WELCOME TO THE SIXTH EDITION OF THE VBPJ TECHNICAL TIPS SUPPLEMENT!

These tips and tricks were submitted by professional developers using Visual Basic 3.0, Visual Basic 4.0, Visual Basic 5.0, Visual Basic for Applications (VBA), and Visual Basic Script (VBS). The tips were compiled by the editors at *Visual Basic Programmer's Journal*. Special thanks to *VBPJ* Technical Review Board members Francesco Balena, Chris Kinsman, Karl E. Peterson, Phil Weber, and Jonathan Wood. Instead of typing the code published here, download the tips from the free, Registered Level of The Development Exchange at http://www.windx.com.

If you'd like to submit a tip to *Visual Basic Programmer's Journal,* please send it to User Tips, Fawcette Technical Publications, 209 Hamilton Avenue, Palo Alto, California, USA, 94301-2500. You can also fax it to 650-853-0230 or send it electronically to vbpjedit@fawcette.com or 74774.305@compuserve.com. Please include a clear explanation of what the technique does and why it is useful, and indicate whether it's for VBA, VBS, VB3, VB4 16- or 32-bit, or VB5. 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.

VB4 16/32

Level: Intermediate

USE BOOLEAN VARIABLES FOR CHECK-BOX VALUES

Visual Basic specifies a Boolean's default values as zero for False and -1 for True. You may save and set the value of a check box, based on the absolute value of a Boolean variable, as these correspond to the intrinsic constants vbUnchecked and vbChecked:

```
Dim bBool as Boolean

bBool = True

'//If bBool = 0 the checkbox will be
'//unchecked, if it is anything else it will
'//be checked.
Check1.Value = Abs(bBool)
```

—Jeremy Boschen, Branchburg, New Jersey

VB4 16/32

Level: Intermediate

DISPLAY HORIZONTAL SCROLLBAR

Unlike the Windows 95 common controls, the standard list box doesn't have a horizontal scrollbar when list items are too wide to fit within the list box. Fortunately, it's not hard to direct a list-box control to display a horizontal scrollbar.

Add this code to a form's Load event. It fills a list box with 100 long strings and calls SetHScroll to show a horizontal scrollbar in the list box:

```
Private Sub Form_Load()
    Dim i As Integer
    For i = 1 To 100
        List1.AddItem CStr(i) & _
            " bottle(s) of beer on the wall."
    Next i
    SetHScroll Me, List1, List1.List(0)
End Sub
```

Add this code, which includes the required API declarations and the SetHScroll routine, to a BAS module. The SetHScroll routine uses the SendMessage API function to send the LB_SETHORIZONTALEXTENT message to a list box. The last argument is an item from the list, preferably one of the longest items. SetHScroll determines the string's width in pixels and passes this value to the list box along with the LB_SETHORIZONTALEXTENT message. The list box sets its horizontal extent to this value, and if it is wider than the list-box control, the list box displays a horizontal scrollbar:

```
#If Win32 Then
Declare Function SendMessage Lib "user32" _
    Alias "SendMessageA" ( _
    ByVal hwnd As Long, ByVal wMsg As Long, _
    ByVal wParam As Long, lParam As Long) As Long
#Else
Declare Function SendMessage Lib "user32" _
    Alias "SendMessageA" ( _
    ByVal hwnd As Integer, ByVal wMsg As Integer, _
    ByVal wParam As Integer, lParam As Long) As Long
#End If

'Define constant for message to list-box control
Const LB_SETHORIZONTALEXTENT = &H194

Public Sub SetHScroll(Frm As Form, Ctrl As _
    Control, strText As String)
    Dim nScaleMode As Integer
    Dim nTextWidth As Integer

    'Scale in pixels for window message
    nScaleMode = Frm.ScaleMode
    Frm.ScaleMode = 3
    'Get the width, in pixels, of the text string
    nTextWidth = Frm.TextWidth(strText)
    'Send a message to the list box
    SendMessage Ctrl.hwnd, _
        LB_SETHORIZONTALEXTENT, nTextWidth, 0
    'Restore previous scale mode
    Frm.ScaleMode = nScaleMode
End Sub
```

—Peter Gomis, Shelton, Connecticut

VB4 32, VB5, VBA
Level: Intermediate

DETERMINE WHICH OPTION BUTTONS HAVE BEEN SELECTED

When attempting to determine which option buttons a user has selected from an array of option buttons, use this code instead of using an inefficient If-Then construct:

```
intSelectedItem = Option(0).Value * 0 - _
    Option(1).Value*1 - Option(2).Value * 2
```

If you have more than a few items, put this code in a loop:

```
intSelectedItems = 0
For iCount = 0 to N
'N = total number of option boxes minus one
    intSelectedItem = - _
        Option(iCount).Value * iCount
Next
```

intSelectedItem is the index of the option button that the user selected.

—Bruce H. Bitterman, San Antonio, Texas

VB4 16/32, VB5
Level: Beginning

GET THE TRUE MEMBER OF AN OPTION ARRAY

Setting an element in an array of option buttons to True is easy. Click on the option button, or use this code:

```
OptionArray(ThisOne) = True
```

ThisOne is the Index of the member you want to be selected. Getting the True member of an option array is not so simple. Because you need to perform this kind of task in almost any reasonably complex program, adding this function to your code library or generic module simplifies the task. You don't need to know the array size in advance:

```
Function OptionTrueIs(OptionArrayName As _
    Variant) As Integer
    ' Returns the index of the True
    ' member of an option array
    Dim Ctl as Control
    For Each Ctl in OptionArrayName
        If Ctl.Value = True Then
            OptionTrueIs = Ctl.Index
            Exit For
        End If
    Next
End Function
```

You can use the routine to set a Public variable from a Properties sheet using this format:

```
SomePublicVariable = OptionTrueIs(SomeOptionArray())
```

Or use this code to save a program variable between runs:

```
SaveSetting("MyApp", "OptionSettings", _
    "OptionKey", OptionTrueIs(SomeOptionArray())
```

Load the routine using this code:

```
SomeOptionArray(GetSetting("MyApp", "", _
    "OptionSettings", "OptionKey",0))=True
```

Or you can load the routine using this code:

```
SomePublicVariable = GetSetting("MyApp", "", _
    "OptionSettings", "OptionKey",0)
```

Use this code to control a Select Case structure:

```
    Select Case OptionTrueIs(SomeOptionArray())
    Case 0:
        ' This code will execute if element 0
        ' of SomeOptionArray() is selected.
    End Select
```

Use this code to test a condition:

```
    If OptionTrueIs(SomeOptionArray()) = SomeValue Then
        ' This code will execute if element
        ' SomeValue of SomeOptionArray() is
        ' selected.
    End If
```

—Roger Gilchrist, Ourimbah, New South Wales, Australia

VB4 16/32, VB5
Level: Intermediate

DEBUG ERROR HANDLERS

Visual Basic gives you the option to turn off error trapping in your projects. This is useful when you suspect that your error handlers contain errors themselves. To turn off error handling globally, choose Options from the Tools menu. Select the General tab, and select Break on All Errors. The next time you run your project in the development environment, Visual Basic will break whenever an error occurs in your code, whether the error is trapped or not.

—Jeffrey P. McManus, San Francisco, California

VB3, VB4 16/32, VB5
Level: Beginning

PROTECT DATA WITHIN MODULES

Use Private variables in BAS modules to protect data that the module's routines must access, but which should be hidden from the rest of the application:

```
Dim hidden_data As Integer

Function GetData() As Integer
    GetData = hidden_data * 2
End Function
```

—Rod Stephens, Boulder, Colorado

VB3, VB4 16/32, VB5, VBA
Level: Intermediate

CONVERT A TEXT FILE INTO ACCESS MDB

It can be troublesome to convert a text file into an Access database. It takes a lot of time to open the file for sequential access and create new records using the AddNew method. Instead, use the Text ISAM driver and SQL to do the job for you. First, create a SCHEMA.INI file for the text file and place it in the same directory as the text file. Use this code to convert the database:

```
Dim db As Database, tbl as TableDef
Set db = DBEngine.CreateDatabase(App.Path & _
    "\mymdb.mdb", dbLangGeneral, dbVersion_0)
Set tbl = db.CreateTableDef("Temp")
tbl.Connect = "Text;database=c:\vbpj\data"
tbl.SourceTableName = "Customer#txt"
db.TableDefs.Append tbl
db.Execute "Select Temp.* into NewTable from Temp"
db.TableDefs.Delete tbl.Name
db.Close
Set tbl = Nothing
Set db = Nothing
```

Now, you only need to create indexes. You can use this method to convert text files in excess of 100,000 records in a few seconds.

—Richard Mageau, Warren, Rhode Island

VB3, VB4 16/32, VB5, VBA
Level: Intermediate

QUICK AND EASY SCHEMA.INI

To create a SCHEMA.INI file for a text file quickly and easily, let the ODBC setup and drivers do it for you. Go to the Control Panel and start ODBC. Click on the Add button and select the Text driver. Click on the Options button and describe how to arrange the text file. If your text file has a header record, click on the Guess button, and the ColumnNames will be filled out for you. On the last screen, choose Cancel to avoid setting up the data source. Check the directory where the text file resides, and you'll find a new SCHEMA.INI file.

—Richard Mageau, Warren, Rhode Island

VB4 16/32, VB5
Level: Intermediate

CONTROL ARRAYS ARE COLLECTIONS

Most VB4 programmers know they can access a form's control collection using a For Each loop. Many, however, are not aware that individual control arrays are actually collections. This knowledge can be helpful when checking the properties of several controls, especially when new controls are loaded at run time. You might wish to create a form in which all text boxes must be filled out. Instead of assigning a unique name to each text box, create a control array with each text box named txtFields. You can use this name as you use the name of any collection in a For Each loop:

```
Dim v
For Each v In txtFields
    If v.Text = "" Then
        MsgBox "All fields must be " & _
            "filled out.", vbExclamation + _
            vbOKOnly
        Exit Sub
    End If
Next
```

Because txtFields is a collection, you can also determine how many text boxes are in the array by checking the value of txtFields.Count.

For more information, see Karl E. Peterson's Q&A column, "The Keys to Your Controls" [*VBPJ* August 1997].

—Daniel Buskirk, The Bronx, New York

VB3, VB4 16/32, VB5, VBA
Level: Beginning

SET A FLAG TO TRUE/FALSE

When you want to set a flag to True or False, depending on the results of a comparison, you don't need to use an If statement like this:

```
If x > 3 then
    Greater = True
Else
    Greater = False
End If
```

Logical operators return a value like any other operator. Instead, use this syntax:

```
Greater = x > 3
```

This way, you move directly to the result of the comparison in your flag. It is shorter and faster than the form example.

—Martin Girard, Chicoutimi, Quebec, Canada

VB3, VB4 16/32,VB5, VBA
Level: Intermediate

SUPPORT A FONT THE PRINTER DOESN'T

Often users print from applications that support a font the printer doesn't. An error occurs if an application uses a statement such as Printer.Print SomethingToPrint following a Printer.FontName = FontName statement, and the printer doesn't support the FontName.

You can use this function procedure as a corrective measure after the error is trapped. This function procedure uses only one parameter—the font names specified by the application. If the printer doesn't support the font, the function procedure returns a font on the printer most similar to the font in the application:

```
Function FindFont (SpecifiedFont As String)
'find the font on the printer which is the
'closest to the font name passed to this procedure:
Dim i%, tempfont%
For i% = 1 To Printer.FontCount
    tempfont = InStr(Printer.Fonts(i%), SpecifiedFont)
        'Get font's position.
    If tempfont <> 0 Then
        Exit For
```

```
    End If
Next
If tempfont <> 0 Then
    FindFont = Printer.Fonts(i%)
Else
    'take an action if no similar fonts are
    'found on this printer
End If
End Function
```

In your error handler, use this function in conjunction with the FontName property of the Printer object. After you take this corrective measure, resume the execution of your program:

```
Printer.FontName = FindFont(SpecifiedFont)
Resume Next
```

—Brian Hunter, Brooklyn, New York

VB4 32, VB5, VBA

Level: Intermediate

CENTER FORMS WITH TASKBAR VISIBLE

Just about every VB developer uses the Move (Screen.Width - Width) \ 2, (Screen.Height - Height) \ 2 method to center the forms on screen. However, when the user has the Windows 95 or NT 4.0 taskbar visible, your form centers on screen but doesn't take into account the position of the taskbar itself. The CenterForm32 routine centers a form in available screen area, taking into account the taskbar. Add this code to the Declarations section of a module, and put the code CenterForm32 Me on the Form_Load event of the forms you want to center:

```
Option Explicit
Private Const SPI_GETWORKAREA = 48
Private Declare Function _
    SystemParametersInfo& Lib "User32" _
    Alias "SystemParametersInfoA" ( _
    ByVal uAction As Long, _
    ByVal uParam As Long, lpvParam As Any, _
    ByVal fuWinIni As Long)
Private Type RECT
    Left As Long
    Top As Long
    Right As Long
    Bottom As Long
End Type

Public Function CenterForm32 (frm As Form)
    Dim ScreenWidth&, ScreenHeight&, _
        ScreenLeft&, ScreenTop&
    Dim DesktopArea As RECT
    Call SystemParametersInfo ( _
        SPI_GETWORKAREA, 0, DesktopArea, 0)
    ScreenHeight = (DesktopArea.Bottom - _
        DesktopArea.Top) * Screen.TwipsPerPixelY
    ScreenWidth = (DesktopArea.Right - _
        DesktopArea.Left) * Screen.TwipsPerPixelX
    ScreenLeft = DesktopArea.Left * Screen.TwipsPerPixelX
    ScreenTop = DesktopArea.Top * Screen.TwipsPerPixelY
    frm.Move (ScreenWidth - frm.Width)  _
        \ 2 + ScreenLeft, (ScreenHeight - _
        frm.Height) \ 2 + ScreenTop
End Function
```

—Miguel Santos, Aveiro, Portugal

VB3, VB4 16/32

Level: Intermediate

DISPLAY TOOLTIPS

You can easily duplicate the tooltips that appear when you float the mouse over a button in Microsoft's products. This code displays the tooltip in the lower right-hand corner of the control (normally a button):

```
' xTip: string to display as the tip
' xCtrl: control you want the tip to show for.
' xTipControl: control being used to display
' the tip (an SSPanel)
Public Sub ShowTip(xTip As String, xCtrl As _
    Control, xTipControl As Control)
    xTipControl.Left = xCtrl.Left + xCtrl.Width
    xTipControl.Top = xCtrl.Top + xCtrl.Height
    xTipControl.Caption = xTip
    xTipControl.Visible = True
End Sub
```

This code hides the tooltip:

```
' xTipControl : control which is being used to
' display the tip.
Public Sub HideTip(xTipControl As Control)
    xTipControl.Visible = False
End Sub
```

Place the ShowTip function in the MouseMove event of the control(s) for which you want to display a tip. Place the HideTip function in the other control(s) for which you don't want to display a tip:

```
Sub Command1_MouseMove (Button As Integer, _
    Shift As Integer, X As Single, Y As Single)
    ShowTip "Command1", Command1, SSPanel1
End Sub

Sub Command2_MouseMove (Button As Integer, _
    Shift As Integer, X As Single, Y As Single)
    ShowTip "Command2", Command2, SSPanel1
End Sub

Sub Form_MouseMove (Button As Integer, _
    Shift As Integer, X As Single, Y As Single)
'No tooltip for form
    HideTip Text1
End Sub
```

—Andy McInturff, Erwin, Tennessee

VB5

Level: Intermediate

USE SUBSCRIPTS AND SUPERSCRIPTS

You can use subscripts and superscripts in a RichTextBox control. To prevent the control from truncating text that is raised or lowered too far, reduce the characters' size accordingly:

```
Private Sub Form_Load()
    RichTextBox1.Font.Name = "Times New Roman"
    RichTextBox1.Font.Size = 10
    RichTextBox1.Text = "H2SO4"

' Move the numbers down 2 points.
```

```
    MakeSubscript RichTextBox1, 1, 1, -2
    MakeSubscript RichTextBox1, 4, 1, -2
End Sub

Private Sub OffsetRichText(box As _
    RichTextBox, start As Integer, _
    length As Integer, offset As Integer)
        box.SelStart = start
        box.SelLength = length
        box.SelFontSize = box.Font.Size + offset
        box.SelCharOffset = -ScaleY(offset, _
            vbPoints, vbTwips)
        box.SelStart = 0
        box.SelLength = 0
End Sub
```

—Rod Stephens, Boulder, Colorado

VB4 32

Level: Advanced

PLAY WAV SOUNDS FROM RESOURCE FILES

Everyone loves a good sound bite. You can use the PlaySound() function in the Windows Multimedia DLL (WINMM.DLL) to play a wave (WAV) file that is stored as a resource. Although you can do this using the sndPlaySound function, sndPlaySound requires you to find, load, lock, unlock, and free the resource. PlaySound achieves all of this with a single line of code.

Add this line of code to your resource file (RC) designating a name for the sound and its location:

```
MySound WAVE c:\music\vanhalen.wav
```

The name MySound is a placeholder for a name you supply to refer to this WAV resource in code.

Compile the resource file using the resource compiler (RC.EXE) to create a RES file. Include this file in your VB project. Use this code to play the sound:

```
Private Declare Function PlaySound Lib _
    "winmm.dll" Alias "PlaySoundA" ( _
    ByVal lpszName As String, _
    ByVal hModule As Long, _
    ByVal dwFlags As Long) As Long

Private Const SND_ASYNC& = &H1
' Play asynchronously
Private Const SND_NODEFAULT& = &H2
' Silence if sound not found
Private Const SND_RESOURCE& = &H40004
' Name is resource name or atom

Dim hInst As Long
' Handle to Application Instance
Dim sSoundName As String
' String to hold sound resource name
Dim lFlags As Long
' PlaySound() flags
Dim lRet As Long
' Return value

Private Sub Command1_Click()
    hInst = App.hInstance
    sSoundName = "MySound"
    lFlags = SND_RESOURCE + SND_ASYNC + _
        SND_NODEFAULT
    lRet = PlaySound(sSoundName, hInst, lFlags)
End Sub
```

Note that you must compile this application and run the resulting EXE directly for this code to work because it relies on the application instance handle. While in the VB development environment, App.Instance returns an instance handle to the running VB32.EXE process, not the application under development. For a complete description of PlaySound(), refer to the Multimedia section of the Win32 SDK.

—Steve Davis, Gaithersburg, Maryland

VB3, VB4 16/32, VB5

Level: Intermediate

A BETTER WAY TO SWAP TWO INTEGER VARIABLES

The algorithm shown in the tip, "Swap Values Using Only Two Variables" ["101 Tech Tips for VB Developers, Supplement to the February 1997 issue of *VBPJ*, page 25], crashes not only if (a+b) >= 32K, but if (a+b) < -32K also. I usually use this algorithm instead:

```
a = a Xor b: b = a Xor b: a = a Xor b.
```

This works for almost any programming language.

—Alex Bootman, Foster City, California

VB4 32, VB5

Level: Intermediate

WRITE LESS CPU-BOUND ANIMATIONS

When doing animation, such as scrolling a label or using picture boxes, I first used the method described by Eric Bernatchez ("Smoother Control Animation," "101 Tech Tips for VB Developers," Supplement to the February 1997 issue of *VBPJ*, page 21). However, I found that while in the Do Loop, the CPU usage is 100 percent! Windows NT, Windows 95, and Win32 have an API call named SLEEP. This call suspends the called application for the supplied amount of milliseconds. When you change the code, the CPU usage on a Pentium 100 drops to 10 percent:

```
Declare Sub Sleep Lib "kernel32" ( _
    ByVal dwMilliseconds As Long)

Public Sub Scrolling()
    Label1.Left = Me.Width
    Do
        Sleep 100
        Label1.Left = Label1.Left - 60
        DoEvents
    Loop Until Label1.Left <= -(Label1.Width + 15)
End Sub
```

The only problem with this method is that your app won't respond to user input while it sleeps, so don't let it sleep too long.

—Derek Robinson, Pretoria, South Africa

VB4 16/32, VB5, VBA

Level: Intermediate

EXPORT DATA TO ISAM DATABASES

Use this code to get data out to different formats without requiring a complete copy of the DBMS on the user machine. Create a new project with a command button, and copy this code into the button's Click event:

```
Dim db As Database
Set db = Workspaces(0).OpenDatabase _
    (App.Path & "\biblio.mdb")

' commented out syntax line followed by
' working example
'    db.Execute "SELECT tbl.fields INTO
'        [dbms type; DATABASE=path].[unqualified
'        file name] FROM [table or tables]"

db.Execute "Select * into [dBASE III;
    DATABASE=" & "C:\My Documents].
    [testa] FROM [authors]
```

By using the brackets and the dot operator, you get a completely proper output in the ISAM database type of your choice. Also, if you choose Text as the database type, the statement creates a SCHEMA.INI for you automatically, or adds a new section for your new data file to a SCHEMA.INI already in the path folder.

You can do any kind of export the client wants without using a DBMS on the machine. This makes your life easier when you notice that some users running Word 97 have problems mail-merging with text files you originally created with Write# and Print# methods.

—Robert Smith, San Francisco, California

VB3, VB4 16/32, VB5

Level: Intermediate

RANDOM VALUES WITHOUT DUPLICATES

This code produces a "random" sequence of integers between specified Lower and Upper values without duplication. On the first call to the Shuffle routine with new values for Upper and Lower, Shuffle performs an efficient prime factorization, computes the constants for a linear sequence generator, and uses Randomize to generate a seed value. Each subsequent call with the same value of Upper and Lower uses this generator to return the next "random" number in the sequence. The sequence does not repeat until all possible values have been returned. This code fills the array with 20 random numbers between 1 and 100, without duplication:

```
For I = 1 To 20
    num(I)=Shuffle(1,100)
next I
```

Here is the Shuffle routine:

```
Static Function Shuffle (Lower As Integer, _
    Upper As Integer) As Integer
Static PrimeFactor(10) As Integer
Static a As Integer, c As Integer, b As Integer
Static s As Integer, n As Integer
Dim i As Integer, j As Integer, k As Integer
Dim m As Integer

If (n <> Upper - Lower + 1) Then
    n = Upper - Lower + 1
    i = 0
    n1 = n
    k = 2
    Do While k <= n1
        If (n1 Mod k = 0) Then
            If (i = 0 Or PrimeFactor(i) <> k) Then
                i = i + 1
                PrimeFactor(i) = k
            End If
            n1 = n1 / k
        Else
            k = k + 1
        End If
    Loop

    b = 1
    For j = 1 To I
        b = b * PrimeFactor(j)
    Next j
    If n Mod 4 = 0 Then b = b * 2
    a = b + 1
    c = Int(n * .66)
    t = True
    Do While t
        t = False
        For j = 1 To I
            If ((c Mod PrimeFactor(j) = 0) Or _
                (c Mod a = 0)) Then t = True
        Next j
        If t Then c = c - 1
    Loop
    Randomize
    s = Rnd(n)
End If

s = (a * s + c) Mod n
Shuffle = s + Lower

End Function
```

—John W. Starner, El Paso, Texas

VB4 16/32, VB5

Level: Intermediate

COMMENT MULTIPLE LINES

VB provides only single-line comment functions: "REM" and "'". If you're looking for another way to comment out a block of code, use conditional compilation in VB4 instead. Define "COMMENT = 0" in the Conditional Compilation Arguments field on the Advanced tab in the Options dialog of the Tools menu:

```
Public Sub TestingSub()
    Print "This subroutine is used to"
    Print "demonstrate block"
    Print "commenting in VB."
#If COMMENT then
    Print "This line will not be printed."
    Print "Since this is commented out."
    Print "VB ignores these lines during compilation."
#End If
End Sub
```

This trick also works with VB5, but you might find the Comment Block command on the Edit toolbar much handier.

—Frewin Chan, Scarborough, Ontario, Canada

VB4 32, VB5

Level: Advanced

CHANGE DISPLAY SETTINGS ON THE FLY

When writing a game for Windows 95, set the display resolution to 640-by-480, set the color palette to True Color when it runs, and restore it to its initial mode when it ends. Use this function to implement it:

```
'– Declares
Private Declare Function lstrcpy _
     Lib "kernel32" Alias "lstrcpyA" _
     (lpString1 As Any, lpString2 As Any) _
     As Long
Const CCHDEVICENAME = 32
Const CCHFORMNAME = 32
Private Type DEVMODE
     dmDeviceName As String * CCHDEVICENAME
     dmSpecVersion As Integer
     dmDriverVersion As Integer
     dmSize As Integer
     dmDriverExtra As Integer
     dmFields As Long
     dmOrientation As Integer
     dmPaperSize As Integer
     dmPaperLength As Integer
     dmPaperWidth As Integer
     dmScale As Integer
     dmCopies As Integer
     dmDefaultSource As Integer
     dmPrintQuality As Integer
     dmColor As Integer
     dmDuplex As Integer
     dmYResolution As Integer
     dmTTOption As Integer
     dmCollate As Integer
     dmFormName As String * CCHFORMNAME
     dmUnusedPadding As Integer
     dmBitsPerPel As Integer
     dmPelsWidth As Long
     dmPelsHeight As Long
     dmDisplayFlags As Long
     dmDisplayFrequency As Long
End Type
Private Declare Function _
     ChangeDisplaySettings Lib _
     "User32" Alias "ChangeDisplaySettingsA" (_
     ByVal lpDevMode As Long, _
     ByVal dwflags As Long) As Long
'– code
' Here is the function that sets the display
' mode. Width is the width of screen, Height
' is the height of screen, Color is the number
' of bits per pixel. Set the Color value to -1
' if you only want to change the screen
' resolution.
Public Function SetDisplayMode(Width As _
     Integer,Height As Integer, Color As _
     Integer) As Long
Const DM_PELSWIDTH = &H80000
Const DM_PELSHEIGHT = &H100000
Const DM_BITSPERPEL = &H40000
Dim NewDevMode As DEVMODE
Dim pDevmode As Long
With NewDevMode
     .dmSize = 122
     If Color = -1 Then
          .dmFields = DM_PELSWIDTH Or DM_PELSHEIGHT
     Else
          .dmFields = DM_PELSWIDTH Or _
               DM_PELSHEIGHT Or DM_BITSPERPEL
     End If
     .dmPelsWidth = Width
     .dmPelsHeight = Height

     If Color <> -1 Then
          .dmBitsPerPel = Color
     End If
End With
pDevmode = lstrcpy(NewDevMode, NewDevMode)
SetDisplayMode = ChangeDisplaySettings(pDevmode, 0)
End Function
```

You can change the display mode easily with this function. For example, write this code that changes the resolution to 640-by-480 and the color palette to 24-bit True Color:

```
i = SetDisplayMode(640, 480, 24)
```

If the function is successful, it returns zero.

—Huang Xiongbai, Shanghai, China

VB5

Level: Intermediate

CREATE A NEW PROJECT TEMPLATE

Upon initialization, the VB5 IDE presents you with a list of project types, such as Standard EXE or ActiveX EXE. You can also create new project types yourself by creating templates and saving them to the Projects folder in your Templates directory. To find the Templates directory on your system, look in Options under the Tools menu, then select the Environment tab. The Templates directory is listed at the bottom of the dialog. After you create a template, its icon appears with the predefined project types whenever you start VB or use the FileNew command.

What *is* a template anyway? It's simply a project you save to the Projects folder. It includes all forms, controls, classes, code, and other elements in the project. It serves as a foundation for a new project and can have as much or as little prebuilt content as you need.

—Ron Schwarz, Mt. Pleasant, Michigan

VB3, VB4 16/32, VB5

Level: Beginning

SELECT TEXT-BOX CONTENTS ON ENTRY AUTOMATICALLY

Many commercial apps select the contents of the text box automatically whenever it gets focus. You can do the same in your apps with a few VB statements. Invoke the SelStart and SelLength functions within a TextBox control's GotFocus event:

```
Private Sub Text1_GotFocus()
     Text1.SelStart = 0
     Text1.SelLength = Len(Text1)
End Sub
```

Or place the code in a public Sub procedure and call it from any text box in your project:

```
Private Sub Text2_GotFocus()
    PreSel Text2
End Sub
Public Sub PreSel(txt As Control)
    If TypeOf txt Is TextBox Then
        txt.SelStart = 0
        txt.SelLength = Len(txt)
    End If
End Sub
```

Because you can call the PreSel procedure from anywhere, you should limit the procedure to acting on TextBox controls to prevent errors. The TypeOf function tests if you've passed the procedure a TextBox control, and if not, it skips over the selection code.

—Ron Schwarz, Mt. Pleasant, Michigan

VB3, VB4 16/32, VB5

Level: Intermediate

CLOSE FORMS UNIFORMLY

To close forms uniformly, treat the system menu's Close command and your File menu's Exit command in the same manner. In your Exit command, simply unload the form. Let the form's QueryUnload event handler see if it is safe to exit. If not, cancel the Exit by setting Cancel to True:

```
Private Sub mnuFileExit_Click()
    Unload Me
End Sub
Private Sub Form_QueryUnload(Cancel _
    As Integer, UnloadMode As Integer)
    Cancel = (MsgBox("Are you sure you " & _
        "want to exit?", vbYesNo) = vbNo)
End Sub
```

—Rod Stephens, Boulder, Colorado

VB3, VB4 16/32, VB5

Level: Beginning

RIGHT-JUSTIFY OR LEFT-JUSTIFY TEXT

Use the Format$ function to produce right- or left-justified text:

```
Format$(123, "@@@@@@")  gives  "   123"
Format$(123, "!@@@@@@") gives  "123 "
```

—Rod Stephens, Boulder, Colorado

VB4 16/32, VB5

Level: Beginning

LEARN CONTROL ARRAY BOUNDS

You can use a control array's UBound and LBound properties to determine how many controls are loaded. This code fills a string with the captions in the Label1 control array:

```
For i = Label1.LBound To Label1.UBound
    txt = txt & Label1(i).Caption & ", "
Next i
```

—Rod Stephens, Boulder, Colorado

VB3, VB4 16/32, VB5

Level: Beginning

PERFORM ONE-TIME INITIALIZATION

You can use static variables to perform one-time initialization:

```
Private Sub GetValue()
Static initialized As Boolean
    If Not initialized Then
        ' Perform one-time initialization.
            :
        initialized = True
    End If
    ' Do other stuff.
        :
End Sub
```

—Rod Stephens, Boulder, Colorado

VB4 16/32, VB5

Level: Intermediate

CREATE GLOBAL PROPERTIES

You can create property procedures in a BAS module. The rest of your program treats the "property" like any other variable, but the property procedures can perform error checking and one-time initialization.

—Rod Stephens, Boulder, Colorado

VB4 16/32, VB5

Level: Intermediate

USE EXPLICIT OBJECT TYPES FOR SPEED

It is much faster to use explicit object data types instead of generic ones. In this test that simply sets an object's public variable, the generic object takes more than 150 times longer than the specific object:

```
Private Sub Command1_Click()
Dim specific As MyClass
Dim generic As Object
Dim start_time As Single
Dim stop_time As Single
Dim i As Long
    Set specific = New MyClass
    Set generic = New MyClass
    start_time = Timer
    For i = 1 To 100000
        specific.value = i
    Next i
    stop_time = Timer
    MsgBox Format$(stop_time - start_time)
    start_time = Timer
    For i = 1 To 100000
        generic.value = i
    Next i
    stop_time = Timer
    MsgBox Format$(stop_time - start_time)
End Sub
```

—Rod Stephens, Boulder, Colorado

VB3, VB4 16/32, VB5
Level: Advanced

MAKE A FORM STAY ON TOP
You can make a form always remain above others, even when it does not have focus. Use this code in a BAS module:

```
#If Win16 Then
     Declare Sub SetWindowPos Lib "User" ( _
          ByVal hWnd As Integer, _
          ByVal hWndInsertAfter As Integer, _
          ByVal X As Integer, ByVal Y As Integer, _
          ByVal cx As Integer, ByVal cy As Integer, _
          ByVal wFlags As Integer)
#Else
     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
#End If
Public Const SWP_NOMOVE = &H2
Public Const SWP_NOSIZE = &H1
Public Const HWND_TOPMOST = -1
Public Const HWND_NOTOPMOST = -2
```

Use this code to make the form stay on top in the form module:

```
SetWindowPos hwnd, HWND_TOPMOST, 0, 0, 0, 0, _
     SWP_NOMOVE + SWP_NOSIZE
```

Use this code to make the form no longer stay on top in the form module:

```
SetWindowPos Parent.hwnd, HWND_NOTOPMOST, _
      0, 0, 0, 0, SWP_NOMOVE + SWP_NOSIZE
```

In 16-bit systems, use the 16-bit equivalents.

—Rod Stephens, Boulder, Colorado

VB3, VB4 16/32, VB5
Level: Intermediate

SET PRINTER MARGINS
You can use the Printer's scale properties to set margins. Set the properties so the position (0, 0) is mapped to where you want the margins. This code makes the left margin 0.5 inch and the top margin 0.75 inch. The factor of 1440 converts inches into twips:

```
Printer.ScaleLeft = -0.75 * 1440
Printer.ScaleTop = -0.5 * 1440
Printer.CurrentX = 0
Printer.CurrentY = 0
```

—Rod Stephens, Boulder, Colorado

VB3, VB4 16/32, VB5
Level: Advanced

MOVE A FORM WITHOUT A TITLE BAR
Enter this code in a BAS module to let users move a window without a title bar:

```
Declare Function ReleaseCapture Lib "user32" () As Long
Declare Function SendMessage Lib "user32" _
     Alias "SendMessageA" ( _
     ByVal hwnd As Long, ByVal wMsg As Long, _
     ByVal wParam As Long, lParam As Any) As Long
Public Const HTCAPTION = 2
Public Const WM_NCLBUTTONDOWN = &HA1
```

To give users a way to begin the move, place a little colored PictureBox in one corner of the form and let its MouseDown event begin the process:

```
Private Sub Picture1_MouseDown(Button As _
     Integer, Shift As Integer, X As Single, Y As Single)
     ReleaseCapture
     SendMessage hwnd, WM_NCLBUTTONDOWN, HTCAPTION, 0&
End Sub
```

—Rod Stephens, Boulder, Colorado

VB5
Level: Beginning

ENUMS IMPROVE READABILITY
An enumerated type is simply a list of constants, each of which has a unique value. An Enum can improve the readability of your code by allowing you to declare a variable of your enumerated type, then assign one of the elements of the Enum to the value of the variable. Visual Basic makes this easy by providing you with a list of values for your variable as you use it:

```
Public Enum TimeOfDay
     Morning = 0
     Afternoon = 1
     Evening = 2
End Enum
Sub Main()
     Dim RightNow As TimeOfDay
     If Time >= #12:00:00 AM# And Time < #12:00:00 PM# Then
          RightNow = Morning
     ElseIf Time >= #12:00:00 PM# And Time < #6:00:00 PM# Then
          RightNow = Afternoon
     ElseIf Time >= #6:00:00 PM# Then
          RightNow = Evening
     End If
End Sub
```

Validity checking is not done when assigning values to the special-typed variable. Even though the TimeOfDay values range from zero to two, VB doesn't attempt to keep you from assigning a value of four, or another number, to such a variable. Be especially aware of this when writing Property Let procedures, which accept an enumerated parameter.

—Charles Caison, Greensboro, North Carolina

VB3, VB4 16/32, VB5, VBA, VBS
Level: Intermediate

IMPROVE PERFORMANCE BY ELIMINATING SUBEXPRESSIONS

Arrays can reduce the amount of code you must write to perform an operation, especially when used with a loop. By using the same array element in multiple places in your code, you can increase the speed with which your code executes if you resolve the array only once. This code determines if the value of the array element is *this value*, or if the value of the array element is *that value*:

```
If intArray(intArrayIndex) = 5 or _
    intArray(intArrayIndex) = 10 Then
```

Because Visual Basic must calculate the location of the array element to resolve its value (*base + intArrayIndex * sizeof_intArray_type*), it makes sense that the code executes faster the less the calculation is performed. This code provides a faster implementation:

```
TheValue = intArray(intArrayIndex)
If TheValue = 5 or TheValue = 10 Then
```

This technique offers performance gains that are even more dramatic when used in a loop:

```
For intArrayIndex = 0 to 10
TheValue = intArray(intArrayIndex)
If TheValue = 5 or TheValue = 10 Then
    ' Do something here
End If
Next
```

By eliminating the redundant subexpression and assigning the value to the variable named "TheValue" only once per loop (executes 10 times) instead of twice per loop (executes 20 times), you significantly reduce the amount of work that VB must perform to complete this loop.

—Charles Caison, Greensboro, North Carolina

VB3, VB4 16/32, VB5, VBA
Level: Intermediate

SIMPLIFY VARIABLE DECLARATIONS BY USING DEFAULT TYPES

Hungarian notation and similar naming schemes tell you at a glance the data type of any variable in your code. You know that the variable intCounter is an integer, and you know that strName is a string.

You can simplify your variable declarations by telling VB that any variable you declare that begins with a certain letter will always be a certain type. This relieves you of having to include the AS DataType clause in your variable declarations, but you can still take advantage of the benefits of explicit variable type declaration. This code informs VB that all variables beginning with the letter "i" will be integers:

```
DefInt i
Dim iCounter
```

If you use the Visual Basic function TypeName() and pass it the variable iCounter, it returns "Integer" rather than "Variant." In addition to specifying a specific character, you can also specify a range or list of letters. This code assigns variables that begin with the letters i, j, or k as integers, and those variables that begin with s or c as strings:

```
DefStr s, c
DefInt i-k

Dim iCounter ' an integer
Dim sName    ' a string
```

Here are more default variable type declarations:

DefBool (Boolean)
DefByte (Byte)
DefLng (Long)
DefCur (Currency)
DefSng (Single)
DefDbl (Double)
DefDec (Decimal)
DefDate (Date)
DefStr (String)
DefObj (Object)
DefVar (Variant)

—Charles Caison, Greensboro, North Carolina

VB3, VB4 16/32, VB5, VBA, VBS
Level: Intermediate

MAINTAIN USERNAME AND PASSWORD

Due to a minor bug, the Internet Transfer Control cannot use the username and password properties if they are set before the URL property. Because of this, you should always set the URL property first. This code will fail:

```
Inet1.Password = "Chicken_Feet"
Inet1.UserName = "JohnnyW"
Inet1.URL = FTP://ftp.32X.com
Inet1.Text = Inet1.OpenURL
```

This code, however, should work properly:

```
Inet1.URL = FTP://ftp.32X.com
Inet1.Password = "Chicken_Feet"
Inet1.UserName = "JohnnyW"
Inet1.Text = Inet1.OpenURL
```

The first section of code fails because the URL is set after the username and password are set. The second section of code works because the URL is set first.

—Charles Casion, Greensboro, North Carolina

VB3, VB4 16/32, VB5
Level: Beginning

VB VERSION ISDATE DIFFERENCES

Under VB3, IsDate returns True only if the string corresponds to the date format indicated by the "Regional" or "International" settings in the Control Panel. Under VB4 and VB5, however, IsDate returns True if the date is a valid date in any format (mm/dd/yy or dd/mm/yy).

There are two morals to this story. First, if your VB3 apps claim that dates are invalid when they look correct, check the Control Panel settings. Second, if you need to enforce a particular date format in your VB4 and VB5 programs, you need to take steps beyond using IsDate. For example, you can use masked-edit controls or extra code to validate date formats.

—Jon Pulsipher, Plano, Texas

VB3, VB4 16/32, VB5

Level: Intermediate

Simulate Win95 Taskbar Within Your App

The Windows 95 taskbar (with the Auto Hide check box set) is quite attractive and provides a compact, nonintrusive way to access and organize information. You can easily simulate this functionality in a VB program and provide a useful alternative to a pull-down menu and/or SpeedButton. To set up such a "taskbar" in your VB application, include a control that can act as a container for other controls in your VB form. The PictureBox control looks good and works well; or you can use a Panel or a Frame.

Set the PictureBox's Align and AutoSize properties. The Align property indicates where the taskbar will reside on the form, and the AutoSize property ensures that the PictureBox is sized to fit the contours of its parent form whenever the form is resized.

Set the PictureBox's Visible property to False. Make sure your custom taskbar is not visible until the mouse is moved over a defined "hot" zone on the form. A "hot" zone is a location on the screen defined in X or Y coordinates. You can use these coordinates in any unit of measurement, such as twips or pixels. Set the Parent form's ScaleMode property to the unit of measurement you wish to work in.

Set the Width and/or Height of the PictureBox to appropriately show embedded controls or information. Set up code in the Form's MouseMove event to handle the taskbar. This code aligns the taskbar to the left of the screen:

```
'Object       Property      Setting
'------       --------      -------
'Form         Name          FrmTaskBarEx
'             ScaleMode     Pixel

'PictureBox   Name          PicTaskBar
'             Align         Align Left
'             AutoSize      True
'             Visible       False

Private Sub Form_Load()
' Set Width of PictureBox to 1/4 size of Parent form
' when form is loaded. This is wide enough to show
' all of the controls I embedded in the PictureBox,
' and will vary with each application.
    PicTaskBar.Width = Me.ScaleWidth / 4
End Sub

Private Sub Form_MouseMove(Button As _
    Integer, Shift As Integer, X As Single, Y As Single)
' When the horizontal mouse position is less
' than or equal to defined hot zone
' (in this case, horizontal pixel position 4), show taskbar
    If (X <= 4) Then
        PicTaskBar.Visible = True
        ' When the horizontal mouse position is
        ' greater than the width of the
        ' PictureBox, hide taskbar
    ElseIf (X > PicTaskBar.Width) Then
        PicTaskBar.Visible = False
    End If
End Sub
```

—Pete Rodriguez, Jersey City, New Jersey

VB4 16/32, VB5

Level: Beginning

Let the Editor Check Your Spelling

To make certain that you declared your variables, place Option Explicit in your Declarations section. Using a mixture of upper and lower case in variable names, you can easily verify that you don't misspell variable names as you type them. By entering the names in all lower or upper case, the editor changes the variables' appearance automatically to match the declared syntax:

```
Public strFetchThis as String
Public iCountMe as Integer
```

When you declare these variables, typing STRFETCHTHIS or icountme results in the editor's changing them to the declared appearance when you move on.

—Randall Arnold, Coppell, Texas

VB3, VB4 16/32, VB5, VBA

Level: Intermediate

Determine Next/Previous Weekday

This function returns week identifier information, such as Week Beginning/Ending date, and makes it more generic to return a date of any previous or following weekday. You can use this function in VB5 and VBA5; with minor modifications to the optional arguments, you can also use it in VB3 and VB4:

```
Public Function SpecificWeekday(d As Date, _
    Optional ByVal WhatDay As Byte = _
    vbSaturday, Optional ByVal GetNext _
    As Boolean = True) As Date
    'Purpose: Returns a date of the
    'Next/Previous specific weekday for a given date
    ' Input:     d - Date for which next/previous
    '            weekday is needed
    '            WhatDay - optional (byte)
    '            argument (default = vbSaturday)
    '            specifying what weekday to find
    '            GetNext - optional (Boolean)
    '            argument: True for
    '            Next(default), False for Previous
    'Output:    Date of the next/previous specific weekday
    'Note:      If both optional arguments are
    '            omitted, returns Week-Ending Date
    '            If passed date (d) falls on the
    '            same weekday as WhatDay, the
    '            function returns the passed date
    '            unchanged.
    Dim iAdd As Integer
    iAdd = (WhatDay - WeekDay(d))
    If GetNext Then
        SpecificWeekday = DateAdd("d", IIf(_
            iAdd >= 0, iAdd, 7 + iAdd), d)
```

```
    Else
        SpecificWeekday = DateAdd("d", IIf(_
            iAdd <= 0, iAdd, iAdd - 7), d)
    End If
End Function
```

—Serge Natanov, received by e-mail

VB3, VB4, VB5, VBA, VBS

Level: Intermediate

FIND TROUBLE SPOTS

Implement a low-tech trace of your application by writing a message whenever your code enters and exits a subroutine. This can help you locate elusive bugs, because you can turn on the trace and run your application until failure. The last entry in your log indicates what subroutine you were executing when the failure occurred. Enter this code in the General Declarations section of a module:

```
Public DebugMode as Boolean
```

Normally, you should set this variable to False. During debugging, however, set it to True (DebugMode = True). Place this sub in the module:

```
Public Sub WriteDebugMessage(ByVal Msg as String)
    If DebugMode then
        'write a message, including the system time
        End If
    End Sub
```

Enter this code as the first statement in every subroutine and function in your project:

```
Call WriteDebugMessage("Entering NameOfSubroutine")
```

Enter this code as the last statement in every subroutine and function:

```
Call WriteDebugMessage("Exiting NameOfSubroutine")
```

VB4 introduced conditional compilation, which offers a simpler way to implement this scheme. By defining a conditional compiler constant rather than a global flag variable, you can optimize your final EXE by toggling the constant to False. Conditional calls to the WriteDebugMessage routine won't be compiled into your shipping build:

```
#Const DebugMode = False
' for final code, true to test

#If DebugMode Then
    Call WriteDebugMessage("Important Info")
#End If
```

Use this code to write other debug information to the log, such as variable values. Don't forget to turn off Debug mode when you're not using it to avoid writing massive amounts of unneeded data to the log.

—Shannon Huggins, Lancaster, South Carolina

VB4 16/32, VB5

Level: Beginning

USE ARRAYS, NOT COLLECTIONS

Arrays take less memory and are faster than collections. If you don't need to use a collection's keyed lookup, easy extensibility, and other features, use an array instead.

—Rod Stephens, Boulder, Colorado

VB3, VB4 16/32, VB5, VBA, VBS

Level: Beginning

DRAW COMPLEX SHAPES

You can create complex shapes easily if you have a copy of Microsoft PowerPoint, CorelDraw, or any other graphics editor. Using your graphic editor's drawing tools, draw the shape you want in the presentation document. PowerPoint features a wide range of drawing tools and autoshapes that let you combine a variety of basic shapes and styles into complex images that you can fill, shade, or rotate to any angle.

When you're satisfied with your complex shape, choose Select from the Edit menu, and Copy from the Edit menu in PowerPoint. You can also save the image as a Windows metafile from packages that support that option. Switch tasks to your VB IDE, add an Image control to the active form, and paste the metafile that was copied from PowerPoint into the Image.

When you want to draw the stored shape in your running VB application, assign the metafile from the Image control to the destination drawing object:

```
picDraw.Picture = imgMetafile.Picture
```

Use drag-and-drop to copy the image from the source control to the drop target control. All scaling and drawing is handled automatically. You don't need to set the ScaleMode property of your destination object. You can use this technique when you want to decorate forms with complex shapes or create a Balloon Help form, a process control modeling application in which the images represent the elements of the production flow, or a flow-chart application.

—Rob Parsons, Dee Why, New South Wales, Australia

VB3, VB4 16/32, VB5

Level: Beginning

PASS PARAMETERS TO USER DIALOGS

It's fairly common practice to add buttons to your date and numeric fields to allow users to select and display a popup calendar or calculator. However, to display a calculator or calendar without allowing it to communicate with the calling application defeats the purpose of giving the user these capabilities. The called dialog is only an appendage that cannot pass the user's calculation back to the field from which the resource was called.

To solve this problem, add a button to a form next to a numeric input field (txtAmount). In the Click event of this button, place a Load statement for the form you want to show the user:

```
Sub cmd_Click()
    Load Calculator_Dialog
    Calculator_Dialog.Show 1   ' modal
End Sub
```

Pass parameters to a form by writing a wrapper utility that handles the passing of these parameters to and from the called dialog:

```
Sub cmd_Click()
    CalculatorDialog_Show txtAmount
End Sub

Sub CalculatorDialog_Show (C As Control)
'' Display the Calculator_Dialog and return
'' the calculated value as a string
Load Calculator_Dialog
Calculator_Dialog.lblReadout = _
    Format(Val(C.Text), "#.################")
' Pass the number to dialog
Calculator_Dialog.Op1 = Format(Val(C.Text), _
    "#.################")
' simulate user input
Calculator_Dialog.LastInput = "NUMS"
Calculator_Dialog.Show 1   ' modal
If Calculator_Dialog.Action = True Then
    C.Text = Format(Calculator_Dialog._
        lblReadout, "General Number")
End If
Unload Calculator_Dialog
Set Calculator_Dialog = Nothing
End Sub
```

The parameters are passed to hidden label controls on the Calculator_Dialog form between the time the form is loaded and the time it's displayed. A hidden label control on the Calculator_Dialog (Action) indicates whether the user chose the OK or Cancel button to exit the dialog. Rather than unload the Calculator_Dialog when the user presses the OK or Cancel button, the form hides until the calling sub (CalculatorDialog_Show) processes the value of Calculator_Dialog.lblReadout. The form is then unloaded, and the Set-to-Nothing statement is executed to purge memory of the form's controls and procedures.

This technique works with any version of VB. Starting with VB4, you can add properties and methods to forms, which is a preferable method to relying on hidden labels for interobject communication.

—Rob Parsons, Dee Why, New South Wales, Australia

VB3, VB4 16/32, VB5

Level: Intermediate

Let the User Control Long Operations

To allow the user to control long operations, use a single CommandButton to control starting and stopping a looping process:

```
Dim Running As Boolean
Private Sub CmdStartStop_Click()
    If Running Then
        ' Stop the process.
        Running = False
        CmdStartStop.Caption = "Start"
        CmdStartStop.Enabled = False
    Else
        ' Start the process.
        Running = True
        CmdStartStop.Caption = "Stop"
        RunLongProcess
        CmdStartStop.Enabled = True
    End If
End Sub
Private Sub RunLongProcess()
    Do While Running
        ' Do something repeatedly.
        :
        DoEvents
    Loop
End Sub
```

—Rod Stephens, Boulder, Colorado

VB3, VB4 16/32, VB5

Level: Beginning

A Quick Explorer-Like Interface

Although the new VB5 Application Wizard can build a shell application with an Explorer-like interface, it is often desirable to drop a quick Explorer form into your code without the toolbar or the extra code included by the wizard. You can use only three controls to create a form quickly with the Explorer split-panel look and feel.

To get an Explorer interface up and running, drop a picture box onto a form and set the BorderStyle property to zero (none). Add two controls to the picture box. These controls will represent the left and right panels. Drop in this code:

```
Private Sub Form_Resize()
    ' Optionally, resize manually if you don't
    ' want container to cover entire form and
    ' resize with form.
    Picture1.Move 0, 0, Me.ScaleWidth, Me.ScaleHeight
End Sub
Private Sub Picture1_Resize()
    Dim offset As Integer
    Static percent As Single

    offset = 4 * Screen.TwipsPerPixelX
    If percent = 0 Then
        percent = 0.5
Else
        percent = (Text1.Width + offset) / _
            Picture1.Width
    End If
    Text1.Top = 0
    Text1.Left = 0
    Text1.Width = Picture1.Width * percent - offset
    Text1.Height = Picture1.Height
    Text2.Top = 0
    Text2.Left = Text1.Width + offset
    Text2.Width = Picture1.Width - Text2.Left
    Text2.Height = Picture1.Height
End Sub
Private Sub Picture1_MouseMove(Button As _
    Integer, Shift As Integer, X As Single, _
    Y As Single)
    Dim offset As Integer, pwidth As Integer
    offset = 2 * Screen.TwipsPerPixelX
    pwidth = Picture1.Width

    If Button = vbLeftButton And X > 20 * _
        offset And X < pwidth - 20 * offset Then
        Text1.Width = X - offset
    ElseIf X < 20 * offset Then
        Text1.Width = 19 * offset
    ElseIf X > pwidth - 20 * offset Then
        Text1.Width = pwidth - 19 * offset
```

```
    End If
    Text2.Left = Text1.Width + 2 * offset
    Text2.Width = pwidth - Text2.Left
End Sub
```

In this example, I used two multiline text boxes with scrollbars, but TreeView and ListView controls work just as well.

—David Grantier, Atlanta, Georgia

VB5

Level: Intermediate

TRICK SETUP WIZARD

Don't you hate that VB5's Setup Wizard doesn't include all the files you reference in your application? For example, if you access a picture or animation file at run time using syntax such as *Animation1.Open App.Path & "\MyVideo.AVI"*, VB5's Setup Wizard doesn't include this file in its setup list.

To make VB5's Setup Wizard include files in the setup automatically, you can declare a fictional API procedure in the Declarations section of any module that references the file within a conditional compilation construct:

```
#If False Then
Private Declare Sub Foo Lib "VIDEO.AVI" ()
'Fake Declare
#End If
```

Because the condition is False, Visual Basic ignores this statement. However, the Setup Wizard interprets this line as a function declaration, locates the file (in this case, VIDEO.AVI), and includes it in the setup list. Setup Wizard will default to placing these files in your App folder, so change it accordingly.

—Miguel Santos, Aveiro, Portugal

VB3, VB4 16/32, VB5

Level: Beginning

READ AND WRITE ARRAYS QUICKLY

You can read and write arrays quickly from files using Put and Get. This approach is faster than reading and writing the array one entry at a time:

```
Dim arr(1 To 100000) As Long
Dim fnum As Integer
    fnum = FreeFile
    Open "C:\Temp\xxx.dat" For Binary As fnum
    Put #fnum, , arr
    Close fnum
```

—Rod Stephens, Boulder, Colorado

VB3, VB4 16/32, VB5

Level: Intermediate

PRINT GRAY LINES AND AREAS

Black and white printers cannot print gray lines, but they can dither to create gray areas. To create a gray border around an area, draw thin gray boxes with DrawStyle = vbInvisible (5) so they have no borders.

—Rod Stephens, Boulder, Colorado

VB4 32, VB5

Level: Intermediate

CREATE INTERNET-STYLE HYPERLINKS

You can easily create a hyperlink in a VB form to let your users connect to a Web page or send e-mail by clicking on the hyperlink. To do this, add a Label control to your form, and set the AutoSize property to True to ensure the label is exactly the same size as your text. Set the Caption property to the text you want, such as "Click here to send e-mail." Set the Font property to Underline. Set the ForeColor property to Blue or your favorite hyperlink color. Set the MousePointer property to 99 - Custom. Set the MouseIcon property to file H_POINT.CUR from your VB CD-ROM (vb\graphics\cursors). The familiar mouse cursor that browsers display over hyperlinks will be displayed when the mouse moves over the label. Double-click on the Label control to open the code window, and add this code to the Click event:

```
Private Sub Label1_Click()
    Dim lRet As Long
    Dim sText As String
    sText = "http://www.yourURL.com"
    lRet = ShellExecute(hwnd, "open", sText, _
        vbNull, vbNull, SW_SHOWNORMAL)
    If lRet >= 0 And lRet <= 32 Then
        MsgBox "Error executing command!"
    End If
End Sub
```

Add these declarations to the top of your form module:

```
Private Declare Function ShellExecute Lib _
    "shell32.dll" Alias "ShellExecuteA" ( _
    ByVal hwnd As Long, ByVal lpOperation As String, _
    ByVal lpFile As String, ByVal lpParameters As String, _
    ByVal lpDirectory As String, _
    ByVal nShowCmd As Long) As Long
Private Const SW_SHOWNORMAL = 1
```

In the label's Click event, you can change the value assigned to sText to specify what action ShellExecute performs. If you specify a URL, ShellExecute launches your Web browser and points it to the specified Web site. You can also use the mailto syntax (mailto:yourname@youraddress.com) to instruct ShellExecute to launch your e-mail program. You can even specify a file name, and ShellExecute automatically loads the file into the program registered to use it. For example, specifying a file with the DOC extension loads Microsoft Word on systems that have Word installed.

—José Rodríguez Alvira, San Juan, Puerto Rico

VB4 16/32, VB5

Level: Intermediate

DETERMINE IF YOUR PROGRAM IS RUNNING WITHIN VB'S IDE

Sometimes it is useful to know if your program is running within Visual Basic's Integrated Development Environment (IDE) or as a standalone EXE. This routine returns True if the program is running within VB's IDE:

```
Public Function RunningInIDE() As Boolean
    'Assume running as EXE for now
```

```
    RunningInIDE = False
    'Trap errors
    On Error GoTo RunningInIDEErr
    'Divide by zero (fails within IDE)
    Debug.Print 1 / 0
    'Exit if no error
    Exit Function
RunningInIDEErr:
    'We get error if Debug.Print was
    'evaluated (i.e. running in the VB IDE)
    RunningInIDE = True
End Function
```

RunningInIDE performs a division by zero in a Debug.Print statement. Within the IDE, this causes a divide overflow, which the function traps in an error handler. When the program is run as a standalone EXE, Debug.Print statements are not evaluated and no error occurs.

—Jeffrey Sahol, Greensboro, North Carolina

VB4 16/32, VB5

Level: Intermediate

Extra DLLs as a Result of Conditional Compilation

Starting with Visual Basic 4.0, you can add conditional compilation statements and use them to create different versions for 16-bit and 32-bit environments or to call debugging DLLs while testing your applications. However, you have to be careful when you use the setup wizard to generate the installation package because those files will be included unless you remove them from the distribution list. It is even worse when 16-bit code is upgraded to VB5 because some of the 16-bit DLLs are no longer included in the operating system, such as USER.DLL. In this case, you have to comment out all the references to these libraries.

—Gerardo Villeda, Drexel Hill, Pennsylvania

VB3, VB4 16/32, VB5

Level: Intermediate

Speed Up Selecting Items in a MultiSelect List Box

Try this routine to select or deselect all items in a list box. Add a list box to your form with the MultiSelect property set to Simple or Extended, and add items to the list box:

```
Sub ToggleSelectAll (lst As ListBox, ByVal bState As Integer)
    Dim i As Integer
    lst.Visible = False
    'Avoid redraws for each item
    For i = 0 To (lst.ListCount - 1)
        lst.Selected(i) = bState
    Next i
    lst.Visible = True
End Sub
```

This routine sets the list box's Visible property to False to prevent the list box from repainting itself after each item is selected. This makes the routine more efficient. However, you can speed up this routine by using SendMessage instead of VB statements:

```
Sub ToggleSelectAll (lst As ListBox, _
    ByVal bState As Integer)
    Dim i As Integer
    Dim nRet As Long
    lst.Visible = False
    For i = 0 To lst.ListCount - 1
        nRet = SendMessage(lst.hWnd, LB_SETSEL, bState, i)
    Next i
    lst.Visible = True
End Sub
```

You can speed up this routine significantly by taking advantage of the fact that the LB_SETSEL message sets the state of all items in the list box at once if the index argument is -1. Because this approach uses a single call to SendMessage, you don't need to worry about setting the list box's Visible property to False to prevent multiple redraws:

```
Sub ToggleSelectAll (lst As ListBox, _
    ByVal bState As Integer)
    Dim nRet As Long
    nRet = SendMessage(lst.hWnd, LB_SETSEL, bState, -1)
End Sub
```

Use these 16-bit declarations:

```
Declare Function SendMessage Lib "User" ( _
    ByVal hWnd As Integer, ByVal wMsg As Integer, _
    ByVal wParam As Integer, ByVal lParam As Long) As Long
Global Const WM_USER = &H400
Global Const LB_SETSEL = (WM_USER + 6)
```

Or use these 32-bit declarations:

```
Public Declare Function SendMessage Lib _
    "User32" Alias "SendMessageA" ( _
    ByVal hWnd As Long, ByVal wMsg As Integer, _
    ByVal wParam As Long, ByVal lParam As Long) As Long
Public Const WM_USER = &H400
Public Const LB_SETSEL = (WM_USER + 6)
```

—Garold Minkin, Brooklyn, New York

VB4 32, VB5

Level: Beginning

Modify a Toolbar's Image List Control

Have you ever tried to create tools on a toolbar using an image list, only to find that you must detach the toolbar from the list and reset all the toolbar button properties every time you want to add to the image list? Annoying, isn't it?

Before adding anything to the image list, cut the toolbar object. Then you can revise the image list with no complaints from the IDE. Provided that you retain all the original image indexes, you can paste the toolbar object back on the form and retain the toolbar's link to the image list.

—David J. Schraff, Cleveland, Ohio

VB4 32, VB5

Level: Intermediate

EASILY CREATE ALL DIRECTORIES IN A PATH

Use this code to create a tree of folders on Windows 95 or NT 4.0 in one line:

```
Private Declare Function _
    MakeSureDirectoryPathExists Lib "IMAGEHLP.DLL" _
    (ByVal DirPath As String) As Long
Private Sub Command1_Click()
    Dim MyPath As String
    Dim bResult As Long
    'Specify the directory that we need
    MyPath = "C:\Win95\Temp\MyFolder\" & _
        "MyFolder\Fld1\Fld2\"
    'Now create that directory if it does exist.
    'The entire path will be created if needed
    bResult = MakeSureDirectoryPathExists(MyPath)
    'Test for failure
    If bResult = 0 Then
        MsgBox "MakeSureDirectoryPathExists failed!"
    End If
End Sub
```

—Mordechai Eli, received by e-mail

VB4 16/32, VB5

Level: Intermediate

IMPLEMENT A BINARY TREE

A binary search tree can be useful when you have to traverse a lot of data in sorted order. As this CBinarySearchTree class demonstrates, you can implement binary search trees easily using objects and recursion (both data recursion and procedural recursion):

```
'class properties:
Private LeftBranch As CBinarySearchTree
Private RightBranch As CBinarySearchTree
Private NodeData As String
'Adds a new value to the binary tree
Public Sub AddNode(NewData As String)
    If Len(NodeData) = 0 Then
        'Store data in current node if empty
        NodeData = NewData
    ElseIf NewData < NodeData Then
        'Store data in left branch if NewData < NodeData
        If LeftBranch Is Nothing Then
            Set LeftBranch = New CBinarySearchTree
        End If
        LeftBranch.AddNode NewData
    Else
        'Store data in right branch if NewData
        '>= NodeData
        If RightBranch Is Nothing Then
            Set RightBranch = New CBinarySearchTree
        End If
        RightBranch.AddNode NewData
    End If
End Sub

'Displays all values in this tree
'If called on a child node, displays all
'values in this branch
Public Sub TraverseTree()
    'Traverse left branch
    If Not LeftBranch Is Nothing Then
        LeftBranch.TraverseTree
    End If
    'Display this node
    MsgBox NodeData
    'Traverse right branch
    If Not RightBranch Is Nothing Then
        RightBranch.TraverseTree
    End If
End Sub
```

Test this class by creating a new CBinarySearchTree object, calling AddNode a few times to store data, and then calling TraverseTree to see the results. Binary search trees don't get much simpler than this.

—David Doknjas, Surrey, British Columbia, Canada

VB3, VB4 16/32, VB5

Level: Advanced

USE TOOLBAR-STYLE TITLE BARS

To make a form use a small toolbar-style title bar, set the form's WS_EX_TOOLWINDOW extended style:

```
Declare Function GetWindowLong Lib "user32" _
    Alias "GetWindowLongA" ( _
    ByVal hwnd As Long, _
    ByVal nIndex As Long) As Long
Declare Function SetWindowLong Lib "user32" _
    Alias "SetWindowLongA" ( _
    ByVal hwnd As Long, _
    ByVal nIndex As Long, _
    ByVal dwNewLong As Long) As Long
Public Const WS_EX_TOOLWINDOW = &H80&
Public Const GWL_EXSTYLE = (-20)
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
Public Const SWP_FRAMECHANGED = &H20
Public Const SWP_NOMOVE = &H2
Public Const SWP_NOZORDER = &H4
Public Const SWP_NOSIZE = &H1
Private Sub Form_Load()
Dim old_style As Long
    old_style = GetWindowLong(hwnd, GWL_EXSTYLE)
    old_style = SetWindowLong(hwnd, _
        GWL_EXSTYLE, old_style Or _
        WS_EX_TOOLWINDOW)
    SetWindowPos hwnd, 0, 0, 0, 0, 0, _
        SWP_FRAMECHANGED Or SWP_NOMOVE Or _
        SWP_NOZORDER Or SWP_NOSIZE
End Sub
```

—Rod Stephens, Boulder, Colorado

VB4 32, VB5

Level: Intermediate

USE UNC NAMES TO REFER TO NETWORK DRIVES

If you write programs that access information on a network

server, you never want to hard-code drive letters. Not all software vendors take advantage of the Uniform Naming Convention (UNC) (for example, \\SERVER\VOLUME\) technology. Use these API declarations:

```
Public Declare Function GetVolumeInformation _
     Lib "kernel32" Alias "GetVolumeInformationA" ( _
     ByVal lpRootPathName As String, _
     ByVal lpVolumeNameBuffer As String, _
     ByVal nVolumeNameSize As Long, _
     lpVolumeSerialNumber As Long, _
     lpMaximumComponentLength As Long, _
     lpFileSystemFlags As Long, _
     ByVal lpFileSystemNameBuffer As String, _
     ByVal nFileSystemNameSize As Long) As Long

Public Function GetNetworkDrive(FileSystem _
     As String, VolumeName As String) As String

     'Returns the first mapped drive letter
     'if successful or "NODRIVE" if fails.

     'FileSystem and SysName refer to the
     'Network file system type:
     'NWCOMPA => Novell
     'NTFS    => Microsoft
     'FAT     => Local\Microsoft

     Dim SerialNum As Long, SysFlags As Long
     Dim retVal As Long, Complength As Long
     Dim VolBuff As String * 255, SysName As String * 255
     Dim DrivePath As String
     Dim i As Integer

     GetNetworkDrive = "NODRIVE"

     For i = 70 To 90
          DrivePath = Chr(i) & ":\"
          retVal = GetVolumeInformation _
               (DrivePath, VolBuff, 255, _
               SerialNum, Complength, SysFlags, SysName, 255)
          If (Mid$(VolBuff, 1, Len(VolumeName)) _
               = UCase$(VolumeName)) And(Mid$(SysName,1, Len _
               (FileSystem)) = FileSystem) Then
                    GetNetworkDrive = DrivePath
                    Exit For
          End If
     Next i
End Function
```

Here's the wrong and right ways to refer to network drives:

```
Option Explicit

Public Sub OpenDatabase()
     'The Wrong Way!
     Dim filespec as String
     filespec = "H:\PUBLIC\APPS\BTRIEVE\MYDB.RDB"
     'Some third-party control that opens the
     'database file.
     ...
     ...
     ...
End Sub

Public Sub OpenDatabase()
     'A Better Way!
     Dim filespec as String
     Dim Vol1Drive As String
     Vol1Drive = GetNetworkDrive("NWCOMPA","VOL1")
     If Vol1Drive = "NODRIVE" Then
          MsgBox "Unable to open database. " & _
               "The user does not have a " & _
               "drive letter mapped to VOL1"
     Else
          filespec = Vol1Drive & _
               "\PUBLIC\APPS\BTRIEVE\MYDB.RDB"
          'Some third-party control that opens
          'the database file.
          ...
          ...
          ...
     End If
End Sub
```

—Charles Douglas, New Orleans, Louisiana

VB4 32, VB5

Level: Intermediate

Suspend Code While Waiting for Shelled Process

This "wait" routine suspends your code and waits for the shelled process to finish. It comes in handy when you need to run another script\batch file during your process. Use this calling syntax:

```
Private sub cmdPrint_Click()
     WaitForProcessToEnd "C:\PLOTTER.BAT"
     'Let the user know when the process is finished.
     MsgBox "The process is finished! "
End Sub
```

Here are the API declarations:

```
Public Const INFINITE = -1&
Public Declare Function WaitForSingleObject _
     Lib "kernel32" (ByVal hHandle As Long, _
     ByVal dwMilliseconds As Long) As Long
Public Declare Function OpenProcess Lib _
     "kernel32" (ByVal dwAccess As Long, _
     ByVal fInherit As Integer, _
     ByVal hObject As Long) As Long
Public Sub WaitForProcessToEnd(cmdLine As String)
     'You can substitute a discrete time
     'value in milliseconds for INFINITE.
     Dim retVal As Long, pID As Long, pHandle _
          As Long
     pID = Shell(cmdLine)
     pHandle = OpenProcess(&H100000, True, pID)
     retVal = WaitForSingleObject(pHandle, INFINITE)
End Sub
```

—Charles Douglas, New Orleans, Louisiana

VB4 32

Level: Advanced

More Tips for "Creating a Better Mouse Trap"

I would like to add a few things to the "Building a Better Mouse Trap" tip [*VBPJ* November 1996, page 56]. This approach also works for any message, including button-up and button-down messages. Windows 95 calls the appropriate function and passes the mouse coordinates and other information in lParam and

wParam, which, in the case of the MouseMove event, VB converts to the Button, Shift, X, and Y arguments. If the user clicks on the icon in the system tray, Windows passes 513 and 514 (WM_LBUTTONDOWN and WM_LBUTTONUP) to VB, and VB performs a conversion that leaves the original mouse message in X. If ScaleMode of the invisible form, frmOmni in the example, is left as Twips, this tip won't work for every system. To make this tip work on every system, set ScaleMode of the invisible form to Pixels.

—Goran Stijacic, San Diego, California

VB4 32, VB5

Level: Advanced

CREATE A GLOBALLY UNIQUE IDENTIFIER (GUID)

I use this routine in almost every application I develop. It's handy whether you need a key for a collection or whether you use it in a database:

```
Option Explicit

Private Declare Function CoCreateGuid Lib _
    "OLE32.DLL" (pGuid As GUID) As Long
Private Declare Function StringFromGUID2 Lib _
    "OLE32.DLL" (pGuid As GUID, _
    ByVal PointerToString As Long, _
    ByVal MaxLength As Long) As Long

' Private members
Private Const GUID_OK As Long = 0

Private Type GUID
    Guid1             As Long
    Guid2             As Integer
    Guid3             As Integer
    Guid4(0 To 7)     As Byte
End Type

Public Function CreateGUIDKey() As String
    '*** Possible max length for buffer
    Const GUID_LENGTH As Long = 38

    Dim udtGUID As GUID
    'User Defined Type
    Dim strFormattedGUID As String
    'The formatted string
    Dim lngResult As Long
    'Useless result flag

    '*** Create a 'raw' GUID
    lngResult = CoCreateGuid(udtGUID)

    If lngResult = GUID_OK Then
        '*** Pre-allocate space for the ID
        strFormattedGUID = String$(GUID_LENGTH, 0)
        '*** Convert the 'raw' GUID to a
        'formatted string
        StringFromGUID2 udtGUID, _
            StrPtr(strFormattedGUID), GUID_LENGTH + 1
    Else
        '*** Return nothing or handle error
        strFormattedGUID = ""
    End If
    ' *** Return our nicely formatted GUID
    CreateGUIDKey = strFormattedGUID
End Function
```

—Doug Olson, Eden Prairie, Minnesota

VB4 32, VB5

Level: Advanced

SEND MESSAGES TO WINPOPUP FROM YOUR APPLICATION

If you have an application that must run "unattended" and you want it to alert you if something goes wrong, your application can send you a message through WinPopUp. WinPopUp uses a mail slot to communicate with other computers in the network. Choose a mail slot name, such as \\computername \mailslot\messngr, and use this code:

```
Option Explicit

Private Declare Function CloseHandle Lib _
    "kernel32" (ByVal hHandle As Long) As Long
Private Declare Function WriteFile Lib _
    "kernel32" (ByVal hFileName As Long, _
    ByVal lpBuff As Any, _
    ByVal nNrBytesToWrite As Long, _
    lpNrOfBytesWritten As Long, _
    ByVal lpOverlapped As Long) As Long
Private Declare Function CreateFile Lib _
    "kernel32" Alias "CreateFileA" ( _
    ByVal lpFileName As String, _
    ByVal dwAccess As Long, _
    ByVal dwShare As Long, _
    ByVal lpSecurityAttrib As Long, _
    ByVal dwCreationDisp As Long, _
    ByVal dwAttributes As Long, _
    ByVal hTemplateFile As Long) As Long

Private Const OPEN_EXISTING = 3
Private Const GENERIC_READ = &H80000000
Private Const GENERIC_WRITE = &H40000000
Private Const GENERIC_EXECUTE = &H20000000
Private Const GENERIC_ALL = &H10000000
Private Const INVALID_HANDLE_VALUE = -1
Private Const FILE_SHARE_READ = &H1
Private Const FILE_SHARE_WRITE = &H2
Private Const FILE_ATTRIBUTE_NORMAL = &H80

Function SendToWinPopUp(PopFrom As String, _
    PopTo As String, MsgText As String) As Long
    ' parms:     PopFrom: user or computer that
    '            sends the message
    '            PopTo: computer that receives the
    '            message
    '            MsgText: the text of the message
    '            to send
    Dim rc  As Long
    Dim mshandle As Long
    Dim msgtxt As String
    Dim byteswritten As Long
    Dim mailslotname As String
    ' name of the mailslot
    mailslotname = "\\" + PopTo + _
        "\mailslot\messngr"
    msgtxt = PopFrom + Chr(0) + PopTo + Chr(0) + _
        MsgText + Chr(0)
    mshandle = CreateFile(mailslotname, _
```

```
        GENERIC_WRITE, FILE_SHARE_READ, 0, _
        OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0)
    rc = WriteFile(mshandle, msgtxt, _
        Len(msgtxt), byteswritten, 0)
    rc = CloseHandle(mshandle)
End Function
```

—Mauro Mariz, Trento, Italy

VB4 16/32, VB5

Level: Beginning

Fill a List Box with the Values from a Database Table

Use this code to fill a list box with the values from a database table. The first field in the query provides the ItemData for the list:

```
Call FillList(dbase, "select personno, " & _
    "personname from tblperson order by " & _
    "personname;", lstperson)
Sub FillList(thedb As Database, thesql As _
    String, THELIST As Control)
On Error Resume Next
    THELIST.Clear
    Call FillListAp(thedb, thesql, THELIST)
End Sub
Sub FillListAp(thedb As Database, thesql As _
    String, THELIST As Control)
    On Error Resume Next
    Dim theset As Recordset
    Dim inlist As String
    Dim I As Integer
    Set theset = thedb.OpenRecordset(thesql, _
        dbOpenSnapshot)
    While Not theset.EOF
        For I = 1 To theset.Fields.Count - 1
            If I = 1 Then
                If IsNull(theset.Fields(I)) Then
                    inlist = "Null"
                Else
                    inlist = theset.Fields(I)
                End If
            Else
                If IsNull(theset.Fields(I)) Then
                    inlist = inlist & Chr(9) & "Null"
                Else
                    inlist = inlist & Chr(9) & _
                        theset.Fields(I)
                End If
            End If
        Next I
        THELIST.AddItem inlist
        THELIST.ItemData(THELIST.NewIndex) = _
            theset.Fields(0)
        theset.MoveNext
    Wend
    theset.Close
End Sub
```

—Tim Miller, received by e-mail

VB3, VB4 16/32, VB5

Level: Intermediate

Make Sure Data is Safe Before Unloading

To decide when it is safe to exit your program, use a DataUnsafe function. Set the DataModified variable to False when you load or save data, or when you start a new file. Set DataModified to "OK" to remove the current data:

```
Dim DataModified As Boolean
Private Function DataUnsafe() As Boolean
    If DataModified Then
        Select Case MsgBox _
            ("Save changes to the data?", vbYesNoCancel)
            Case vbYes
                ' Save the data. SaveData should
                ' set DataModifed False if successful.
                SaveData
                DataUnsafe = DataModified
            Case vbNo
                ' OK to clear data.
                DataUnsafe = False
            Case vbCancel
                ' Cancel operation and keep data.
                DataUnsafe = True
        End Select
    Else
        ' The data is not modified.
        DataUnsafe = False
    End If
End Function
Private Sub Form_Unload(Cancel As Integer)
    Cancel = DataUnsafe()
End Sub
```

—Rod Stephens, Boulder, Colorado

VB3, VB4 16/32, VB5

Level: Beginning

Convert from Fractional to Decimal Numbers

While developing a database front end for hand-tool management, I discovered a need to handle both fractional and decimal representations of dimensions in the same text box. This makes it easier on users who worked from a variety of prints to input part feature sizes. To accomplish this, place this function in the LostFocus event of any text box that can receive numerical input. You can also cycle through the appropriate text boxes and run the function against the text value of each. In addition, this function only checks for the inches character (double quotes) at the end of the text string of fractional dimensions. It also looks for spaces and/or dashes between whole numbers and fractions and checks for both forward and backward slashes within fractions. It doesn't work with negative values:

```
Private Function ReturnDecimalValue(Size As _
    String) As String
    Dim strSize As String
    Dim iGap As Integer
    Dim iSlash As Integer
    Dim sWhole As Single
    If Size <> "" Then Size = LTrim(RTrim(Size))
```

```
    'previous code may have stripped text to nothing
    'if it was just spaces, so test
    If Size <> "" Then
        'strip off inch character (double
        'quotes) if it's there
        If Right(Size, 1) = Chr$(34) Then _
            Size = Left(Size, Len(Size) - 1)
        iGap = InStr(Size, "-")
        If iGap = 0 Then iGap = InStr(Size, " ")
        If iGap Then sWhole = CSng(Left(Size, iGap - 1))
        strSize = Right(Size, Len(Size) - iGap)
        iSlash = InStr(strSize, "/")
        'user may have input backward slash
        'in fraction instead of forward slash;
        'verify
        If iSlash = 0 Then iSlash = InStr(strSize, "\")
        'convert result to decimal form for
        'saving in database
        If iSlash Then Size = CStr(sWhole + _
            (CSng(Left(strSize, iSlash - 1)) / _
            CSng(Right(strSize, Len(strSize) - iSlash))))
    End If
    ReturnDecimalValue = Size
End Function
```

—Randall Arnold, Coppell, Texas

VB3, VB4 16/32, VB5

Level: Intermediate

SEED THE RANDOM NUMBER GENERATOR

You can use Rnd followed by Randomize to seed Visual Basic's random number generator. Whenever you use the same seed, Rnd produces the same sequence of random numbers:

```
Rnd -1
Randomize seed
```

—Rod Stephens, Boulder, Colorado

VB4 16/32, VB5

Level: Beginning

BETTER AUTO-PROPERCASE

The "Auto-Propercase Text Box at Entry" tip [*VBPJ* May 1997, page 63] has a simpler solution. The StrConv function can propercase any string. You can achieve the same effect with this code in the KeyUp event of a text box:

```
Private Sub Text1_KeyUp(KeyCode As Integer, _
    Shift As Integer)
    Dim iCurPos As Integer
    iCurPos = Text1.SelStart
    Text1.Text = StrConv(Text1, vbProperCase)
    Text1.SelStart = iCurPos
End Sub
```

—Manish Garg, Bloomington, Minnesota

VB3, VB4 16/32, VB5

Level: Intermediate

AVOID FORM CONFUSION

When you define a form named MyForm, Visual Basic creates one instance with the name "MyForm." Many programmers confuse this special instance with other instances created using Dim. For example, the Caption statement in this code sets the caption on the automatically created instance. When the instance FRM appears, its caption is not set:

```
Dim frm As New MyForm
    MyForm.Caption = "This is the single instance"
    frm.Show
```

Even more confusing problems can arise if you use MyForm within the MyForm code module, when an instance such as FRM might accidentally set properties on the automatically created form rather than itself. For this reason, never use MyForm within the MyForm code module.

—Rod Stephens, Boulder, Colorado

VB3, VB4 16/32, VB5

Level: Beginning

READ-ONLY TEXT CONTROLS WITHOUT THE CARET

Assume you want a text control with a vertical scrollbar, but you don't want users to be able to modify it and you don't want users to see any caret in the control when they click on it. However, you still want users to be able to scroll all text.

Here is the solution. First, put a Text1 control and a Command1 control on Form1. Change the Text1 control property to Locked = True. Change the Text1 control property to MousePointer = 1 - Arrow. Change the Text1 control property to ScrollBars = 2 - Vertical, and enter this code:

```
'code in Form1
Private Sub Form_Load()
    'make Command1 control disappear when the form shows
    Command1.Left = -2000
End Sub
'code in Text1 control
Private Sub Text1_GotFocus()
    'move the focus to somewhere else
    Command1.SetFocus
End Sub
```

—Jian Wang, Woodside, New York

VB3, VB4 16/32, VB5

Level: Beginning

SWITCH TO ANOTHER APP AND CLOSE IT

Use the built-in AppActivate command to switch to another application easily if you know its title. You can even close the application programmatically:

```
'This will close app
Private Sub CloseApp (AppTitle As String)
    On Error GoTo ErrExit
    AppActivate AppTitle
    SendKeys "%{F4}", True
ErrExit:
End Sub
```

—Ketan Patel, Lansing, Michigan

VB3, VB4 16/32, VB5

Level: Intermediate

SHOW 3-D TEXT MESSAGES

If you want to print text on an object with 3-D effects, use this subroutine to convert fonts into 3-D fonts with borders. In this routine, the user can define shadow length, shadow color, font color, border color, and position of text on the object. Note that all color values are in the range of 0-15 because they are used as arguments for the QBColor function:

```
Sub Fonts3d(Print_Object As Object, Text1 As _
     String, postx As Single, Posty As _
     Single, Shadow_Length As Integer, _
     FontsColor As Integer, ShadowColor As _
     Integer, BorderColor As Integer)
     Dim I As Integer, Prev_Scale_Mode As Integer
     Prev_Scale_Mode = Print_Object.ScaleMode
     If postx = -1 Then 'for center align
          postx = (Print_Object.ScaleWidth - _
               Print_Object.TextWidth(Text1)) / 2
     End If
     Print_Object.ForeColor = QBColor(ShadowColor)
     'Generate shadow
     For I = 1 To Shadow_Length * 16 Step 8
          Call PrintText(Print_Object, _
               postx + I, Posty + I, Text1)
     Next I
     'Print border
     Print_Object.ForeColor = QBColor(BorderColor)
     Call PrintText(Print_Object, postx - 15, Posty, Text1)
     Call PrintText(Print_Object, postx + 15, Posty, Text1)
     Call PrintText(Print_Object, postx, Posty - 15, Text1)
     Call PrintText(Print_Object, postx, Posty + 15, Text1)
     Print_Object.ForeColor = QBColor(FontsColor)
     Call PrintText(Print_Object, postx, Posty, Text1)
End Sub
Sub PrintText(Print_Object As Object, _
     Xposition As Single, Yposition As _
     Single, Text1 As String)
     Print_Object.CurrentX = Xposition
     Print_Object.CurrentY = Yposition
     'Print text on object
     Print_Object.Print Text1
End Sub
' example of usage:
Call Fonts3d(Picture1, "WELCOME TO T.T.T.I.", _
  50, 150, 5, 6, 11, 12)
```

—Atmabodh Hande, Shamla Hills, Bhopal, India

VB5

Level: Beginning

ADD RIGHT-CLICK FUNCTIONALITY

By default, the right-click functionality needed for What's This? Help is not available in VB. Selecting a menu command from a context menu created for the button lets you display the What's This? Help topic when the user right-clicks on a command button. First, set the WhatsThisHelp property of the form to True. Place a CommandButton control on the form. Using the Menu Editor, create a menu with a top-level invisible item named mnuBtnContextMenu, and a submenu named mnuBtnWhatsThis with a caption of "What's This?". Finally, place this code in the MouseUp event of the command button:

```
Private ThisControl As Control
Private Sub Command1_MouseUp(Button As _
     Integer, Shift As Integer, X As Single, Y As Single)
     If Button = vbRightButton Then
     Set ThisControl = Command1
     PopupMenu mnuBtnContextMenu
     End If
     Set ThisControl = Nothing
End Sub
Private Sub mnuBtnWhatsThis_Click()
     ThisControl.ShowWhatsThis
End Sub
```

—Blue Sky Software Technical Support

VB5

Level: Beginning

ADD THE "?" BUTTON TO A FORM

The WhatsThisButton property returns or sets a value that determines whether the WhatsThisButton appears in the title bar of a Form object during run time. For the WhatsThisButton to appear on your form, set the WhatsThisButton and WhatsThisHelp properties to True; the ControlBox property to True; the BorderStyle property to Fixed Single or Sizable; and the MinButton and MaxButton properties to False, or the BorderStyle property to Fixed Dialog.

—Blue Sky Software Technical Support

VB4 16/32, VB5

Level: Advanced

PROVIDE PRINT PREVIEW

You can provide Print Preview at different scales. First, make your printing routine take on the object it should draw as a parameter. Then, for printing, pass the routine the Printer object. For preview, pass the routine a hidden PictureBox. Then use PaintPicture to copy the hidden picture to a visible PictureBox at the desired scale. Drawing at full scale on the hidden PictureBox allows you to scale fonts and other graphics without distortion.

—Rod Stephens, Boulder, Colorado

VB3, VB4 16/32, VB5

Level: Intermediate

FASTER SEARCHES ON ARRAYS, LIST BOXES, AND COMBO BOXES

This method is handy when the user enters data that you need to validate against a lookup table. Validating the value by querying the database is out of the question because of the relatively slow response time. An alternative is to keep the lookup table locally in a sorted array, or in a list-box or a combo-box control whose Sorted property is set to True. Lookup tables are static, so you don't need to worry about the local copy being out of sync with the database. For large sorted arrays of strings or for list-box or combo-box controls with large sorted lists (in the range of 10,000 entries), you will find the binary search to be 10 to 20 times faster than calling the API, and hundreds of times faster than a sequential search. The difference becomes dramatic if the operation needs to be done multiple times. If you want to

use an array instead of a combo-box or a list-box control, there is no API call you can use. Use this code for an array:

```
'In:
'Array to search
Dim rasArray() As String
'String to search for
Dim vsName As String
'Out:
'Index in the array of the string if found
Dim rlIndex As Long
'Local variables:
'Index in the array
Dim lnIdx As Long
'Lower bound of the search interval
Dim lnMin As Long
'Upper bound of the search interval
Dim lnMax As Long
'Return an invalid index, if string is not found
rlIndex = LBound(rasArray) - 1
lnMax = UBound(rasArray)
lnMin = LBound(rasArray)
'lookup vsName in rasArray()
Do While lnMin <= lnMax
    lnIdx = (lnMax + lnMin) \ 2
    If vsName = rasArray(lnIdx) Then
        rlIndex = lnIdx
        Exit Do
    ElseIf vsName < rasArray(lnIdx) Then
        lnMax = lnIdx - 1
    Else
        lnMin = lnIdx + 1
    End If
Loop
```

You can easily modify this code for a combo-box control to use with list-box controls as well:

```
'In:
'Combo to search
' (change into As ListBox for listbox controls
' or use As Controls to use with both types
' of controls)
Dim rcboCombo As ComboBox
'String to search for
Dim vsName As String
'Out:
'Index in the combo if the string is found
Dim rlIndex As Long
'Local variables
'Index in the array
Dim lnIdx As Long
'Lower bound of the search interval
Dim lnMin As Long
'Upper bound of the search interval
Dim lnMax As Long
'Return an invalid index, if string is not found
rlIndex = -1
lnMin = 0
lnMax = rcboCombo.ListCount - 1
lnIdx = lnMax \ 2
'lookup name in the combo
Do While rlIndex = -1 And lnMin <= lnMax
    If vsName = rcboCombo.List(lnIdx) Then
        rlIndex = lnIdx
    ElseIf vsName < rcboCombo.List(lnIdx) Then
        lnMax = lnIdx - 1
        lnIdx = (lnMax + lnMin) \ 2
    Else
        lnMin = lnIdx + 1
        lnIdx = (lnMax + lnMin) \ 2
    End If
Loop
```

—Adrian Cerchia, received by e-mail

VB3, VB4 16/32, VB5

Level: Intermediate

COMPRESS SHORT STRINGS INTO A LONG VALUE

When developing relational database applications, you often need to use list boxes or combo boxes to store records with alphanumeric keys. You can load numeric keys into the ItemData array of combo boxes and list boxes as Long integers. Programmers traditionally store alphanumeric keys in a String array with the index of the array elements stored in ItemData. However, for four-character, alphanumeric keys, avoid the overhead of an array by using CodeCrypt to convert the key into a Long integer.

CodeCrypt accepts a variant and returns a variant. If a four-character string is received, then CodeCrypt converts it to a Long integer. Each character, in turn, is converted to its ASCII value. The value in CodeCrypt is multiplied by 256 to shift the current value left by one byte. The ASCII value is then added to CodeCrypt (filling the empty low-order byte). Because a Long integer is made up of 32 bits (four bytes), up to four characters can now be stored in ItemData. If CodeCrypt receives a Long integer, the process is reversed and a string is returned:

```
Function CodeCrypt(pvInput As Variant) As Variant
    Dim i As Integer, temp As Long
    If UCase(TypeName(pvInput)) = "LONG" Then
        temp = pvInput
        Do While temp > 0
            CodeCrypt = Chr$(temp Mod 256) & CodeCrypt
            temp = CLng(temp / 256)
        Loop
    Else
        If Len(pvInput) > 4 Then Exit Function
        For i = 1 To Len(pvInput)
            CodeCrypt = CodeCrypt * 256 + _
                Asc(UCase$(Mid$(pvInput, i, 1)))
        Next i
    End If
End Function
```

Use this code to add alphanumeric code to lstBox.ItemData:

```
lstBox.AddItem text
lstBox.ItemData(lstBox.NewIndex) = CodeCrypt(code)
```

Use this code to get alphanumeric code back from lstBox.ItemData:

```
sCode = CodeCrypt(lstBox.ItemData(lstBox.ListIndex))
```

This routine is unable to deal with strings whose first character is greater than Chr$(127).

—Matthew B. Butler and Steve Hochreiter, San Angelo, Texas

VB5

Level: Intermediate

SORT DATA IN THE SHORT DATE FORMAT

Suppose you have a database field that returns a date in the Short Date format. Neither a numeric sort nor a string sort would order this column correctly. To sort by date, you need to add an extra column to the grid and set its width to zero. Populate the column with values obtained by converting the date to a number, and sort on that column as demonstrated here (this code assumes the date field is in column 2):

```
Dim Ro As Integer
Dim SortCol As Integer
Dim SortDate As Double

'add a column to hold the sort key
MSFlexGrid1.Cols = MSFlexGrid1.Cols + 1
SortCol = MSFlexGrid1.Cols - 1
MSFlexGrid1.ColWidth(SortCol) = 0 'invisible

'calculate key values & populate grid
For Ro = 1 To MSFlexGrid1.Rows - 1
SortDate = DateValue(MSFlexGrid1.TextMatrix(Ro, 2))
MSFlexGrid1.TextMatrix(Ro, SortCol) = SortDate
Next Ro

'do the sort
MSFlexGrid1.Col = SortCol 'set the key
MSFlexGrid1.Sort = flexSortNumericAscending
```

—VideoSoft Tech Support

VB4 16/32

Level: Intermediate

PROGRESS BARS WITH A GRADIENT

The tip "Color Status Indicator" ["101 Tech Tips for VB Developers," Supplement to the February 1997 issue of *VBPJ*, page 26] got me thinking that it would look even more attractive to have the status change color in gradient steps. Instead of changing the color at particular points, start at red, blend to orange, then yellow, and finally green.

To try this, place a Sheridan 3-D panel on your form and set the FloodType property to 1 (Left to Right) and ShowFloodPct to False. Place a command button and a text box (set the Caption to blank) on the form to do your testing. Place this code in the form:

```
Private Sub Command1_Click()
    Dim i As Integer, iIncrement As Integer
    Dim iMidPoint As Integer, iCheckpoint As Integer
    Dim j As Integer, sAddIncrement As String
    Dim sSubIncrement As String
    iIterations = Val(Text1)
    iMidPoint = iIterations * 0.5 - 1
    ' find the midpoint
    If iIterations / 510 = iIterations \ 510 Then
        'perfect algorithm
        iIncrement = 1
        iCheckpoint = 1
    ElseIf iIterations < 510 Then
        'increment the color faster
        iIncrement = 510 \ iIterations
        iCheckpoint = 1
    Else
        'change the color less frequently
        iIncrement = 1
        iCheckpoint = iIterations \ 510 + 1
    End If
    sAddIncrement = "&H0000" & _
        Format$(iIncrement, "00") & "00"
    sSubIncrement = "&H0000" & _
        Format$(iIncrement, "0000")
    SSPanel1.FloodColor = &HFF
    'start the panel at Red
    'At your preference show the
    'percentage. However, the tighter
    'the loop, the less valuable this is.
    SSPanel1.FloodShowPct = True
    For i = 0 To iIterations - 1
        'do your real processing here
        If i / iCheckpoint = i \ iCheckpoint Then
            With SSPanel1
                If i <= iMidPoint Then
                    .FloodColor = _
                        .FloodColor + sAddIncrement
                Else
                    .FloodColor = _
                        .FloodColor - sSubIncrement
                End If
                .FloodPercent = (i + 1) / iIterations * 100
            End With
        End If
        'if the loop is tight, be sure to
        'release the processor
        'occasionally - also, some flood
        'controls (other than SSPanel)
        'may require this to get the
        'color updated in the control
        DoEvents
    Next
    MsgBox "All done!", vbOKOnly + vbInformation
    SSPanel1.FloodPercent = 0
    SSPanel1.FloodShowPct = False
End Sub
```

The effect works best when your video is set to at least 16-bit (High Color). Because there are exactly 510 progressive gradients between red and green, the number of iterations you enter will influence what shade of green you end up with. As the iterations increase (>900), you'll end up with bright green.

—Jim Poland, Cedar Rapids, Iowa

VB3, VB4 16/32, VB5

Level: Intermediate

ADD USER PROPERTIES TO CONTROLS

To add user-defined properties to controls, use this code to store them in the control's tag. You can define multiple additional properties at design or run time. You can easily retrieve the value stored:

```
Control.Tag = AddParam((Control.Tag), _
    "tooltip", "Close this window.")
Control.Tag = AddParam((Control.Tag), "user", "True")
Tooltip = GetParam((Control.Tag), "tooltip")

Function AddParam (Text As String, _
    Param As String, Value) As String
Dim Text1 As String
If Not IsNull(GetParam(Text, Param)) Then
    Text1 = DelParam(Text, Param)
```

```
Else
    Text1 = Text
End If
If Text1 <> "" Then
    Text1 = Text1 & ";"
End If
AddParam = Text1 & Param & "=" & (Value)
End Function
Function DelParam (Text As String, _
    Param As String) As String
Dim i As Integer
Dim j As Integer
Dim s1 As String
Dim s2 As String
Dim Text1 As String
Text1 = Text
i = InStr(UCase(Text), UCase(Param) & "=")
If i > 0 Then
    If i = 1 Then
        s1 = ""
    Else
        s1 = Left(Text, i - 2)
    End If
    j = InStr(i, Text, ";")
    If j > 0 Then
        s2 = Mid(Text, j + 1)
    Else
        s2 = ""
    End If
    If s1 = "" Or s2 = "" Then
        Text1 = s1 & s2
    Else
        Text1 = s1 & ";" & s2
    End If
End If
DelParam = Text1
End Function
Function GetParam (Text, Param)
Dim i As Integer
Dim j As Integer
Dim Text1 As String
If Len(Text) Then
    If Left(Text, 1) <> "'" Then
        Text1 = ";" & Text & ";"
        i = InStr(LCase(Text1), ";" _
            & LCase(Param) & "=")  + 1
        If i > 1 Then
            j = InStr(i + Len(Param), Text1, ";")
            GetParam = Mid(Text1, i + Len(Param) + 1, _
                j - i - Len(Param) - 1)
        Else
            GetParam = Null
        End If
    Else
        GetParam = Null
    End If
Else
    GetParam = Null
End If
End Function
```

—Dean Genner, North Sydney, Australia

VB4 16/32, VB5

Level: Beginning

Jump Back and Forth Among Procedures Quickly

My code often has a lot of nested procedure calls, which can be painful to trace while doing walk-throughs. Developers might be unaware of the "Procedure Definition" (Shift+F2) feature, which provides a way to go to the code of a called procedure. This is particularly convenient if you like to keep your hands on the keyboard as much as possible.

Simply place the cursor on the procedure call, and press Shift+F2 to get to the procedure's code. When you're done, Ctrl+Shift+F2 takes you back.

—Aseem Pitamber, Haryana, India

VB3, VB4 16/32, VB5

Level: Intermediate

Use VB System Color Constants in API Calls

Visual Basic includes constants, such as vbActiveTitleBar and vbButtonFace, for Windows system colors, which the user might change through the Control Panel. (In VB3, these constants are defined in the file CONSTANT.TXT.) When you assign one of these constants to a VB color property, Visual Basic automatically translates it to the actual color the user has chosen for that item. You cannot, however, use VB's system color constants directly with API functions, such as SetPixel, that expect a color as one of their parameters.

VB's system color constants are the same as those defined by the Windows API, except that VB's constants have the high bit set. You can use this function to translate both VB and Windows system color constants into the corresponding RGB color value, suitable for use in API calls:

```
' 32-bit
Option Explicit
Declare Function GetSysColor Lib "User32" ( _
    ByVal nIndex As Long) As Long
Public Function SysColor2RGB(ByVal lColor As Long) As Long
    lColor = lColor And (Not &H80000000)
    SysColor2RGB = GetSysColor(lColor)
End Function
```

For 16-bit versions of VB, replace the GetSysColor declaration with this code:

```
Declare Function GetSysColor Lib "User" ( _
    ByVal nIndex As Integer) As Long
```

—Steve Cisco, Perrysburg, Ohio

VB3, VB4 16/32, VB5

Level: Beginning

Change ToolTips Background Color

To change the ToolTips background color, open the Control Panel's Display tool. Click on the Appearance tab. In the Item combo box, select ToolTip. Click on the little color box to change the color.

—Rod Stephens, Boulder, Colorado

VB3, VB4 16/32, VB5

Level: Beginning

DOUBLING QUOTES FOR SQL STATEMENTS

This routine parses a string and returns an equivalent string where all the instances of a given substring are doubled. This is especially useful for doubling quotes within SQL statements to be passed to Access or other database engines that expect double quote marks to represent a single quote mark:

```
Public Function DoubleString ( _
     ByVal str As String, _
     ByVal strDoubleString As String) As String
     Dim intStringLength As Integer
     Dim intDoubleStringLength As Integer
     Dim intPosition As Integer
     Dim strTemp As String
     intStringLength = Len(str)
     intDoubleStringLength = Len(strDoubleString)
     strTemp = str
     If intStringLength >= _
          intDoubleStringLength And _
          intDoubleStringLength > 0 Then
          intPosition = 1

          Do While (intPosition > 0) And _
               (intPosition <= intStringLength)
               intPosition = InStr(intPosition, _
                    strTemp, strDoubleString)
               If intPosition > 0 Then
                    strTemp = Left(strTemp, _
                         intPosition - 1 + _
                         intDoubleStringLength) & _
                         strDoubleString & _
                         Mid(strTemp, intPosition + _
                         intDoubleStringLength, _
                         intStringLength)
                    intStringLength = Len(strTemp)
                    intPosition = intPosition + _
                         (intDoubleStringLength * 2)
               End If
          Loop
     End If
     DoubleString = strTemp
End Function
```

—Eric Lynn, Ballwin, Missouri

VB4 32, VB5

Level: Intermediate

DRAW USERS' ATTENTION BY FLASHING WINDOWS' CAPTION

Do you want your programs to call users' attention? Use the API's FlashWindow function. Put this code in a module:

```
Option Explicit
Declare Function FlashWindow Lib "user32" ( _
     ByVal hwnd As Long, ByVal bInvert As Long) As Long

Sub Flash(hFlash As Long, iTimes As Integer, _
     sInterval As Single)
     Dim i As Integer
     For i = 0 To iTimes
          'iTimes sets the number of flashes
          Call FlashWindow(hFlash, True)
          Dim Start As Single
          Start = Timer     ' Set start time.
          ' sInterval sets the time between flashes
          Do While Timer < Start + sInterval
               DoEvents
               ' Yield to other processes
          Loop
     Next i
     ' Puts everything back to normal
     Call FlashWindow(hFlash, False)
End Sub
```

Add this code to a button:

```
Private Sub Command1_Click()
     'Start the fun...
     Flash Me.hWnd, 20, 0.5
End Sub
```

—Iván Márquez, Miranda, Venezuela

VB5

Level: Beginning

USE PICTURES ON COMMAND BUTTONS

To use a picture on a CommandButton, you must set its Picture property and set its Style to 1 - Graphical. Otherwise, the picture will be ignored.

—Rod Stephens, Boulder, Colorado

VB5

Level: Intermediate

PROVIDING AN "INSERT OBJECT" DIALOG FOR THE RICHTEXTBOX

One of the enhancements to the RichTextBox control in VB5 is the addition of an OLEObjects collection, which allows Rich Text documents to include linked or embedded OLE objects. However, the RichTextBox control provides no interactive way for users to insert such objects; they must be added by means of the Add method of the OLEObjects collection.

You can, however, use the Insert Object dialog provided by VB's OLE Container control to allow the user to insert an OLE object into a RichTextBox. Add an OLE Container control (oleObj) to your form, and set its Visible property to False. Then add this code behind a button or menu item:

```
Private Sub cmdInsertObject_Click()
     Dim sClass As String
     ' Show OLE Container control's
     ' Insert Object dialog
     oleObj.InsertObjDlg
     ' If user makes a selection, add it to
     ' RichTextBox's OLEObjects collection
     sClass = oleObj.Class
     If Len(sClass) Then
          rtfText.OLEObjects.Add , , , sClass
          ' Clear OLE Container's object to conserve memory
          oleObj.Class = ""
     End If
End Sub
```

—Basil Hubbard, Hamilton, Ontario, Canada

VB3, VB4 16/32, VB5

Level: Beginning

RUN YOUR VB APP ON STARTUP

Do you have an application you need to launch on Windows startup? Typically, you do this by placing an item in the Startup directory. However, you can also do this with a registry entry. Add a new string value using the name of your application. The value is the path to your executable:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\_
    Windows\CurrentVersion\Run
```

A sample entry might look like this:

```
MyApp        C:\Program Files\MyApp\MyApp.exe
```

—Kevin Gilchrist, Glendale, Arizona

VB5

Level: Beginning

CODE-COMMENTING SHORTCUTS

Instead of going to the Edit menu to comment out a section of code, you can add the comment command to the shortcut menu you access by right-clicking in a code window. Select View/Toolbars/Customize. On the Toolbars tab of the Customize dialog, click on the "Shortcut Menus" check box. On the toolbar that pops up, click on Code Windows... Code Window. On the Commands tab of the Customize dialog, select "Edit" in the categories list box. Drag and drop "Comment Block" and "Uncomment Block" from the "Commands list" box to the code window menu. Hit the "Close" button on the Customize dialog. Now go to a code window, drag the mouse over a section of code, right-click, and select "Comment code."

—Greg Ellis, St. Louis, Missouri

VB3, VB4 16/32, VB5, VBA

Level: Intermediate

IS MY OBJECT VALID?

Have you ever needed to determine if an object variable is holding a valid reference? TypeName() returns a class name if the object holds a valid reference. It returns the word "Nothing" if it does not:

```
Public Function IsObjValid(objTest As Object) As Boolean
    Dim vntResult As Variant
    vntResult = TypeName(objTest)
    If vntResult <> "Nothing" Then _
        IsObjValid = TrueEnd Function
```

—Lee Taylor, Cornelius, North Carolina

VB3, VB4 16

Level: Advanced

CANCEL A PENDING PRINT JOB

In the tip, "Please Stop Printing!" ["101 Tech Tips for VB Developers," Supplement to the August 1997 issue of *VBPJ*, page 13], the code works fine with one exception. If you use PrintManager, anything printed before you press the Abort button is sent to the printer. This code uses the Windows API AbortDoc to prevent this. Put a button named cCancel on the form and caption it "Abort Printing." Add this declaration to your program:

```
Declare Function AbortDoc% Lib "GDI" ( _
    ByVal hDC As Integer)
```

Add this code to the cCancel button's Click event procedure:

```
Sub cCancel_Click ()
    dim i%
    AbortPrintFlag=True
    On Local Error Resume Next
    i = AbortDoc(Printer.hDC)
End Sub
```

Add this code to the print routine:

```
AbortPrintFlag=False
Do While ......
    Printer.Print ......
    DoEvents
    If AbortPrintFlag then exit do
Loop
On Local Error resume next
Printer.EndDoc
```

—Dennis E. Hining, Houston, Texas

VB3, VB4 16/32, VB5, VBA, VBS

Level: Beginning

CLEAN DRIED MARKS ON YOUR WHITE BOARD

If old words or marks have dried on a white board and you can't wipe them off, write on top of them with a fresh pen and then rub them off. This also works if someone has used a permanent pen.

—Rob Parsons, DeeWhy, New South Wales, Australia

VB4 32, VB5

Level: Intermediate

EXPORT SQL DATA TO A COMMA-SEPARATED VALUE FILE

Use this code to create a comma-separated value (CSV) file from a recordset based on a SQL query. A number of applications, including Excel and Access, can read CSV files:

```
Function CSVExport(sSQL As String, sDest As _
    String) As Boolean
On Error goto Err_Handler
    Set snpExport = db.OpenRecordset(sSQL, dbOpenSnapshot)
    Open App.Path & "\" & sDest For Output As #1
    For iLoop = 0 To snpExport.Fields.Count _
         - 1     'Export Field Names
        sFieldText = "" & _
            (snpExport.Fields(iLoop).Name)
        Write #1, sFieldText;
    Next
    Write #1,
    snpExport.MoveFirst
    Do While snpExport.EOF = False
```

```
        For iLoop = 0 To snpExport.Fields.Count - 1
            sFieldText = "" & (snpExport.Fields(iLoop))
            Write #1, sFieldText;
        Next
        Write #1,
        snpExport.MoveNext
    Loop
    CSVExport = True
    Exit Function
Err_Handler
    MSGBOX("Error: " & Err.Description)
    CSVExport=False
End Function
```

—Kevin O'Donnel, Mission Viejo, California

VB4 32, VB5
Level: Intermediate

Close All Open Recordsets and Databases

This small subroutine iterates through and closes all databases in the Databases collection of the Workspaces object. For each database in the collection, it iterates through each Recordset and closes it first:

```
Public Sub DBCloseAll()
Dim DBInstance As Database
Dim RSInstance As Recordset
On Error Resume Next
    For Each DBInstance In Workspaces(0).Databases
'For all open databases....
            For Each RSInstance In DBInstance.Recordsets
'For all open=20
Recordsets....
                RSInstance.Close
            'Close all Recordsets
            Next
    DBInstance.Close
    ' Close DB
 Next
End Sub
```

—Marcus Cedergren, Gothenburg, Sweden

VB3, VB4 16/32, VB5
Level: Intermediate

Retrieve the ID of a Newly Inserted Database Record

The ID (counter field) of a newly inserted record is available immediately after the AddNew method executes. Retrieving the ID at this point means you don't have to mess with the LastModified property and bookmarks. You can clear the variable used to store the ID if you cancel the update. Try this code from a button Click event:

```
With Data1.Recordset
    .AddNew
    MsgBox !ID
    .CancelUpdate
End With
```

—Joe Maki, received by e-mail

VB4 16/32, VB5
Level: Beginning

Return to a Previous Location in Your Code

To return to a previous location in your code, press Ctrl+Shift+F2. This way you don't need to set bookmarks. VB only keeps track of the last eight lines of code that you accessed or edited. This feature is available in design time and in break mode.

—Trish Anders, Peoria, Arizona

VB3, VB4 16/32, VB5
Level: Beginning

A Fast Way to Find Routines

To quickly go to the code for a called function, subroutine, or property, simply place your cursor on the called routine and hit Shift+F2. VB will immediately go to the routine as long as it is defined in your project.

—Fred Meyer, Iowa City, Iowa

VB4 16/32, VB5
Level: Beginning

Force Uppercase Characters in a Text Box

To change text in a text box to uppercase as soon as the user types it in, use this code:

```
Private Sub txtname_KeyPress(KeyAscii As Integer)
        'Puts in uppercase as soon as typed
        KeyAscii = Asc(UCase(Chr(KeyAscii)))
End Sub
```

—Trish Anders, Peoria, Arizona

VB4 32, VB5
Level: Intermediate

Add Columns to a Standard VB List Box

Use the Win32 API to set tab stops in a Visual Basic list box by creating these declarations and routine in a module:

```
Public Const LB_SETTABSTOPS = &H192
Declare Function SendMessage Lib "user32" _
    Alias "SendMessageA" (ByVal hwnd As Long, _
    ByVal wMsg As Long, ByVal wParam As Long, _
    lParam As Any)As Long
Sub SetListTabStops(iListHandle As Long)
' sets 2 tab stops in a list box at the 24th
' and 48th character
' iListHandle = the window handle of the list box
Dim iNumColumns As Long
Dim iListTabs(1) As Long
Dim Ret As Long
    iNumColumns = 2
```

```
    iListTabs(0) = 96  ' 96/4 = 24 characters
    iListTabs(1) = 192 ' 192/4 = 48 characters
    Ret = SendMessage(iListHandle, _
        LB_SETTABSTOPS, iNumColumns, iListTabs(0))
End Sub
```

In your form, create a list box called lstMyListBox. Call this routine in the form load to set your tab stops:

```
Call SetListTabStops(lstMyListBox.hwnd)
```

Then add items to the list box using vbTab:

```
lstMyListBox.AddItem "Column 1 data" & _
    vbTab & "Column 2 data"
```

—Catherine Spence, Hollis, New Hampshire

VB3, VB4 16/32, VB5

Level: Intermediate

Place a Variable-Length String in Text

VB doesn't have a function to replace part of a string (of n-length) with another string (of x-length). The Mid() statement works only with target and replacement strings of the same length. Use this code to solve the problem:

```
Function ReplaceText( _
    ByVal sOriginalString As String, _
    ByVal sTargetString As String, _
    ByVal sReplacement As String) As String
    Dim iOriginalLen As Integer
    Dim iTargetLen As Integer
    Dim iReplaceLen As Integer
    Dim iTargetLoc As Integer
    iOriginalLen = Len(sOriginalString)
    iTargetLen = Len(sTargetString)
    iReplaceLen = Len(sReplacement)
    If (iTargetLen = 0) Or (iOriginalLen = 0) Or _
        (iReplaceLen = 0) Then
        ReplaceText = sOriginalString
'Improper use of function.
    Else
        iTargetLoc = InStr(sOriginalString, sTargetString)
        ReplaceText = Iif( iTargetLoc = 0, sOriginalString, _
            Left(sOriginalString, iTargetLoc - _
            1) & sReplacement & Right( _
            sOriginalString, iOriginalLen - _
            iTargetLoc iTargetLen + 1) )
    End If
End Function
```

—Bruce Goldstein, Highlands Ranch, Colorado

VB3, VB4 16/32, VB5, VBA

Level: Beginning

Put an Ampersand (&) in a Menu or Command Button

When creating menus and command buttons, use the ampersand symbol to create a shortcut key for that option. For example, you might use "&Cancel" for a command button with a "Cancel" button used to cancel an event. To create menu options or command buttons with an ampersand in the caption, set the Caption property (either by coding or through the property settings form), using "&&" to place it in the caption.

—James Kahl, St. Louis Park, Minnesota

VB4 32, VB5

Level: Advanced

Register ActiveX Components Manually

Not all ActiveX DLL and OCX files include installation programs to properly register the control. You need to run REGSVR32.EXE manually to perform the registration. You can save time by creating a shortcut in your WINDOWS\SENDTO folder called "Register ActiveX Control." Use the target name of C:\WINDOWS\REGSVR32.EXE for the shortcut. When you have an ActiveX control you wish to register, right-click on the file name to display the context menu, click on "Send To," and click on "Register ActiveX Control." The RegSvr32 program will display a message box indicating the success or failure of the registration.

—Tony Pappas, Calabasas, California

VB4 16/32, VB5

Level: Intermediate

Destroying Collections' Items

Many programmers mistakenly believe they can remove all the items from a collection by setting the collection itself to Nothing:

```
Dim children as New Collection
...
Set children = Nothing
```

This technique only works if you don't have another object variable pointing to the same collection. In this case, setting the children variable to Nothing doesn't destroy the collection; instead, it decreases its internal reference counter. The only way to remove all the items of the collection is with a loop of Remove methods:

```
Do While children.Count
    children.Remove 1
Loop
```

—Francesco Balena, Bari, Italy