB does most of the work for you. VB automatically reassigns the tab orders if you change the TabIndex property of a control. Because you started by setting the tab index of the last control first, VB increments the last control's tab index as you set the rest of the controls. It's a heck of a lot easier to set the tab index to zero while you're going through the order. I can rip through a form with more than 50 controls in about two minutes using this technique.

—James Bragg, Tulsa, Oklahoma

VB4 16/32, VB5, VB6

Level: Beginning

Supply Users with More Icon Choices

In the past, VB allowed an app to have only one icon. However, resource files expose any contained icons to Windows. Simply add the icons you wish to show to a RES file and add the RES file to your project. Then compile the app, make a shortcut to it, and open the Shortcut tab under the Properties window with a right click and select Properties. Press the Change Icon button and voila!—all the icons are there to choose from.

—Michael Lewis, Chaing Mai, Thailand

VB5, VB6

Level: Beginning

Create a Password-Protected Database

In the Winter 1998/1999 edition of *Getting Started with Visual Basic*, the Ask the VB Pro column contains a tip from Phil Weber with the title "Protect Data from Prying Eyes." This tip says you can use Access 7.0 to protect your database files with a password and then use this password in your applications to open the database file.

The tip works perfectly. But what if you don't have Access 7.0, or you need the application to create a protected database? Here's a tip that does the job. When you create your database use this code, and pay attention to the connect parameter:

```
Dim wsAccount As Workspace
Dim dbAccount As Database

Set wsAccount = DBEngine.Workspaces(0)
Set dbAccount = _
   wsAccount.CreateDatabase("mydatabase.mdb", _
   dbLangGeneral & ";pwd=TOP SECRET", dbVersion30)
```

You now have a database with a password. You can replace the words "TOP SECRET" with any you choose, depending on the level of encryption you desire to use. To open the file:

```
sPassword = "TOP SECRET"
Set db=OpenDatabase("mydatabase.mdb", False, _
   False, ";pwd=" & sPassword)
```

—José Arturo Ramirez Guzman, Coatzacoalcos, Veracruz, Mexico

VB5, VB6

Level: Intermediate

Detect Unused Objects

Improper use of objects and object references can cause your application's objects to stay resident in memory. Such "memory leaks" often causes system performance degradation and prevents your application from scaling well over time. Instead of using complex system utilities and third-party tools to detect such "bad code" in your application, you can put VB5/VB6's event technology to work for you.

Create an ActiveX DLL—such as ListObjects.dll—with a single class: ResidentObjs.cls. Add a method to raise an event. You may call this subroutine DetectAllObjects. Declare an event in the Declarations section of the class:

```
Option Explicit
Public Event ObjectNotification()

Public Sub DetectAllObjects()
   DoEvents ' let the system do its job.
   RaiseEvent ObjectNotification()
End sub
```

In your application, add a reference to ListObjects.dll to your project. Then declare a global variable in a BAS module as a ResidentObjs type, and instantiate this object variable at Application startup:

```
Sub Main()
   Set oListObject = New ListObject.ResidentObjs
End Sub
```

In each class of your project, you must declare a ResidentObjs variable using WithEvents. Set this object variable to the global variable upon class initialize or some other class function that is called each time you create an object of that class. In the event function, you can add any code that warns you about the current object instance. For example, you can add Debug.Print, or some code that writes the class name to a file.

At any suitable point in your project code, you can call DetectAllObjects(). The number of times the event function gets called is the number of objects resident in your memory at that time. To zero in on only some of the main classes, you can limit the DLL object declaration to only those classes.

How does this work? Events are a form of anonymous broadcast messages. They are sent to every object instance in your application. Hence, this simple yet powerful mechanism can detect invalid objects. In fact, you can step into the code to actually see the event function being called several times.

—Rahul Pandhe, Hayward, California

VB3 and up

Level: Intermediate

Translate Color Values Faster

This tip is an update to "Translate Color Values," which appeared in the last edition of this supplement ["101 Tech Tips for VB Developers," Supplement to *VBPJ*, February 1999]. That method converted numbers to strings, then performed string manipulation to get individual RGB values. This solution uses a concept near and dear to C programmers—unions—and is an order of magnitude faster.

With the RGB function, VB provides a neat and valuable tool for converting separate Red, Green, and Blue values into a single Long color value. However, VB doesn't provide any way to convert this color value back to its constituent RGB values. Enter the much-feared LSet command, which copies the storage of one user-defined type (UDT) onto another.

Put this code in a module:

```
Public Type RGB_TYPE
   R As Byte
   G As Byte
   B As Byte
   Filler As Byte
End Type

Private Type RGB_FULL_TYPE
   lngRGB As Long
End Type

Public Function ToRGB(ByVal vlngColor As Long) As RGB_TYPE
   Dim udtRGBFull As RGB_FULL_TYPE
   udtRGBFull.lngRGB = vlngColor
   ' Poor man's C Union
   LSet ToRGB = udtRGBFull
End Function
```

To use this function, put a picture in a form's Picture property, and insert this code:

```
Private Sub Form_MouseUp(Button As Integer, _
   Shift As Integer, X As Single, Y As Single)
   Dim udtRGB As RGB_TYPE
   udtRGB = ToRGB(Point(X, Y))
   With udtRGB
      Me.Caption = "R=" & .R & " G=" & .G & " B=" & .B
   End With
End Sub
```

Click on different places on the picture. VB3 users must return the values differently, because before VB4 VB didn't support the return of a UDT.

—Richard Lindauer, Stroudsburg, Pennsylvania

VB4 32, VB5, VB6

Level: Intermediate

Generic ListBox Columns Routine

Use the SendMessage API to set tab stops in a VB listbox by creating these declarations and this routine in a module:

```
Private Declare Function SendMessage Lib _
   "user32" Alias "SendMessageA"(ByVal hWnd As _
   Long, ByVal wMsg As Long, ByVal wParam As _
   Long, lParam As Any) As Long
Private Const LB_SETTABSTOPS = &H192

Public Sub SetListTabStops(ListHandle As Long, _
   ParamArray ParmList() As Variant)
   Dim i As Long
   Dim ListTabs() As Long
   Dim NumColumns As Long

   ReDim ListTabs(UBound(ParmList))
   For i = 0 To UBound(ParmList)
      ListTabs(i) = ParmList(i)
   Next i
   NumColumns = UBound(ParmList) + 1

   Call SendMessage(ListHandle, LB_SETTABSTOPS, _
```

```
      NumColumns, ListTabs(0))
End Sub
```

Call the routine in Form_Load to set the tab stops where MyListBox is the listbox and the tab stop will be around the 12th character. Generally speaking, TabStop divided by four equals roughly the number of characters per column:

```
Call SetListTabStops(lstMyListBox.hWnd, 48)
```

If more columns are needed, simply add them to the function call:

```
Call SetListTabStops(lstMyListBox.hWnd, 48, 74, 100)
```

Add items to the listbox using vbTab to separate columns:

```
lstMyListBox.AddItem "Column1" & vbTab & "Column2"
```

—Jim Farrell, Lincolnshire, Illinois

VB4 32, VB5, VB6

Level: Advanced

Set Error Level in DOS

Use this code to set the error level in DOS, or as I use it, to return a calculated date:

```
Private Declare Sub ExitProcess Lib "kernel32" ( _
   ByVal uExitCode As Long)

Sub Main()
   Dim lngDate As Long
   Dim intCmd As Integer
   intCmd = 0

' Check CommandLine for parameters
   If Command <> "" Then
      intCmd = Val(Command)
   End If

' Format date and convert to Long
   lngDate = CLng(Format$(Now + intCmd, "yymmdd"))
   ExitProcess lngDate
End Sub
```

Compile the program and save it as VB_Day.exe. For example, make a DOS batch file, and test the program:

```
@echo off
rem *** Calulate date from VB_Day
VB_Day -1
SET MyDay=%ERRORLEVEL%
echo Today - 1 = %MyDay%

VB_Day 7
SET MyDay=%ERRORLEVEL%
echo Today + 7 = %MyDay%

VB_Day
SET MyDay=%ERRORLEVEL%
echo Today is  = %MyDay%
```

Do not call ExitProcess from anywhere but the end of Sub Main, and only after releasing all object references. ExitProcess has the same practical effect on your application as VB's End statement, with the added "bonus" of also shutting down the IDE if you're not running a compiled EXE.

—Peter André, Malmö, Sweden

VB4 16/32, VB5 VB6

Level: Beginning

Use IsLoaded to Check for Forms

The IsLoaded function tests to determine if any instances of a form are already loaded or exist in memory:

```
If IsLoaded("frmMyForm") Then
   Debug.Print "frmMyForm is loaded"
End If

Public Function IsLoaded(sForm As String)
   Dim Frm As Form
   For Each Frm In Forms
      If Frm.Name = sForm Then
         IsLoaded = True
         Exit For
      End If
   Next
End Function
```

—Geir Arnesen, Oslo, Norway

VB4 32, VB5, VB6

Level: Beginning

Assign Toolbar Button Images at Run Time

When developing applications, quite often you end up experimenting with different toolbar icons to test how well they serve their purpose. If you change an icon in an ImageList, you have to de-assign this ImageList from your toolbar and you lose all your image assignments. Instead of doing this repeatedly, use the same key values in your buttons and images.

Use this code to assign images to your buttons. For me it has been a time-saver:

```
Dim myButton As Button

On Error Resume Next

With Toolbar1
   .ImageList = ImageList1
   For Each myButton In .Buttons
      myButton.Image = myButton.Key
   Next
End With
```

—Lasse Rantanen, Kantvik, Finland

VB5, VB6

Level: Beginning

Access Your Help and INI Files Within VB

In small projects I am involved with, I frequently access project-related INI files and help files. At times I work with identical databases on Access and SQL Server, so I need to keep changing the data-source name (DSN) entry in the INI file. Or when I update the help file due to some change in the program, I simply include these files (INI or RTF) as related documents in the project. To do this, use the Project|Add File menu option and select the Add As Related Document checkbox in the file-open dialog. VB automatically opens the associated application (Notepad/Word) when you double-click on these files.

—Ravindra Okade, Plainsboro, New Jersey

VB4 32, VB5, VB6
Level: Intermediate

ActiveX Can't Create *Which* Object?
When working on a large VB application that uses hundreds of COM objects, the "429 can't create object" error doesn't give you much help in determining which object could not be created. You can get around this limitation by writing a function to wrap the VB runtime CreateObject function:

```
Public Function CreateObject(sProgID as string) As Object
   On Error Goto CreateErr

   ' Call the VB runtime CreateObject function
   Set CreateObject = VBA.CreateObject(sProgId)

   Exit Function
CreateErr:
   ' return the error with the name of the object
   ' that could not be created
   Err.Raise Err.Number, _
      "CreateObject Wrapper", Err.Description & _
      ": '" & sProgID & "'"
End Function
```

With this wrapper function, you get the 429 error and the name of the object that could not be created.

—Kelley L. Larsen, Monroe, Washington

VB4 32, VB5, VB6
Level: Intermediate

Leverage Office to Spellcheck RichText
Integrate Microsoft Word 97's spellchecking capability into VB apps while maintaining formatting within a rich textbox. To test this code:

1. Create a standard EXE project in VB.
2. Add the RichTextBox control from the Components menu.
3. Add a reference to the Microsoft Word 8.0 Object Library.
4. Drop a RichTextBox and a CommandButton onto the form.
5. Rename the RichTextBox to rtfText.
6. Change the caption of the CommandButton to Spell Check.
7. In the Click event of the CommandButton, add the next code listing.
8. Save and run the project.
9. Type some text in the RTF box and click on the CommandButton to check the spelling.

```
On Error GoTo SpellCheckErr
   Dim oWord As Object

   Set oWord = CreateObject("Word.Application")

   'Save the RTF Box contents to a temporary file
   rtfText.SaveFile "C:\TEST.RTF", rtfRTF

   'Open the saved document and spellcheck it
   oWord.Documents.Open ("C:\TEST.RTF")
   oWord.ActiveDocument.SpellingChecked = False
   oWord.Options.IgnoreUppercase = False
   oWord.ActiveDocument.CheckSpelling

   'Save the changes to the RTF file & close
   oWord.ActiveDocument.Save
   oWord.ActiveDocument.Close
   oWord.Quit

   'Load the changes back to the rtf text box.
   rtfText.LoadFile "C:\TEST.RTF", rtfRTF

   Set oWord = Nothing
   Screen.MousePointer = vbDefault
   MsgBox "Spell Check is complete", _
      vbInformation, "Spell Check"
   Exit Sub

SpellCheckErr:
   MsgBox Err.Description, vbCritical, "Spell Check"
   Set oWord = Nothing
```

—Rohit Kapoor, Gaithersburg, Maryland

VB5, VB6
Level: Beginning

Insert Persistent Breakpoints
VB won't let you save a breakpoint when you have a lengthy debugging session. Use Debug.Assert to create persistent breakpoints that trigger when you are in the environment, but not in the compiled program. Just insert "Debug.Assert False" where you want the breakpoint.

This technique is also useful to ensure different branches of the code are debugged. When coding, insert these persistent breakpoints in every section of the code you know needs to be tested by stepping through it, and delete the breakpoints once you have verified that the section works as intended.

—Tore Bostrup, Columbia, South Carolina

VB3 and up
Level: Beginning

Count Substrings
This little routine demonstrates how easily you can determine the number of substrings within a string, given any specified separator character(s). Pass the string to be parsed, and the separator, which might be multiple characters long, and DCount returns the number of substrings:

```
Public Function DCount(ByVal vData As String, _
   SP As String) As Integer
   Dim x As Integer
   Dim n As Integer
   If vData = "" Or SP = "" Then Exit Function
   vData = Trim(vData)
   n = 1
   DCount = 1
   Do
      x = InStr(n, vData, SP, vbTextCompare)
      If x > 1 And x < (Len(vData) - Len(SP)) Then
         DCount = DCount + 1
      End If
      n = x + Len(SP)
   Loop Until x = 0
End Function

s = "GTL-00030/22*M121222*C001"
cnt = DCount(s ,"*")       ->       cnt=3
cnt = DCount(s ,"/")       ->       cnt=2
cnt = DCount(s ,"0")       ->       cnt=7
```

—Tan Shing Ho, Kuala Lampur, West Malaysia

VB4 32, VB5, VB6
Level: Intermediate

Determine Visible Part of a Window
Programmers often need to know whether only a part of a window is visible. This can require a difficult calculation with coordinates. Use this routine to easily determine the visible part of your window or any control that has an hWnd property:

```
Option Explicit

Public Type RECT           ' Declare API type
   Left As Long
   Top As Long
   Right As Long
   Bottom As Long
End Type

' Declare API functions:
Private Declare Function InvalidateRect Lib _
   "user32" (ByVal hWnd As Long, lpRect As _
   RECT, ByVal bErase As Long) As Long
Private Declare Function GetUpdateRect Lib _
   "user32" (ByVal hWnd As Long, lpRect As _
   RECT, ByVal bErase As Long) As Long
Private Declare Function GetClientRect Lib _
   "user32" (ByVal hWnd As Long, lpRect As _
   RECT) As Long

Public Function GetVisibleRect(ByVal hWnd As _
   Long, lpRect As RECT) As Boolean
   Dim lpClientRect As RECT

   Call GetClientRect(hWnd, lpClientRect)
   Call InvalidateRect(hWnd, lpClientRect, False)
   GetVisibleRect = GetUpdateRect(hWnd, lpRect, False)
End Function
```

—Meszaros Akos, Szalonkau, Hungary

VB5, VB6
Level: Intermediate

Construct Shortcut to an Interface
When using a class that implements an interface, it can be frustrating to have to create two variables to reference all the properties of both the interface class and the implemented class. For example, if the CEmployee class implements the IPerson interface, then you need two variables:

```
   Dim Emp1 As CEmployee
   Dim Person1 As IPerson

   Set Emp1 = New CEmployee
   Set Person1 = Emp1

   Person1.Name = "Joe Smith"
   Emp1.HireDate = "1/1/1998"
```

Instead, create a method in the CEmployee class that returns itself as the interface object:

```
   Public Function AsIPerson() As IPerson
      Set AsIPerson = Me
   End Function
```

Now you can rewrite the preceding code using just one object variable:

```
   Dim Emp1 As CEmployee

   Set Emp1 = New CEmployee

   Emp1.AsIPerson.Name = "Joe Smith"
   Emp1.HireDate = "1/1/1998"
```

—Matthew Janofsky, Lawrenceville, Georgia

VB4 32, VB5, VB6
Level: Beginning

Toggle Toolbar Captions Using Tag
Many applications have an option to show the toolbars as text and image or text-only. I have seen programmers handle this with two toolbars and show them based on the user's selection. This code uses only one toolbar. The only requirement is to store the desired Caption in each button's Tag property. This technique works only if you are not changing the toolbar button's Tag at run time:

```
' tbmain is the only toolbar. At design time, the
' caption of buttons are left blank.
Sub HideShowToolbarText(bShowText As Boolean)
   Dim i As Integer

   For i = 1 To tbMain.Buttons.Count
      tbMain.Buttons(i).Caption = IIf( _
         bShowText, tbMain.Buttons(i).Tag, "")
   Next i
End Sub

Private Sub mnuToolbarImageText_Click()
   mnuToolbarImageText.Checked = Not _
      mnuToolbarImageText.Checked
   HideShowToolbarText (mnuToolbarImageText.Checked)
End Sub
```

—Kanthi Chonachalarn, Lake In The Hills, Illinois

VB5, VB6
Level: Advanced

Use Pointers to Parents
When trying to navigate through a complex class hierarchy from any given instance of an object, it's useful to be able to reference its parent. But how do you clean up any circular references when terminating the objects? Here is a simple way to retrieve a parent object using the undocumented ObjPtr function in VB:

```
' Code for clsChild Class
Option Explicit

Private Declare Sub CopyMemory Lib "kernel32" _
   Alias "RtlMoveMemory" (dest As Any, source _
   As Any, ByVal numBytes As Long)

Private lngParent As Long

Property Set Parent(Frm As Form)
   ' obtain a pointer to the object
   lngParent = ObjPtr(Frm)
End Property

Property Get Parent() As Form
```

```
   Dim frmobj As Form
   CopyMemory frmobj, lngParent, 4
   Set Parent = frmobj
   CopyMemory frmobj, 0&, 4
End Property
```

To test the object's new Parent property, create a new form and add a button called cmdTest. Place this code in the button's Click event:

```
Option Explicit

Private Sub cmdTest_Click()
   Dim loChild As New clsChild
   With loChild
      Set .Parent = Me
      MsgBox  .Parent.Caption
   End With
End Sub
```

Because the child class only contains the pointer to the parent's memory, there are no circular references to resolve.

—Jeff Turner, Loami, Illinois

VB3 and up

Level: Intermediate

Debug Print in an EXE

Contrary to common belief, Debug.Print statements are *not always* removed from an executable. This behavior can be demonstrated easily. Open a new project, place a single CommandButton on the default form, and add this code:

```
Public Function DebugTime()
   MsgBox "Caught!"
End Function

Private Sub Command1_Click()
   Debug.Print DebugTime
End Sub
```

Compile the program to an EXE, run, and click on the button. Unexpectedly, the message box appears. This is obviously a contrived example, but it's easy to imagine cases where Debug.Print is used to print the return value of a function, for example. If variables are passed as ByRef parameters, and if the function alters the value of those variables, this error is propagated through to the executable, and might be extremely difficult to find. The morals of this tip are:

- Pass parameters ByVal unless you're certain you're not going to change them, or you're using them as "output" parameters.
- Be careful when using Debug.Print. It might be doing more than you think.

—John Cullen, Pedroucos, Portugal

VBA

Level: Beginning

Streamline Math in VBA

It's possible to streamline some calculation procedures in Excel and add calculation capabilities that don't exist in Word. Unfortunately, most textbooks on VB for Office 97 are focused on nonmathematical features such as text formatting or graphic displays. This situation creates a significant challenge for those who want to use VB macros for their superior mathematical capabilities, because the method for referencing numerical values in a cell of a Word table is significantly different from referencing a cell in an Excel spreadsheet. If you're sufficiently persistent and a little clairvoyant, you can figure out the correct syntax from the Help found in the VB editors for Excel and Word. You can also get an almost complete set of hints on the subject from *Microsoft Office 97: Visual Basic Programmer's Guide* by The Microsoft Corporation (Microsoft Press, 1997, ISBN: 1572313404). However, these are the only examples I know of that spell it out explicitly. They calculate the sum of cubes of the values for the first eight cells of column 1 and place the result in the ninth cell of column 1.

For Word VBA:

```
Sub columnmath()
   Dim x As Long, i As Long
   Dim myTable As Table
   Dim myStr As String

   x = 0
   Set myTable = ActiveDocument.Tables(1)
   For i = 1 To 8
      x = x + myTable.Cell(i, 1).Range.Calculate ^ 3
   Next i
   myStr = Str(x)
   myTable.Cell(9, 1).Range.InsertAfter (myStr)
End Sub
```

For Excel VBA:

```
Sub columnmath()
   Dim x As Long, i As Long
   Sheets("Sheet1").Activate
   x = 0
   For i = 1 To 8
      x = x + Cells(i, 1).Value ^ 3
   Next i
   Range("a9").Value = x
End Sub
```

Although this calculation can be done entirely within the capabilities of Excel, it would take up an extra column. The calculation can't be done at all within Word. In trying to make sense out of it all, consider that Word VBA has a nonintuitive syntax for referencing cells. To read the value of a cell, the terminal word must be Calculate. To write a value into a cell, it must first be converted to a string, and the nonintuitive terminal InsertAfter must be used. Tables are numbered consecutively in a Word document; therefore the Word example deals with the first table in the document. Each sheet of an Excel workbook is just one big table; therefore, the Excel example deals with the first sheet of the workbook. In the Excel example, the Range("a9") could be replaced by Cells(9,1), or one of several other variants involving the use of Range.

—Peter Gottlieb, Los Angeles, California

VB6

Level: Intermediate

Expose MultiUse Classes in ActiveX Control Projects

A lesser-known, new feature in VB6 is the ability to have MultiUse and GlobalMultiUse classes within an ActiveX control project. This is useful if you have ActiveX controls and creatable classes you want to expose from the same project.

There was no way to do this in VB5. In VB6, it's as easy as adding a class module to your ActiveX control project. Unfortunately, there's no way to do the converse; you can't have a public user control in an ActiveX DLL project. So, if you want creatable classes and public user controls in the same project, you have to make it an ActiveX Control project. If you prefer your component to be a DLL instead of an OCX, just rename the file; internally, an OCX is the same as a DLL.

—Russell Davis, Garden Grove, California

VB4 32, VB5, VB6

Level: Beginning

Enumerate Treeview Nodes Recursively

Trying to parse a set of TreeView nodes and their children's nodes and *their* children's nodes can be confusing. This algorithm makes the process easier. Recursion describes an algorithm that can call itself. This is especially useful in COM's object hierarchy. Collections that can reference other collections can be easily handled with a recursive procedure.

To begin the recursion, place this code in a CommandButton labeled "View Nodes." Here you create a variable to store the results of the procedure and start the recursion, and also display the results when the routine is finished:

```
Dim N As Node, aNodes As String, lLevel As Long
Set N = TreeView1.SelectedItem
If N.Children Then aNodes = "+" & N.Text Else aNodes = N.Text
EnumChildren N, aNodes, lLevel
MsgBox aNodes
```

Next is the recursive procedure. It calls itself after changing the Node parameter, and the lLevel variable is incremented at the beginning of the routine and decremented at the end. This variable determines the distance to tab from the beginning of the display line, and therefore shows the nodes in the proper child-parent relationship:

```
Sub EnumChildren(N As Node, aNodes As String, lLevel As Long)
   lLevel = lLevel + 1
   Dim nC As Node
   If N.Children Then
      Set nC = N.Child
      Do
         If nC.Children Then
            aNodes = aNodes & vbCrLf & String$ _
               (lLevel, vbTab) & "+" & nC.Text
         Else
            aNodes = aNodes & vbCrLf & String$ _
               (lLevel, vbTab) & nC.Text
         End If
         EnumChildren nC, aNodes, lLevel
         If nC.Index = N.Child.LastSibling. Index Then Exit Do
         Set nC = nC.Next
      Loop
   End If
   lLevel = lLevel - 1
End Sub
```

—Matt Hart, Tulsa, Oklahoma

VB4 32, VB5, VB6

Level: Advanced

Generate Unique String IDs

If you need unique string IDs and don't have a sure-fire way of either generating or guaranteeing the generated ID is unique, then you need a Universally Unique ID (UUID) or Globally Unique ID (GUID) as Microsoft calls them. A UUID is a 128-bit number that's generated based on a time value and your computer's network interface card (NIC) and is guaranteed to be unique (at least within your network and until about the year 3400).

This routine generates UUIDs and converts them into 36-byte strings. Just paste this code into a module:

```
Option Explicit

Private Declare Function UuidCreate Lib _
   "rpcrt4.dll" (pId As UUID) As Long
Private Declare Function UuidToString Lib _
   "rpcrt4.dll" Alias "UuidToStringA" (uuidID _
   As UUID, ppUuid As Long) As Long
Private Declare Function RpcStringFree Lib _
   "rpcrt4.dll" Alias "RpcStringFreeA" _
   (ppStringUuid As Long) As Long
Private Declare Function CopyMemory Lib _
   "kernel32.dll" Alias "RtlMoveMemory" (pDst _
   As Any, pSrc As Any, ByVal nSize As Long) As Long

Private Type UUID
   Data1 As Long
   Data2 As Long
   Data3 As Long
   Data4(8) As Byte
End Type

Public Function GenUuid(sUuid As String) As Boolean
   Const RPC_S_OK As Long = 0
   Const SZ_UUID_LEN As Long = 36
   Dim uuidID As UUID
   Dim sUid As String
   Dim ppUuid As Long

   sUid = String(SZ_UUID_LEN, 0)
   If UuidCreate(uuidID) = RPC_S_OK Then
      If UuidToString(uuidID, ppUuid) = RPC_S_OK Then
         CopyMemory ByVal sUid, ByVal ppUuid, SZ_UUID_LEN
         If RpcStringFree(ppUuid) = RPC_S_OK Then
            sUuid = sUid
            GenUuid = True
         End If
      End If
   End If
End Function
```

Use the function like this:

```
Dim sId As String
Call GenUuid(sId)
MsgBox "Id is " & sId
```

I'm sure you can find better uses for the ID than just displaying it. But be aware that these numbers have the potential to uniquely identify the machine on which they were generated, raising security concerns. Depending on your customers and how you use the ID, this might not be an issue for you.

—Edward Lennox, Toronto, Ontario, Canada

VB4 32, VB5, VB6
Level: Intermediate

Share Variables Between Multiple Apps
Sometimes it's necessary to share variables between multiple instances of an application. For example, you might want to share a database connection. Here's how: Create an ActiveX EXE with an exposed MultiUse class named Class1. An ActiveX DLL won't work because it's running in-process within each client. Include a module in the project and declare the required variable you want to share in this module as a public variable.

The code in an ActiveX DLL looks like this:

```
'**** Module code *****
Global gCon As Database
Global glngNrOfObjects As Long

'**** Code in Class ****
Public Property Set DBConn(Con As Database)
   Set gCon = Con
End Property
Public Property Get DBConn() As Database
   Set DBConn = gCon
End Property

Private Sub Class_Initialize()
   ' this variable keeps count of objects created
   glngNrOfObjects = glngNrOfObjects + 1
End Sub
Private Sub Class_Terminate()
   ' decrease object count
   glngNrOfObjects = glngNrOfObjects - 1

   ' if no objects exist, close the connection
   If glngNrOfObjects = 0 Then
      If Not gCon Is Nothing Then
         gCon.Close
         Set gCon = Nothing
      End If
   End If
End Sub
```

Now reference this ActiveX EXE in your project and use it. Here's how I did it:

```
'** Code in application using the ActiveX EXE **

Dim clsDBConn As Class1

Private Sub Form_Load()
   Set clsDBConn = New Class1
   ' set the connection, when first object is created
   If clsDBConn.DBConn Is Nothing Then
      Set clsDBConn.DBConn = _
         Workspaces(0).OpenDatabase("\temp\db1.mdb")
      End If
End Sub

Private Sub Form_Unload(Cancel As Integer)
   Set clsDBConn = Nothing
End Sub
```

You don't need to include the code to count the object instances and close the database if your variable is a simple one—that is, if it's not an object reference. Also, in that case, change the Property Set to Property Let.

Both the projects have a reference to DAO because they're using the database object.

—Ravindra Okade, Plainsboro, New Jersey

VB5, VB6
Level: Beginning

Check It Out
First swallow your food, then open the Object Browser (F2) in either VB5 or VB6, select the VBRUN library, and read the definition of ContainedControls.

—Edward Lennox, Toronto, Ontario, Canada

VB5, VB6
Level: Intermediate

Add Your Icon to the Add-Ins Menu
Say you've written a cool add-in for the VB design environment. It's got its own menu item in the Add-Ins menu, and when you click on it, your add-in launches with a snazzy icon on its titlebar. Wouldn't it be great if you could have that icon displayed next to your add-in's menu item, too? Well, you can. Convert the icon to a 16-by-16 bitmap (because icons won't work in this process) and either add the bitmap to your app's resource file or drop it into an invisible Image control on your main form. This code assumes you're using an Image control. Use this in your Connect class's AddToAddInCommandBar method:

```
Dim cbMenuCommandBar As Office.CommandBarButton
Dim cbMenu As Object
' See if we can find the Add-Ins menu
Set cbMenu = VBInstance.CommandBars("Add-Ins")
If Not cbMenu Is Nothing Then
   ' Add it to the command bar
   Set cbMenuCommandBar = cbMenu.Controls.Add(1)
   ' Set the caption
   cbMenuCommandBar.Caption = "My Add-in"
   ' Paste an image
   Clipboard.Clear
   Clipboard.SetData frmAddIn.imgMenuPic.Picture
   cbMenuCommandBar.PasteFace
   Set AddToAddInCommandBar = cbMenuCommandBar
End If
```

—Ben Baird, Twin Falls, Idaho

VB4 32, VB5, VB6
Level: Advanced

Load Tree Subnodes on Demand
Here's a quick way to drop a drive's folder hierarchy into a TreeView control. The advantage to using this method is that folders are enumerated only when a node is expanded, so your app won't waste time adding folders the user will never look at. A dummy item is added to each folder to get a plus sign on the node, but the item is removed when the node is expanded or simply disappears if the folder has no subdirectories. This code assumes your TreeView is named tvFiles and you have a button named cmdEnum on the form. The TreeView is connected to an ImageList whose first image is a folder icon (for the code listing, download the RTF file for this supplement under the Code section on *VBPJ*'s Web site, www.vbpj.com).

—Ben Baird, Twin Falls, Idaho

VB4 32, VB5, VB6
Level: Intermediate

Translate OLE_COLOR to Actual RGB Value

Have you ever tried to pass a VB system color constant—such as vbButtonFace—to an API call that asks for a color? I frequently need to use system colors for GDI calls, and prefer to use the VB system color constants. The problem is that GDI doesn't know what to do with them and the resultant color is always black. The solution is simple: Use the OleTranslateColor API, which takes any of these constants and converts them to literal RGB colors GDI can understand. I usually package the API in a simple wrapper as well:

```
Private Declare Function OleTranslateColor _
   Lib "oleaut32.dll" (ByVal lOleColor As Long, _
   ByVal lHPalette As Long, lColorRef As Long) _
   As LongPublic Function TranslateColor(inCol _
   As OLE_COLOR) As Long
   Dim retCol As Long
   OleTranslateColor inCol, 0&, retCol
   TranslateColor = retCol
End Function
```

From that point on, simply call TranslateColor() with a system color constant to get the color you need. Furthermore, if a standard RGB value is passed to TranslateColor, it returns unaltered, so you don't have to worry about whether a color value you're storing is a system constant or an actual color value.

—Ben Baird, Twin Falls, Idaho

VB4 32, VB5, VB6
Level: Intermediate

Remove Unwanted System Menu Options

You've probably wanted to limit the normal operations of a form, such as resizing it, preventing it from being minimized or maximized, or allowing it to be closed only when you say so. The trick is to remove the control menu that corresponds to the functionality you want to limit. For example, when you remove the Size menu from the Control menu, the user won't be able to resize your form. The same goes for Minimize, Maximize, and Move. For the code to add to a BAS module, download the RTF file for this supplement under the Code section on *VBPJ*'s Web site, www.vbpj.com.

Then, when you load your form, call VBRemoveMenu with one of the menu enumerations:

```
   VBRemoveMenu Me, rmMaximize
```

Not only will the menu be removed from the Control menu, but the corresponding functionality of the form will be removed as well. Because the menus are removed by ordinal position, remove them in descending order if you want to remove more than one. For example:

```
Private Sub Form_Load()
   Call VBRemoveMenu(Me, rmClose)
   Call VBRemoveMenu(Me, rmMaximize)
End Sub
```

—Earl Damron, Louisville, Kentucky

VB4 32, VB5, VB6
Level: Intermediate

☆☆☆☆☆ **Five Star Tip**

Display Proprietary Information in a WebBrowser Control Without an HTML File

When using the WebBrowser control in VB, you might want to present information without having that data accessible to the end user. In this case, you can simply insert the HTML code directly into the control. First create a blank document within the control, then set the HTML text directly—you don't need an external HTML file. This method also protects your HTML code; if users choose to View Source, all they see is "<HTML></HTML>":

```
Private Sub Form_Load()
   Dim strHTMLText As String
   ' Create a blank document in the WebBrowser control
   WebBrowser1.Navigate2 "about:Blank"
   ' Web browser may take awhile to process each command
   DoEvents
   On Error GoTo WaitAwhileLonger
   ' Set the backcolor here
   WebBrowser1.Document.body.bgcolor = "#000000"
   ' Set the HTML Text through code or from a Database
   strHTMLText = "<html>" & vbCrLf & _
      "<head>" & vbCrLf & _
      "<title>Common Controls Replacement" & _
      "Project</title>" & vbCrLf & _
      "</head>" & vbCrLf & "<body>" & _
      "<p align=""center"">_
      <font face=""Arial"" size=""5"" " & _
      "color=""#FFFFFF""><strong>_
      The Common Controls " & _
      "Replacement Project</strong></font></p>" _
      & "<p align=""center""> _
      <a href=""http://www.mvps.org/ccrp"">" & _
      "<img " & "src=" _
      "http://www.mvps.org/ccrp/images/banner/" _
      & "ccrplogo.gif"" _
      alt=""Visit the CCRP's Internet Site." _
      " " & "border=""0""></a></p></body>" & _
      vbCrLf & "</html>"
   strHTMLText = strHTMLText & "<head>" & vbCrLf
   ' Send the HTML Text to directly to the
   ' WebBrowser Control
   WebBrowser1.Document.body.innerhtml = strHTMLText
Exit Sub

WaitAwhileLonger:
   Debug.Print Hex(Err.Number), Err.Description
   DoEvents
   Resume
End Sub
```

The WebBrowser control sometimes needs a little "encouragement" to fully finish the last task assigned before you can proceed with the next. That's the purpose of the error trap, which allows the WebBrowser a chance to catch its breath before attempting operations again.

—Ramon Guerrero, Garland, Texas

VB5, VB6
Level: Advanced

Swap Strings Faster
Here's a cool way to swap strings:

```
Declare Sub CopyMemory Lib "kernel32" Alias _
   "RtlMoveMemory" (Destination As Any, _
   Source As Any, ByVal Length As Long)
Sub SwapString(String1 As String, String2 _
   As String)
   Dim Save As Long
   ' This code swaps the string descriptors, not
   ' the data. StrPtr returns the address of the
   ' first character in a string. VarPtr returns
   ' the address of a string's descriptor, which
   ' is 4 bytes long and contains the address of
   ' the first character in the string. StrPtr
   ' and VarPtr are undocumented VB functions.
   Save = StrPtr(String1)
   Call CopyMemory(ByVal VarPtr(String1), _
      ByVal VarPtr(String2), 4)
   Call CopyMemory(ByVal VarPtr(String2), _
      Save, 4)
End Sub
```

Even for short strings, this is faster than the traditional method:

```
Sub SwapString(String1 As String, String2 As String)
   Dim Save As String
   Save = String1
   String1 = String2
   String2 = Save
End Sub
```

But as the strings grow in length, the speed difference gets more and more dramatic.

—Thomas Weiss, Deerfield, Illinois

VB3 and up
Level: Beginning

Make More Versatile Trim Functions
One of the nice things about VB is the ability to redefine most of the built-in commands. For example, you can extend the functionality of the Trim family of commands. LTrim and RTrim remove leading and trailing spaces from a string, but it also would be useful to remove other nonprinting characters, such as tabs, or carriage return/line feed pairs that might be present after reading in a text file. This code does exactly that by removing all leading and trailing characters with an ASCII value less than or equal to that of a space character:

```
Public Function Trim(ByVal inString As String) As String
   Trim = RTrim(LTrim(inString))
End Function

Public Function RTrim(ByVal inString As String) As String
   Dim nPos As Long
   nPos = Len(inString)
   If nPos > 0 Then
      Do While (Asc(Mid$(inString, nPos, 1)) <= 32)
         nPos = nPos - 1
         If nPos = 0 Then Exit Function
      Loop
      RTrim = Left$(inString, nPos)
   End If
End Function

Public Function LTrim(ByVal inString As String) As String
   Dim nPos As Long, nLen As Long
   nLen = Len(inString)
   If nLen > 0 Then
      nPos = 1
      Do While Asc(Mid$(inString, nPos, 1)) <= 32
         nPos = nPos + 1
         If nPos = nLen Then Exit Function
      Loop
      LTrim = Mid$(inString, nPos)
   End If
End Function
```

—John Cullen, Pedroucos, Portugal

VB3 and up
Level: Beginning

Create Nested Folders in One Call
Suppose you need to create a tree of directories, all at once, in code. For example, you could create the tree C:\1stDir\2ndDir\3rdDir\4thDir with one call simply by feeding that path, as a string, into this procedure:

```
Public Function MkDirs(ByVal PathIn As String) As Boolean
   Dim nPos As Long
   MkDirs = True  'assume success
   If Right$(PathIn, 1) <> "\" Then PathIn = PathIn + "\"
   nPos = InStr(1, PathIn, "\")
   Do While nPos > 0
      If Dir$(Left$(PathIn, nPos), _
         vbDirectory) = "" Then
         On Error GoTo Failed
            MkDir Left$(PathIn, nPos)
         On Error GoTo 0
      End If
      nPos = InStr(nPos + 1, PathIn, "\")
   Loop
   Exit Function
Failed:
   MkDirs = False
End Function
```

If any part of the path already exists, the routine creates only the new part. This routine works on strings representing local and mapped drives—those with a letter, colon, and backslash at the beginning. If the drive designation is left out, the directories are created starting at the default directory on the current drive.

—Frederick Rothstein, Trenton, New Jersey

VB4 16/32, VB5, VB6
Level: Intermediate

Calculate Date/Time Differences in Multiple Units
VB's DateDiff function works fine when you want to know the difference between two dates for a specific interval. To get the number of days between now and Christmas, you can use: DateDiff("d", Now, "25 Dec"). Sometimes you want to find the difference using more than one interval—for example, you might want to know the months and days until a special event, or the hours and minutes between two times. The DateIntervals routine allows you to pass in two dates and your own variables for the intervals you want. The routine fills the variables with

the largest full interval of the desired type, less any other intervals you request. To get the hours and minutes between 9 a.m. and 5:15 p.m., pass in two times and two variables, using commas to skip over the larger intervals you don't need:

```
DateIntervals "9:00am", "5:15pm", , , , Hours, Minutes
```

Upon returning, Hours holds 8, and Minutes holds 15. The results are always positive numbers, up to the limits of the DateDiff function:

```
Public Sub DateIntervals(ByVal Date1 As Date, _
   ByVal Date2 As Date, ParamArray Prams())
   ' Returns the greatest full interval (yr, mo,
   ' day, hr, min, sec) between two dates
   ' Calling procedure supplies variable(s) for
   ' the desired interval(s)
   ' Ex1: DateIntervals Birthday, Now, Yr, Mo,
   ' Dy, Hr, Mn, Sec <Exactly how old are you?
   ' Ex2: DateIntervals Now, "25 Dec", , , Days
   ' <How many days until Christmas?
   ' LFS: 1999 for VBPJ

   If UBound(Prams) < 0 Then Exit Sub

   Dim Temp As Date
   Dim i As Long, itr As String * 1
   Const interval = "mdhns"

   If (DateValue(Date1) = 0) Xor _
      (DateValue(Date2) = 0) Then
   ' Assume today if one is a time and the other
   ' is a date...
      If DateValue(Date1) = 0 Then _
         Date1 = Date1 + DateValue(Now)
      If DateValue(Date2) = 0 Then _
         Date2 = Date2 + DateValue(Now)
   End If
   If Date1 > Date2 Then
      ' Swap dates if first is after second...
      ' Temp = Date1
      Date1 = Date2
      Date2 = Temp
   End If
   If Not IsMissing(Prams(0)) Then
      Prams(0) = Year(Date2) - Year(Date1)
      Temp = DateAdd("yyyy", Prams(0), Date1)
      Prams(0) = Prams(0) + (Temp > Date2)
      Date1 = DateAdd("yyyy", Prams(0), Date1)
      If UBound(Prams) < 1 Then Exit Sub
   End If
   For i = 1 To IIf(UBound(Prams) > 5, 5, UBound(Prams))
      If Not IsMissing(Prams(i)) Then
         itr = Mid$(interval, i, 1)
         Prams(i) = DateDiff(itr, Date1, Date2)
         Prams(i) = Prams(i) + (DateAdd(itr, _
            Prams(i), Date1) > Date2)
         Date1 = DateAdd(itr, Prams(i), Date1)
      End If
   Next i
End Sub
```

—Larry Serflaten, Monticello, Minnesota

VB3 and up

Level: Beginning

Don't Guess Sizes

It's often useful to know the space occupied by a form's caption or menu bars—for example, for positioning or resizing a form. VB doesn't give direct access to this information, but a quick call to one of the Windows API functions does the job. Add these declarations to the General section of a form. Or add them to a separate module, in which case you make the declarations Public:

```
' for 16-bit Windows, ...
' Lib "user" (ByVal nIndex as Integer) as Integer
Private Declare Function GetSystemMetrics Lib _
   "user32" (ByVal nIndex As Long) As Long
Private Const SM_CYCAPTION = 4
Private Const SM_CYMENU = 15
```

Use this code to employ the function:

```
' menubar height in pixels
MenuHeight = GetSystemMetrics(SM_CYMENU)
' caption bar height in pixels
CaptionHeight = GetSystemMetrics(SM_CYCAPTION)
```

Note that these height values are in pixels, so you multiply them by Screen.TwipsPerPixelY to convert the value to twips. Search MSDN online or your own SDK docs for GetSystemMetrics to see the hundred or so other useful values this single API function can return.

—John Cullen, Pedroucos, Portugal

VB3 and up

Level: Beginning

Embed Double Quotation Marks

When you try to insert a string with single quotation mark into a text field of an Access or Oracle table, you get an error. But you can use the Chr(34) function to embed double quotation marks in a string passed to the Jet database engine:

```
Private Sub CmdTest_Click()
   Dim dbCustomer As Database
   Dim strSql As String
   Dim strCustID As String
   Dim strFirstName As String
   Dim strAddress As String
   Set dbCustomer = OpenDatabase("myconnect", _
      dbDriverNoPrompt, False, _
      "odbc;uid=scott;pwd=tiger;dsn=myconnect")
   strCustID = "A003"
   strFirstName = "Annie"
   strAddress = "Reflection's"
   strSql = "insert into CUSTOMER values('" & _
      strCustID & "'" & ",'" & strFirstName & _
      "'," & Chr(34) & strAddress & Chr(34) & ")"
   dbCustomer.Execute (strSql)
   dbCustomer.Close
End Sub
```

For more information on this problem, see article Q147687 in the Microsoft Knowledge Base.

—Mini Gopinath, San Ramon, California

VB3 and up
Level: Beginning

Watch Out for Root Installs
Be careful when reading the Path property of the App object. If the path is in a subdirectory, the resulting string terminates with the name of the subdirectory—for example, "C:\dir_x\dir_y\dir_z". Unfortunately, if the path happens to be the root directory, the resulting string terminates with a backslash—such as "C:\". The difference between the two is the possible terminating backslash.

If you use App.Path to find a program-related file, you need to add the backslash before the file name, except if the path is the root directory:

```
strFullFileName = App.Path & IIf(Right$( _
   App.Path, 1) = "\", "", "\") & strFileName
```

If this code is too tiresome to type, use this AppPath function:

```
Public Function AppPath() As String
   ' NOTE: Replace all occurrences of "App.Path"
   ' with "AppPath"
   Dim strAppPath As String
   strAppPath = App.Path
   If (Right$(strAppPath, 1) <> "\") Then
      strAppPath = strAppPath & "\"
   End If
   AppPath = strAppPath
End Function
```

If you automatically append a backslash to every App.Path call and the path happens to be the root directory, you get the uninformative error message, "Run-time error '5': Invalid procedure call."

—Andrew J. Marshall, Fairfax, Virginia

VB3 and up
Level: Beginning

Toggle Textbox Word Wrap at Run Time
On page 2 of "101 Tech Tips for VB Developers" [Supplement to *VBPJ*, August 1998], the first item is "Change the Appearance Property of a Text Box at Run Time." Here's a similar trick that's more useful and needs few lines of code. I believe it's compatible with all VB platforms, with no platform-specific routines or functions, Property Let or Get procedures, or API calls.

The textbox's Appearance property's limitation applies to the MultiLine and ScrollBars properties as well: these properties cannot be set at run time. But you can implement a functional Word Wrap feature for a textbox by deceiving the mind into believing these properties are changeable at run time. Place two textboxes on a form, leaving them named Text1 and Text2. Use the menu editor to create a menu called Text (mnuText) and a menu element called Word Wrap (mnuTextWrapToggle). Set Text1's properties as MultiLine = True and ScrollBars = 2 - Vertical. Set Text2's properties as MultiLine = True and ScrollBars = 3 - Both. Then add this code in the appropriate events:

```
Private Sub Form_Load()
   ' Add sufficient text to Text2 for demo purposes
   Text2.Text = "Word upon "
   Text2.SelStart = Len(Text2.Text)
   For i = 0 To 100
      Text2.SelText = "word upon "
   Next i
   Text2.SelText = "word."
   Text2.ZOrder 0
End Sub

Private Sub Form_Resize()
   ' Make text boxes resize with form
   Text1.Move 0, 0, .ScaleWidth, .ScaleHeight
   Text2.Move 0, 0, .ScaleWidth, .ScaleHeight
End Sub

Private Sub mnuTextWrapToggle_Click()
   ' Toggle word wrapping On or Off
   mnuTextWrapToggle.Checked = Not _
      mnuTextWrapToggle.Checked
   If mnuTextWrapToggle.Checked Then
      Text1.Text = Text2.Text
      Text1.ZOrder 0 'Make topmost
   Else
      Text2.Text = Text1.Text
      Text2.ZOrder 0 'Make topmost
   End If
End Sub
```

—George W. Hetherington, Newcastle Upon Tyne, England

VB4 16/32, VB5, VB6
Level: Beginning

Q&D Way to Hide the Mouse Pointer
Here's a quick and easy way to hide the mouse pointer without using an API call: Don't hide it. Instead, use a custom pointer that's 100 percent transparent. Create an icon that has nothing but a flood of the "transparent" color—any icon editor should be able to create one. Set this icon as the Screen object's MouseIcon property. When you want to hide the pointer, set the Screen's MousePointer property to vbCustom (99). The pointer is still on the screen, but there's nothing to see. You can still move the mouse, and click on objects, albeit with some difficulty. This demo program toggles the pointer on and off with each click of the Command1 button:

```
Private Sub Form_Load()
   ' Set the Form's MouseIcon property at
   ' design time so there is no external file to
   ' load at run time.
   Screen.MouseIcon = Me.MouseIcon
End Sub

Private Sub Command1_Click()
   If Screen.MousePointer = vbCustom Then
      Screen.MousePointer = vbDefault
      Command1.Caption = "Hide"
   Else
      Screen.MousePointer = vbCustom
      Command1.Caption = "Show"
   End If
End Sub
```

You can click on Command1, and drag the invisible pointer to select text in the textbox. This is ideal for a screen saver, where you want to react to mouse events but don't want to show a pointer and don't want to bother with API calls.

—Bob Ashcraft, Winchester, Virginia

VB4 16/32, VB5, VB6

Level: Intermediate

Autoinstantiation Isn't Automatic

The Tip entitled "Declare Your Objects Properly" ["101 Tech Tips for VB Developers," Supplement to *VBPJ*, August 1998, page 23] stated, "Never declare an Object variable as New," which can be good advice in many circumstances. It went on to erroneously state, "If you do, you'll always increment the reference count of the object, regardless of whether you use it." This statement is incorrect. When using an As New declaration, an object is not instantiated until it's first used. Run this test as proof. The reference count is not incremented and the object will not be created until the TestMethod method is invoked:

```
'---In Form1
Private Sub Form_Load()
   Dim TestObject As New Class1
   MsgBox "1) The object has been declared but not used."
   TestObject.TestMethod
End Sub

'----In Class1
Private Sub Class_Initialize()
   MsgBox "2) The object has now been instantiated."
End Sub

Public Function TestMethod()
   MsgBox "3) The method is being invoked."
End Function
```

Beware—this lazy object instantiation comes at a price. Each time the object is used, the code checks whether the object has been instantiated. The checking creates a performance hit, especially for frequently used objects. This technique also can obscure the location where the object is being instantiated, which can make code harder to read and bugs harder to find. For these reasons, it's usually better not to declare objects using As New. However, in some situations, the delayed instantiation offered by an As New declaration can be helpful. One appropriate situation involves objects used sporadically in an application. For example, if an object is used only in error handling, the developer can't predict where, when, or even if the object will be instantiated. In cases like this, the As New declaration might be ideal. The object won't be created until it is used, and if it's not needed it won't be created.

—Rick Lindstrom, Shelton, Connecticut

VB3 and up

Level: Beginning

Make Menus at Run Time

Many programmers forget that menu items act like controls, too. For example, you can set their Caption, Enabled, and Visible properties as you can with other controls. If you give a menu item an Index property, you can also create and remove menus at run time using the Load and Unload statements. Using these methods, you can make menus with a variable number of items.

—Rod Stephens, Boulder, Colorado

VB4 32, VB5, VB6

Level: Intermediate

Create an ODBC Entry

You can create, edit, or delete your program's ODBC entry automatically. This code checks if an ODBC source has been made; if not, it configures one according to your program's specification. To edit or remove an existing ODBC source, change the fRequest parameter to the ODBC_CONFIG_DSN or ODBC_REMOVE_DSN values, respectively:

```
Declare Function SQLConfigDataSource Lib _
   "ODBCCP32.DLL" (ByVal hwndParent As Long, _
   ByVal fRequest As Long, ByVal lpszDriver _
   As String, ByVal lpszAttributes As String) As Long
Public Sub MakeODBCDataSource()
   Const ODBC_ADD_DSN = 1
   ' Add data source
   Const ODBC_CONFIG_DSN = 2
   ' Configure (edit) data source
   Const ODBC_REMOVE_DSN = 3
   ' Remove data source
   Const vbAPINull As Long = 0&
   ' NULL Pointer
   Dim lngRet
   ' Check if it has been done, only do this once
   If GetSetting(App.ExeName, "options", _
      "ODBCSetup", "No") = "No" Then
      Dim sDriver As String
      Dim sAttributes As String
      sDriver = "Microsoft Access Driver (*.mdb)"
      sAttributes = sAttributes & "DSN=MyDSN" & Chr$(0)
      sAttributes = sAttributes & _
         "DBQ=C:\Temp\Myfile.mdb" &  Chr$(0)
      lngRet = SQLConfigDataSource(vbAPINull, _
         ODBC_ADD_DSN, sDriver, sAttributes)
      SaveSetting App.ExeName, "options", _
         "ODBCSetup", "Yes"
   End if
End Sub
```

—Dan Newsome, received by e-mail

VB4, VB5, VB6

Level: Beginning

Use SQL Statement in DAO

When using DAO, if you use a DataControl to populate a bound grid quickly, it's better to send the control's RecordSource property a full SQL statement instead of passing a DAO Recordset object reference to the DataControl's RecordSet property. When you pass a reference to a created Recordset object, the DataControl essentially reverse-engineers the Recordset object to SQL to figure field headers and other information. Once the DataControl hits the 255-character limit, it generates an error that implies there was a problem in your SQL statement (that doesn't exist). However, if you pass a long and complex SQL string to the control's RecordSource property, the control doesn't do any analysis—it simply generates the result and displays it in bound controls.

—Robert Smith, Kirkland, Washington

VB3 and up
Level: Beginning

Use DAO Code for Non-Jet Sources

DAO still has a number of features that make it wonderful, especially for ISAM-based apps. For example, it's simple to use DAO code against non-Jet data sources such as Text(csv) and HTML tables with no MDB linking or importing required. Just use the correct ISAM specifier as part of the OpenDatabase method:

```
Dim db As Database
Dim tDef As TableDef
Dim Fld As Field
Set db = Workspaces(0).OpenDatabase("C:\MyDBFs", _
   False, False, "dBase III;")
For Each tDef In db.TableDefs
   Debug.Print "Table: " & tDef.Name
   For Each Fld In tDef.Fields
      Debug.Print Fld.Name
   Next
Next
db.Close
Set db = Nothing
```

For dBase III, dBase IV, Text, Excel 3 or 4, and HTML sources, the "Database" is the path to the directory in which the "tables" (files) are located. For Excel 5 (Excel 95) and Excel 97, the Database argument must be fully qualified to include the XLS file because later Excel files are workbooks that encapsulate worksheets the way Access encapsulates its tables. For more information, see the VB online help for the Connect (DAO) property and note that the specifier for Excel 97 is "Excel 8.0;" not "Excel97;" as the help page says.

—Robert Smith, Kirkland, Washington

VB3 and up
Level: Beginning

Find Matching Records as you Type

You can construct a search box to find information as you type using the textbox's Change event. A SQL statement with the Like operator and an "*" added to the end gives you the matching record as you type. This example uses option buttons to allow searches in different fields. I have used this feature in several programs—some with more than 4,000 records—and it is quite fast:

```
Private Sub searchbox_Change()
   On Error Resume Next
   Select Case True
      Case Option1.Value
         Data1.Recordset.FindFirst _
            "[accountnumber] Like '" & _
            searchbox.Text + "*" & "'"
      Case Option2.Value
         Data1.Recordset.FindFirst _
            "[customername] Like '" & _
            searchbox.Text + "*" & "'"
      Case Option3.Value
         Data1.Recordset.FindFirst _
            "[businesstype] Like '" & _
            searchbox.Text + "*" & "'"
   End Select
End Sub
```

—Barry Rudd, Waycross, Georgia

VB3 and up
Level: Beginning

Manage SQL Statements with a Resource File

Managing SQL statements in your code can get unwieldy in a large app. The job is simpler when you use either a resource file or constants to store the SQL's basic structure and a couple of functions. In this code, you replace the "?" in the query's parameter spot with actual criteria and return a completed SQL statement ready to run:

```
' Get the formatted stored proc or SQL to use
' Parameters:
'  sSQL        String      Stored procedure, '?'
' indicate placeholders
'  sVarList    Paramarray
' List of values to replace, SHOULD MATCH
' THE NO. OF PLACEHOLDERS.
' Return:
'  String      Modified SP/SQL to use
'-------------------------------------------
Public Function GetSQL(ByVal sSQL As String, _
   ParamArray sVarList()) As String
   Dim iPos As Integer
   Dim i As Integer
   For i = LBound(sVarList) To UBound(sVarList)
      iPos = InStr(sSQL, "?")
      If iPos <> 0 Then
         sSQL = Left$(sSQL, iPos - 1) & _
            ParseForSQL(sVarList(i)) & _
            Mid$(sSQL, iPos + 1)
      End If
   Next i
   GetSQL = sSQL
End Function
```

—Chun Wong, Morden, England

VB5 (with ADO), VB6, VBA
Level: Intermediate

List All Row-Returning Objects With adSchemaTables

The Connection object's OpenSchema method permits your application to browse the collections available through an ADO connection without enumeration. The method returns a recordset with fields characterizing the members of a collection. The output from the OpenSchema method with an adSchemaTables query type against the Jet 4.0 data provider contains information about local tables, linked tables, pass-through queries, system tables, and the Access table. You can specify any of more than 30 query type arguments to gather information about the contents of an OLE DB data source. Besides the adSchemaTables query type, selected others return information about familiar database objects such as check constraints, indexes, primary keys, foreign keys, procedures, and views. This listing reveals how you can use the OpenSchema method with the Jet 4.0 provider to list the views available through a connection:

```
Public Sub OpenSchemaX()
   Dim cnn1 As New ADODB.Connection
   Dim rstSchema As ADODB.Recordset
   cnn1.Open _
      "Provider=Microsoft.Jet.OLEDB.4.0;" & _
      "Data Source=C:\Program Files\" & _
      "Microsoft Office\Office\Samples\Northwind.mdb;"
```

```
   Set rstSchema = cnn1.OpenSchema(adSchemaTables)
   ' Print just views; other selection criteria
   ' include TABLE, LINK, PASS-THROUGH, ACCESS
   ' TABLE, and SYSTEM TABLE
   Do Until rstSchema.EOF
      If rstSchema.Fields("TABLE_TYPE") = "VIEW" Then
         Debug.Print "View name: " & _
            RstSchema.Fields("TABLE_NAME") & vbCr
      End If
      rstSchema.MoveNext
   Loop
   rstSchema.Close
   cnn1.Close
End Sub
```

—Rick Dobson, Louisville, Kentucky

VB6, VBA

Level: Beginning

Instant Recordset Contents With the GetString Method

The GetString method replaces a pair of nested loops. If the defaults are acceptable, you can use the method without any arguments. This makes for a simple way to extract values from a recordset. Although nested loops can help you grasp how the columns within a row combine, the GetString method can achieve a similar result in single line. It accomplishes this without even one loop, whereas you need two loops to enumerate the columns within a row automatically.

The GetString method, which returns a recordset as a string, can take up to five arguments. This sample illustrates the use of three of those arguments. Designate the adClipString constant as the first argument, which is your only choice for this argument. It specifies the format for representing the recordset as a string. The second argument specifies the number of recordset rows to return—five, in this case. Leaving this argument blank would enable the method to return all the rows in the recordset. The third argument designates a semicolon delimiter for the columns within a row (the default column delimiter is a tab):

```
Sub NoEasyLoop()
   Dim rsCustomers As Recordset
   Set rsCustomers = New ADODB.Recordset
   rsCustomers.Open "customers", _
      "Provider=Microsoft.Jet.OLEDB.4.0;" & _
      "Data Source=C:\Program Files\" & _
      "Microsoft Office\Office\Samples\Northwind.mdb;"
   ' Print records without a loop
   Debug.Print rsCustomers.GetString(adClipString, 5, "; ")
   rsCustomers.Close
End Sub
```

The fourth and fifth arguments, neither of which appears in the sample, specify a column delimiter and an expression to represent null values. The default values for these arguments are a carriage return and a zero-length string.

—Rick Dobson, Louisville, Kentucky

VB3 and up

Level: Beginning

End Your Applications Gracefully

Although you can use the End statement to halt execution immediately, it's likely to prevent memory and system resources from being released. Instead, try unloading all forms and setting all globally defined objects to Nothing to shut down your application. Here's a routine you can adapt to end your application more gracefully than using the End statement:

```
Function ExitApplication() As Long
' PURPOSE:
'  Unloads all forms, releases all file locks,
'  and destroys all user-defined objects Returns:
'  0  on success
'  >0 on Failure
   Dim i As Integer
   Dim lngRetVal As Long
   ' Initialize routine
   On Error GoTo ExitApplication_EH1
   ExitApplication = 0  ' assume success
   lngRetVal = 0
   ' Unload all forms
   For i = Forms.Count - 1 To 0 Step -1
      Unload Forms(i)
   Next
   ' Destroy all "global" objects
   Set gobjUser = Nothing
   Set gobjSession = Nothing
ExitApplication_Exit:
   On Error Resume Next
   Close    ' release any file locks
   DoEvents
   ExitApplication = lngRetVal
   Exit Function
ExitApplication_EH1:
   lngRetVal = Err
   Resume ExitApplication_Exit
End Function
```

—Jim De Carli, Southbury, Connecticut

VB4 and up

Level: Beginning

Avoid Missing References

Occasionally you might get an error message such as "Connection to type library or object library for remote process has been lost. Press OK for dialog to remove reference," "Undefined Function," or "Can't find project or library." You get this kind of error for two reasons: Either the type library for your component project is not available or the GUID of your component's type library has changed. The right solution varies depending on the situation, but here's a quick fix to try first.

Choose Project | References to open the References dialog box. (In VB4, choose Tools | References.) Scan through the list of available references, looking for any with a description that begins with the text "Missing."

For every entry marked as missing, note which component it refers to. When you highlight the entry, the fully qualified path to the component is displayed at the bottom of the dialog. Make a note of this file specification because you will need it later. Clear the selection, and repeat these steps for any other missing reference. Then click on the OK button to close the dialog.

Open the References dialog box again. Find each of the entries you previously cleared and select them. If an entry is no longer listed in the dialog, use the Browse button to select the file that entry refers to. Once all references have been reselected, click on the OK button to close the dialog. Then rerun your project.

—Jim De Carli, Southbury, Connecticut

VB3 and up
Level: Intermediate

Use Fill Properties With API Functions
The GDI drawing functions honor VB's Fill properties. For example, when you use the Polygon API function, the system fills the polygon using the current FillStyle and FillColor properties. Use these properties to specify how to fill the polygon instead of using GDI's more complicated pens and brushes.

—Rod Stephens, Boulder, Colorado

VB3 and up
Level: Beginning

Compare Floating-Point Numbers Safely
Single and Double variables usually don't store values exactly, so you can't safely compare them for equality. For example, in this code, rounding makes VB think that a - b = 0.2000008, so the If statement is never satisfied:

```
Dim a As Single
Dim b As Single
   a = 40
   b = 39.8
   If a - b = 0.2 Then
      MsgBox "a = b"
   Else
      MsgBox "a <> b"
   End If
```

To see if two floating-point values are equal, subtract them and compare the absolute value of the result to some small value:

```
If Abs((a - b) - 0.2) < 0.0001 Then ...
```

—Rod Stephens, Boulder, Colorado

VB5, VB6
Level: Intermediate

Avoid In-Process Component Conflicts
Each time you create an in-process component, a base address is recorded. When your VB application loads the component, the component is placed in a memory location according to its base address. If two components have the same base address, a conflict occurs, and time is wasted as all the offsets within the second component are adjusted. Base address conflicts are somewhat common because many people rely on using the default values when developing their components.

To specify a new base address for your component, use the Compile tab in the Project Properties dialog. The base address is entered in the DLL Base Address textbox. The default value is &H11000000, but the operating system can accept any value from &H1000000 and &H80000000. Because the operating system doesn't like to have components loaded above 2 gigabytes (the high-end value of &H80000000), the maximum value for the base address is effectively 2 gigabytes minus the size of your component rounded up to the nearest 64K multiple.

—Jim De Carli, Southbury, Connecticut

VB4 and up
Level: Intermediate

Use Properties to Keep Controls Private
With object-oriented design, avoid referencing a form's controls from anywhere outside the form. The controls should be considered private. If you need to get to information in a control, establish a form property for it, and have the Property Let and Property Get procedures manage the control's value.

Suppose you have a form named frmOptions that uses five option buttons—named optFormat(0) through optFormat(4)—to allow the user to specify a format type for a report. You need to find the format type from outside the form (say from a BAS module). With old-fashioned VB syntax, you would access the format type like this:

```
For nIndex = 0 To 4
   If frmOptions.optFormat(nIndex) Then
      nFormatNumber = nIndex
      Exit For
   End If
Next nIndex
```

But what if you later have to change either the control name or the number of option buttons? Or what if you have to switch to a combo box holding the format types because they increase in number and start varying? In that case, code from outside the form—such as the lines above—would all have to change. Instead, at the outset, add a property to the form to allow outside code to fetch the format type. You could call the property FormatType:

```
Public Property Get FormatType () As Long
   For nIndex = 0 To 4
      If optFormat(nIndex) Then
         FormatType = nIndex
         Exit For
      End If
   Next nIndex
End Property
```

Now calling code from outside the form can get the format type with a single line:

```
nFormatNumber = frmOptions.FormatType
```

If you now have to change the controls for the format type, adjust the property procedure—inside frmOptions—to reflect the changes. For example, if you switch to a combo box, the property procedure would look something like this:

```
Public Property Get FormatType () As Long
   FormatType = cboFormat.ListIndex
End Property
```

Calling code—from outside frmOptions—doesn't need to change at all. Ideally, the property procedures would contain some error checking, which I have left out for simplicity. You would also want Property Let procedures for setting the controls from outside the form. The small amount of additional work in creating the property procedure, instead of accessing the controls directly, can provide a big payoff in maintainability.

—Billy Hollis, Nashville, Tennessee

VB3 and up
Level: Beginning

Use Mid$ on the Left Side of an Assignment
I'm always surprised when VB programmers don't know an old trick from QuickBASIC: Using the Mid$ function on the left side of an assignment. Here's an example:

```
Dim sName as string
sName = "Jack Smith, Jr."
Mid$(sName, 6, 5) = "Jones"
```

When this code is finished, sName contains "Jack Jones, Jr.". This use of Mid$ is a lot easier—and a *lot* faster—than the way I've seen other VB programmers do the same operation:

```
sName = left$(sName, 6) & "Jones" & right$(sName, 4)
```

—Billy Hollis, Nashville, Tennessee

VB3 and up
Level: Beginning

Force Arguments to be Passed by Value
When you pass parameters to a function or procedure, the VB default is to pass them by reference unless the procedure or function's declaration specifies ByVal. But sometimes you want to pass a parameter by value, regardless of what the procedure's declaration specifies. In that case, put an extra set of parentheses around each argument you want to pass by value. The parentheses cause the argument to be passed by value, even though the procedure's declaration might show the parameter is passed by reference.

Assuming the sub or function's parameter is defined as being passed by reference, pass the parameter by value with this code:

```
intValue = FindValue((intStart), blnFlag)
ShowValue(strText)  or  Call ShowValue((strText))
```

To pass the parameter by reference:

```
intValue = FindValue(intStart, blnFlag)
ShowValue strText  or  Call ShowValue(strText)
```

—Fan Wang, Tampa, Florida

VB4 16/32, VB5, VBA
Level: Intermediate

Use Checkboxes Instead of Option Buttons in a Group
Checkbox and option button controls function similarly but with an important difference: A user can select any number of checkbox controls on a form at the same time, but can select only one option button control in a group at one time.

A small piece of code allows the user to select only one checkbox in a group at any given time. This change is useful when you want to use checkboxes instead of option buttons. Create a control array of checkboxes named chkOption. These controls can be inside a frame or directly on the form. Place this code in the control array's Click event:

```
Private Sub chkOption_Click(Index As Integer)
   Dim i As Integer
   Static blnIntChg As Boolean
   If blnIntChg = False Then 'user clicked it
      blnIntChg = True
      'to stop racing when internally other
      'checkboxes are changed
      For i = 0 To chkOption().UBound
         chkOption(i).Value = IIf(i = Index, 1, 0)
      Next i
      blnIntChg = False
      'Ready for next user click
   End If  'external change
End Sub
```

The code sets the clicked-on checkbox to 1 and all others to zero. When this code sets the Value property of the checkboxes, the Click event gets called as many times as the number of controls. To prevent a racing condition and subsequent stack fault, the code sets the blnIntChg flag to True when the user causes the Click event to fire. The internal calls to Click events are ignored. When all checkboxes have been set or reset, the code resets the flag for the next user click.

—Debashish Roy, Cleveland, Ohio

VB6
Level: Intermediate

Put Unbound Data Into the Data Report's Page and Report Sections
If you've tried putting live data into the Data Report Designer's nondetail sections, you learned that bound controls are a no-no. Here's how you can put any data from any database into these sections without binding it to the Data Report directly. By placing labels in the sections where you want the data to appear, you can change their Caption properties in code to reflect the data desired.

First make sure you have a form in your app. Then in the form where the report is launched—for example, frmAux—put one label for each data item you want to see in the restricted report sections on the form.

Create a data source or use one previously created, such as DED, ADODC, or ADO recordsets, and make all the necessary changes for putting the data you want to see into the report. Bind this data source to the textbox you created. Write code similar to this in the Initialize event of the Data Report. In this sample, "Sections(5)" is the Report Footer:

```
With frmAux
   DataReport1.Sections(5).Controls( _
      "lblCreditsTot").Caption = .Text1(1).Text
   DataReport1.Sections(5).Controls( _
      "lblPaysTot").Caption = .Text1(2).Text
   DataReport1.Sections(5).Controls( _
      "lblLoansTot").Caption = .Text1(3).Text
End With
```

—Erwin Cortagerena, Capital Federal, Argentina