WELCOME TO THE FOURTH 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 for Applications, and Visual Basic Script. The tips were compiled by the editors at *Visual Basic Programmer's Journal*. Special thanks to *VBPJ* Technical Review Board members Doug Haynes, Karl E. Peterson, and Phil Weber for testing all the code. Instead of typing the code published here, download the tips from the 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 415-853-0230 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, indicate if it's for VBA, VBS, VB3, or VB4, 16- or 32-bit version. Please try to limit code length to 20 lines. Don't forget to include your e-mail and mailing address. We'll pay you $25 if we publish your tip.

VB3, VB4 16/32
Level: Beginning

EASY CR/LF PADDING

When I write text in message boxes, labels, and so on, and I need to include a carriage return/line feed, I use this function, passing it the number of CR/LFs I need. This saves a lot of typing and looking up of ASCII values:

```
Sub cmdDelete_Click()

    msg = "Are you sure you want " & _
        "to delete this item?" & NL(2) & "Press OK"
    rc = MsgBox(msg, 4 + 32 + 256, "Confirm Delete")
    ...
end sub

Function NL(num_lines As Integer) As String
'****************************************
'   This function returns a New Line
'   character for the number of times
'   passed to the function.
'****************************************

    Dim msg As String
    Dim i As Integer

    For i = 1 To num_lines
        msg = msg & Chr(13) & Chr(10)
    Next

    NL = msg
End Function
```

—Bret Cutler, Layton, Utah

VB3, VB4 16/32
Level: Beginning

AUTOSELECT TEXT BOX CONTENTS

Users often find it faster to retype the entire contents of a text box rather than position the cursor within the data, and then edit the data. This is especially true if the length of the data is short or if the data isn't visible, as with a password field, for example. Double-clicking or using a mouse to select the control's contents is slow and inconvenient.

I created this small routine to automatically select all data within a control. I place this routine in a code module so that it's accessible from all forms. I call this routine from a control's GotFocus event. This way, the data is selected if the user tabs to or clicks on the control, or if a data validation routine does a SetFocus:

```
Private Sub MyTextBox_GotFocus()
    AutoSelect MyTextBox
End Sub
```

The AutoSelect routine is quite simple:

```
Sub AutoSelect(SelObject As Control)
' The AutoSelect routine "selects" the
' control's entire contents as if it were
' doubled-clicked.
    SelObject.SelStart = 0
    If TypeOf SelObject Is MaskEdBox Then
        SelObject.SelLength = Len(SelObject.FormattedText)
    Else
        If TypeOf SelObject Is TextBox Then
            SelObject.SelLength = Len(SelObject.Text)
        End If
    End If
End Sub
```

—Kevin Forth, St. Clair Shores, Michigan

VB4 16/32
Level: Beginning

CLEAR THE CLUTTER

Provide your users with a quick way to clear their "messy desktop" of extraneous forms by dropping this code into the Click event of a command button:

```
For Each Form In Forms
        If Form.Name <> Me.Name Then
            Unload Form
        End If
    Next Form
```

—James Bell, Charlotte, North Carolina

VB3, VB4 16/32
Level: Beginning

CALCULATE AGE USING DATEDIFF

Use the function DateDiff to calculate an individual's exact age based on birth date. DateDiff first calculates the total number of days an individual has been alive and then divides by 365.25 to account for leap years. The Int function truncates the division results by removing the decimal and not rounding:

```
Function CalcAge(datEmpDateOfBirth as Variant) as Integer
    CalcAge = Int(DateDiff("y",datEmpDateOfBirth,Date())_
        /365.25)

End Function
```

—Michael Finley, Clarendon Hills, Illinois

VB4 16/32
Level: Intermediate

PREVENT UNWANTED RECURSION

A bug affecting the Sheridan 3D Controls appears in VB 4.0a. I don't believe it was in 4.0, and I am certain it was not in 3.0. In any case, even if you use 4.0, you should pay attention because the newer OCX may be installed already on users' machines when your apps are distributed. I use the Sheridan command buttons quite a bit for their ability to display an icon as well as text, and I ran into this problem when I installed VB 4.0a.

If the user double-clicks on a common dialog box (for example, to select a file), and the double click is physically located above a Sheridan 3D command button, the Click procedure of that command button will be fired. To test this, start a new project and place a Sheridan command button and a common dialog control on Form1. In the Click procedure of the command button, include this code:

```
Private Sub SSCommand1_Click()
    Const CDERR_CANCEL = &H7FF3
    CommonDialog1.DialogTitle = "Open File"
    CommonDialog1.filename = "*.*"
    CommonDialog1.DefaultExt = ".*"
    CommonDialog1.CancelError = True
    CommonDialog1.Filter = "All Files (*.*)|*.*"
    On Error Resume Next
    CommonDialog1.Action = 1
    If Err = CDERR_CANCEL Then
        Exit Sub
    End If
    On Error GoTo 0
End Sub
```

Run the program, click on the command button, and when the common dialog appears, move it so that the name of any file appears over the button. Double-click on the file name. The common dialog will disappear and a new one will appear, resulting from the SSCommand1_Click event firing again.

The solution is to declare a static "flag" variable, FalseClick, within the button's Click event. Then change the code in the SSCommand1_Click procedure to read:

```
Private Sub SSCommand1_Click()
    Const CDERR_CANCEL = &H7FF3
    Static FalseClick As Boolean
    If FalseClick Then Exit Sub
    FalseClick = True
    CommonDialog1.DialogTitle = "Open File"
    CommonDialog1.filename = "*.*"
    CommonDialog1.DefaultExt = ".*"
    CommonDialog1.CancelError = True
    CommonDialog1.Filter = "All Files (*.*)|*.*"
    On Error Resume Next
    CommonDialog1.Action = 1
    If Err = CDERR_CANCEL Then
        FalseClick = False
        Exit Sub
    End If
    On Error GoTo 0

    DoEvents 'allow for recursion
    FalseClick = False
End Sub
```

The DoEvents function is required in any procedure that is so short that the last statement executes before the recursion begins. I hope this workaround can help others fix the same problem.

—Michael P. Rose, Kingwood, Texas

VB3
Level: Intermediate

OBTAINING SYSTEM DATE FORMAT

The DateCheck function is useful when you use the Masked Edit control for entering the date according to the system format. Some countries use the MM/DD/YYYY format, and other countries, like India, use the DD/MM/YYYY format. Instead of hard-coding the Masked Edit controls format as well as mask properties, use this function to set the required date format according to the control panel settings.

In the General declaration section of the module, define this:

```
Declare Function GetProfileString Lib _
    "Kernel" ( Byval Sname$, ByVal Kname$, Byval Def$, _
    Byval Ret$, Byval Size% ) as integer
global datemask as variant
global dateformat as variant
    Sub DateCheck()
        Dim Strsecname as string
        Dim Strkeyname as string
        Dim Varsuccess as variant
        Dim Strretdate as string
        Dim Strretsep as string
        Dim Strchar as string * 1
    Strsecname = "Intl"
    Strkeyname = "sShortDate"
    Strretdate = string$(11,0)
    Varsuccess = GetProfilestring_
        (Strsecname,Strkeyname,"",Strretdate,_
        Len(Strretdate))
    Strsecname = "Intl"
    Strkeyname = "sDate"
    Strretsep = String$(2,0)
    Varsuccess = GetProfileString_
        (Strsecname,Strkeyname,"",_Strretsep,Len(Strretsep))
    Strretsep = Left$(strretsep,1)
    Strchar = Ucase$(Left$(strretdate,1))
    datemask = "##" & Strretsep & "##" & Strretsep & "####"
        Select case strchar
            case "D" : dateformat = "DD" _
                & strretsep & "MM" & strretsep & "YYYY"
            case "M" : dateformat = "MM" _
```

```
              & strretsep & “DD” & strretsep & "YYYY"
          case "Y" : datemask = "####" _
              & Strretsep & "##" & strretsep & "##"
              dateformat = "YYYY" & _
                  strretsep & "MM" & strretsep & "DD"
    End select
End Sub
```

In the Form_Load event, call the DateCheck procedure to get the current date format, and assign format and mask properties of the Masked Edit control:

```
    Sub Form_Load()
    DateCheck
    ' Call the datecheck procedure
    Mskdd.format = dateformat
    Mskdd.mask = datemask
End Sub
```

—S. Saravanan, Selaiyur, Madras, India

VB4 32
Level: Intermediate

LONG FILE NAMES CAN BE CONFUSING

If you want to open Paint with a file from your application, it’s better to convert the path of the file you want to open from long to short names. Doing so is wise because in some situations—if your path contains spaces, for example—Paint may refuse to work properly. Before passing a file name to Paint, convert it to short names with this routine, which takes advantage of the Win32 API:

```
Declare Function GetShortPathName Lib "kernel32" Alias _
    "GetShortPathNameA" (ByVal lpszLongPath As String, _
    ByVal lpszShortPath As String, _
    ByVal cchBuffer As Long) As Long

Function ShortName(LongPath As String) As String
    Dim ShortPath As String
    Const MAX_PATH = 260
    Dim ret&
    ShortPath = Space$(MAX_PATH)
    ret& = GetShortPathName(LongPath, ShortPath, MAX_PATH)
    If ret& Then
        ShortName = Left$(ShortPath, ret&)
    End If
End Function
```

This trick may prove useful with any application you pass file names to. It would be smart to try passing “strange” file names/paths to make sure.

—Andrea Nagar, Torino, Italy

VB3, VB4 16/32
Level: Beginning

“FILE EXISTS?” REVISION

To adapt the tip “File Exists?” (see “101 Hot Tech Tips for VB Developers,” Supplement to the August 1996 issue of *VBPJ*, page 7) to a directory list box, it is necessary to control the error that occurs when the user selects the root (C:\):

```
Sub Dir1_Change
    Dim File As String
    File = Dir1.Path
    If Right$(File, 1) = "\" Then
        File = File & "himem.sys"
    Else
        File = File & "\" & "himem.sys"
    End If
    ' Chuong Van Huynh’s tip
    If Dir$(File) <> "" Then
        MsgBox "himem.sys exists !"
    End If
End Sub
```

Without testing to see if the path already ends with a backslash, as it would when it’s the root directory, an error occurs because of the string “c:\\himem.sys.”

—Michel Rohan, Menthonnex Sous Clermont, France

VB3, VB4 16/32
Level: Intermediate

TRIMMED TO FIT

This piece of code trims long file names to fit into forms captions, text boxes, and other limited spaces. The code allows you to specify the number of characters a file name must be before it performs the trimming. For example, if the label can hold 50 characters, then you would type LongDirFix(nFile,50). It’s as simple as that. Here’s the code:

```
Function LongDirFix(Incomming As _
    String, Max As Integer) As String
Dim i As Integer, LblLen As Integer, StringLen As Integer
Dim TempString As String
TempString = Incomming
LblLen = Max
If Len(TempString) <= LblLen Then
    LongDirFix = TempString
    Exit Function
End If
LblLen = LblLen - 6
For i = Len(TempString) - LblLen To Len(TempString)
    If Mid$(TempString, i, 1) = "\" Then Exit For
Next
LongDirFix = Left$(TempString, 3) + _
    "..." + Right$(TempString, Len(TempString) - (i - 1))
End Function
```

—Shafayat Kamal, Wallington, New Jersey

VB3, VB4 16/32
Level: Beginning

USE RLES TO REDUCE EXE SIZE

Many programs use a splash screen to display the logo of the program. The logo is usually made by an image. You used to store it as a BMP (uncompressed) in an image box. Perhaps you don’t know that you also can put an RLE image (compressed bitmap—PaintShop Pro supports this format) within VB4. Select the image, click on All Files, and select your RLE file. The size of your EXE will decrease and your app may load faster. There’s a trade-off, however, in the fact that it will be smaller to read from disk, but it will consume more time as it’s decoded to the screen.

—Andrea Nagar, Torino, Italy

VB4 16/32
Level: Beginning

WHERE DOES IT END?

When using the line-continuation character facility—the combination of a space followed by an underscore (_) used in the development environment to extend a single logical line of code to two or more physical lines—it can be difficult to determine where the sentence begins and ends.

One easy way to avoid this difficulty is to set a breakpoint in the desired line. The line will be set to the breakpoint color, becoming an obvious code line. To remove the breakpoint from the Run menu, choose Toggle Breakpoint (F9) again.

To set a breakpoint, position the insertion point anywhere in a line of the procedure where you want execution to halt. From the Run menu, choose Toggle Breakpoint (F9). The breakpoint is added, and the line is set to the breakpoint color defined in the Editor tab of the Options dialog box.

—Jose Alberto Marques da Silva, Coimbra, Portugal

VB3, VB4 16/32
Level: Beginning

CHEAP GRAPHIC BUTTONS

You can use the Wingdings font to put simple graphics on a standard command button. Put a standard command button on a form, and in the Properties window, change the button's font to Wingdings. Load the Character Map application that comes with Windows, and change the font to Wingdings. Select the picture you want and copy it to the clipboard. Now change back to VB and select the Caption property for the command button in the Properties window. Paste the new character in the Caption property (you can also use the Keystroke that is shown at the bottom of the character map window). You can make the picture bigger by changing the font size. This method is useful if you don't need color graphics and don't want the additional overhead of a 3-D command button. Be forewarned that it's possible your user might have removed this font from his or her system, and this could cause unexpected runtime errors.

—David Moulton, Knoxville, Tennessee

VB3, VB4 16/32
Level: Intermediate

"REMEMBER SWAP?" CORRECTION

The tip "Remember SWAP?" ["101 Hot Tech Tips for VB Developers," Supplement to the August 1996 issue of *VBPJ*, page 13] has a common error. This line actually creates three variants and one string, rather than the four strings desired:

```
Dim a, b, c as string * 4
```

The line should read:

```
Dim a as string * 4
Dim b as string * 4
Dim c as string * 4
```

—Timothy J. Hoffmann, El Segundo, California

VB4 16/32
Level: Intermediate

SHELLING MS INTERNET MAIL

If you are using the new mail program that can be downloaded for free from Microsoft's Web site, you can run it from within your program. Simply add this command to a command button:

```
X = Shell("C:\WINDOWS\EXPLORER.EXE /root,C:\WINDOWS\ _
    Internet Mail.{89292102-4755-11cf-9DC2-00AA006C2B84}", 3)
```

Note that for this example, the Windows directory name is hard coded. For a production app, call the GetWindowsDirectory API and use that return value instead.

—Fred Lyhne, Salt Spring Island, British Columbia, Canada

VB4 16/32
Level: Intermediate

REFER TO COLUMNS IN A DBGRID CONTROL

Columns in a DBGrid control are bound to database fields. However, the index number of a column doesn't tell you which field it represents. Therefore, referring to DBGrid1.Columns(0).Value is not very descriptive. It's better to introduce variables with clear names, which are bound to specific columns. In the declarations section of the form, for example, the code reads:

```
Dim ColOrder_ID As Column
Dim ColArticle_ID As Column
Dim ColAmount As Column
```

In the Form_Load event of the form, the code reads:

```
With DBGrid1
    Set ColOrder_ID = .Columns(0)
    Set ColArticle_ID = .Columns(1)
    Set ColAmount = .Columns(2)
End With
```

It's no longer necessary to refer to DBGrid1.Columns(0).Value, for example, but you can use ColOrder_ID.Value instead. ColOrder_ID is, of course, the same as DBGrid1.Columns(0). If any property value of DBGrid1.Columns(0) changes, the same property value of ColOrder_ID changes accordingly, and vice versa.

Later on, when you insert a database field into the grid, you change only the index numbers of the columns in the code of the Form_Load event.

—George van der Beek, Nieuw Lekkerland, The Netherlands

VB4 32
Level: Intermediate

MAKE COLUMNHEADERS THE PERFECT WIDTH

When you're dynamically adding ColumnHeaders to a ListView control at run time, you may not know how long the text for the header will be, so the user must readjust the width of the column to see it. But by making a label with its Visible property set to False, and its Autosize property set to True, you can fill up the

For even more tricks and tips go to
http://www.windx.com

label with the same text that's going to be in the header. Then use Label1.Width in the Add argument for the ColumnHeader:

```
Private Sub Command1_Click( )
    Dim ColumnText as String, clmx as ColumnHeader
    ColumnText = _
        "A very long header for " & "the ListView control"
    Label1.Caption = ColumnText
    set clmx = ListView1.ColumnHeaders.Add_
        (, , ColumnText, Label1.Width)
    ListView1.View = lvwReport
End Sub
```

—Rich Wigstone, Hoffman Estates, Illinois

VB3
Level: Beginning

HIDDEN MDI CHILDREN

"Hide" an MDI child form with a non-sizable border style (0 or 1) using this code in the MDI child's Form_Load event:

```
form1.Move form1.Left, form1.Top, 0, 0
```

Use a label control covering the visible area of the form to allow switching on and off:

```
If form1.Height = 0 Then
    form1.Move form1.Left, form1.Top, _
        form1!Label1.Width, form1!Label1.Height
Else
    form1.Move form1.Left, form1.Top, 0, 0
End If
```

The form remains loaded but invisible and is immediately available when required. The label control should consume no extra resources. This method provides a quick popup window within MDI.

—Ross Gourlay, Edinburgh, Scotland

VB3, VB4 16/32
Level: Intermediate

DISPELLING PERFORMANCE MYTHS

• Omitting the counter from a Next statement does not speed up a For loop.
• Calling a procedure in a BAS module is not slower than calling a procedure contained in the same FRM module. The only exception is the first call to any procedure in a BAS module because VB loads BAS modules on first reference.
• Because VB is interpreted, the overhead of interpretation far outweighs the time it takes to execute the actual code that does the work. Wrong. References to control properties, for example, can be two to three orders of magnitude slower than references to simple variables.

—Pat Dooley, Cleveland, Ohio

VB3, VB4 16
Level: Advanced

GET/PUT ARRAYS TO DISK

The API functions "_hwrite" and "_hread" can be useful for writing arrays directly to files and, at some later time, reading these files directly into arrays having the same structure. They are fast and easy to use:

```
'this API function is used to write arrays to binary files:
Declare Function hwrite Lib "Kernel" Alias "_hwrite" _
    (ByVal hFile As Integer, _
    hpbBuffer As Any, ByVal cbBuffer As Long) As Long
'this API function is used to read arrays from binary files:
Declare Function hread Lib "Kernel" Alias "_hread" _
    (ByVal hFile As Integer, _
    hpbBuffer As Any, ByVal cbBuffer As Long) As Long
```

The next routine writes a two-dimensional integer array ("integer_array") to a binary file ("binary_file") with the API function "hwrite." The array can later be read back from the file with a nearly identical routine calling the "hread" API function. It is straightforward to modify this procedure for other sizes and types of arrays (except for arrays of user-defined types and variable-length strings), but it cannot be made generic over any number of array dimensions because of the way these API functions are called:

```
Sub WriteIntegerArrayToFile_
    (ByVal binary_file As String, _
    integer_array() As Integer)
Const INTEGER_BYTE_SIZE = 2
Dim binary_file_handle As Integer, _
    dos_file_handle As Integer
Dim bytes_written As Long, bytes_to_write As Long
    'get the size of the array in bytes:
    bytes_to_write = _
        (UBound(integer_array, 1) - _
        LBound(integer_array, 1) + 1) _
        (UBound(integer_array, 2) - _
        LBound(integer_array, 2) + 1) _
        INTEGER_BYTE_SIZE
    'open the file in binary mode:
    binary_file_handle = FreeFile
    Open binary_file For Output As binary_file_handle
    dos_file_handle = _
        FileAttr(binary_file_handle, 2)
    'make the API call:
    If dos_file_handle <> 0 Then
        bytes_written = _
            hwrite(dos_file_handle, integer_array ( _
            LBound(integer_array, 1), _
            LBound(integer_array, 2)), _
            bytes_to_write)
    End If
    Close binary_file_handle
End Sub
```

—Dave Doknjas, Surrey, British Columbia, Canada

VB4 32
Level: Intermediate

Debugging Using the Query TopValues Property

I speed up the debugging of applications with queries that return large record sets by using Access 95 to temporarily set the TopValues within the query stored in the MDB. VB4 cannot get at the TopValues property of a stored query. This method allows the query to run in context without any artificial tests in VB to reduce its output.

Open the MDB with Access 95. Select the Queries tab. Highlight the query. Click on Design, then right-click. Select Properties and click again. Change the TopValues property and save the query design. To reverse the change, set TopValues to All.

—Stan Mlynek, Burlington, Ontario, Canada

VB3
Level: Beginning

Fixed Dialog Forms in VB3

A VB3 form that has a fixed double border does not show the form's icon in the upper left-hand corner if you have the control box set to True. Here's a way to have a double border (3-D dialog border) form that sports a visible icon as well as the standard close "X" in the upper right-hand corner, similar to the VB4 Fixed Dialog form. Choose the form you want and set BorderStyle to 2 - Sizable, ControlBox to True, and MaxButton and MinButton to False, and set the form's icon to your application's ICO file. Place this code in the form events:

```
(general) (declarations)
Dim MeHeight As Integer
Dim MeWidth As Integer

Sub Form_Load ()
    MeHeight = Me.Height
    MeWidth = Me.Width
End Sub

Sub Form_Resize ()
    Me.Height = MeHeight
    Me.Width = MeWidth
End Sub
```

When the user tries to resize the form, the sizing rectangle will be visible, but the form will snap back to its former size.

—Ken Zinn, Miamisburg, Ohio

VB3, VB4 16/32
Level: Intermediate

Null Handling in Database I/O

Nulls are still a problem in VB3/VB4—they cause unexpected errors. To handle them, you can read the MDB field into a variant, which can hold a null and then test for it later, or replace all nulls with blank strings to protect the application.

To me, nulls serve no logical purpose. I prefer to eradicate them immediately. I have developed a family of I/O pick-up routines that I use at the physical interface next to the database field:

```
Public MyDB As Database
Public MyRS As Recordset
Dim TestVariant As Variant
Set MyDB = OpenDatabase("testjet3db")
Set MyRS = MyDB.OpenRecordset("NameTable")
'Sample Calls
TestVariant = ScreenForNull(MyRS![FirstName])
TestVariant = ScreenForNull(MyRS![FirstName], " ")
Public Function ScreenForNull(aField As Variant, Optional _
    ByVal DefaultReturn As Variant) As Variant
    If IsNull(aField) Then
        If Not _
            IsMissing(DefaultReturn) Then
            ScreenForNull = DefaultReturn
        Else
            ScreenForNull = ""
        End If
    Else
        ScreenForNull = aField
    End If
End Function
```

—Stan Mlynek, Burlington, Ontario, Canada

VB3, VB4 16/32
Level: Advanced

Automatically Upgrade Workstation EXEs

I'm programming for a LAN and quite often I add requested features to the program. The LAN is set up so that each workstation is running its own copy of the program and is only reading/writing data files on the server. This arrangement has significantly increased the startup speed of the program; however, when the EXE file is changed, all the workstation programs must be changed. I get around having to go to each station with this program:

```
Private Sub Form_Load()
    On Error GoTo errorhandler
    ' the Command function Returns the
    ' argument portion of the command
    ' line used to launch Microsoft
    ' Visual Basic or an executable
    ' program developed with Visual Basic.
    ' ie:(thisprog.exe c:\localdir\prgcopied.exe
    ' k:\servrdir\prgtocopy.exe)
    If FileDateTime(Left(Command$, _
        InStr(Command$, " ") - 1)) < _
        FileDateTime(Mid$(Command$, _
        InStr(Command$, " ") + 1)) Then
        ' case the file does not exist
        ' locate the form designed to your pref.
        Top = (Screen.Height - Height) / 2
        Left = (Screen.Width - Width) / 2
        ' containing a label
        label1 = "Copying " & Mid$(Command$, InStr_
            (Command$, " ") + 1) & _
             " to your hard_disk..."
        ' make the form it visible so the
        ' operator has something to look
        ' at while the program is copied
        Visible = True
        Refresh
        ' copy file as per parameters in Command$
        FileCopy Mid$(Command$, _
            InStr(Command$, " ") + 1), _
            Left(Command$, InStr(Command$, " ") - 1)
    End If
    'start the program
    x = Shell(Left(Command$, InStr(Command$, " ") - 1), 3)
    End
    Exit Sub
errorhandler:
    If Err = 53 Then 'file not found
        Resume Next 'copy the file anyway
    Else 'trap other errors
        MsgBox "Error # " & Err & Chr(10) & Error _
        & Chr(10) & "program will be terminated"
        End
    End If
    Exit Sub
End Sub
```

—Fred Lyhne, Salt Spring Island, British Columbia, Canada

VB3, VB4 16/32

Level: Beginning

AUGMENTING ERROR$

In many places, I include Error$ as picked up in an error trap in a descriptive message. I use a function to expand its meaning. I call the function TSS for Time Stamped String to keep it short. It works something like this:

```
MsgBox TSS(Error$) & " in Mytest"
Public Function TSS (ByVal aString As String)
        If Len(aString) Then
            TSS = aString & " at " & CStr(Now)
        Else
            TSS = "Error$ is Blank at " & CStr(Now)
        End If
End Function
```

—Stan Mlynek, Burlington, Ontario, Canada

VB3, VB4 16/32

Level: Intermediate

CALLING ON WORD

Microsoft Word exposes the WordBasic object. Through this object, you can execute WordBasic statements. The WordBasic statements and functions can be used as methods of the WordBasic object. Most WordBasic method names match the menu selection available in Word, and the parameters match the dialog items:

```
'Declare form level
Dim wd As Object
Sub CreateWordObject( )
Set wd = CreateObject ("Word.Basic")
wd.FileNewDefault
wd.FontSize 20
wd.Insert "Hello, World"
wd.FileSaveAs "Hello.Doc"
wd.FileClose
```

If you are familiar with Microsoft Word, you know that each method name corresponds to the menu name concatenated by submenu option names, and the functionality is the same as in Word. You can also use the Word Macro recorder to create code and then copy that code to your Visual Basic application.

—Shikha Arora, Detroit, Michigan

VB3

Level: Intermediate

PRINTING PROBLEM WITH WIN95

Microsoft is currently working on a printing problem. Visual Basic 3.0 applications running in Windows 95 cannot print to shared printers with an embedded space. If a shared printer has an embedded space in the computer or printer name, the Visual Basic application will generate an Error 482.

To work around this problem, change the computer or shared printer name to a name without an embedded space, or create a local printer and redirect the output to the shared printer with an embedded space (see Windows 95 online help).

Add this code to your Visual Basic 3.0 applications to check for the error. Warning your users with a polite message box is more tolerable than an Error 482 message. This code assumes that you have declared the Windows API functions GetVersion and GetProfileString:

```
Function Win95PrintBug ()
    Dim LongReturn As Long, IntegerReturn As Integer, _
        PortName As String
    Dim StringReturn As String * 256, _
        NullString As String * 256
    LongReturn = GetVersion ()
    If LongReturn = 0 Then
        Win95PrintBug = False
        'Code for failed GetVersion call
    Else
        LongReturn = LongReturn And &HFFFF&
            'Mask-Off upper two bytes
        If LongReturn = (95 * 256) + 3 _
            Then ' (Minor Version * 256) _
            ' + Major Version Code for Windows 95
            ' operating system
            IntegerReturn = GetProfileString ("windows", _
                "device", NullString, StringReturn, 256)
            StringReturn = Left$ _
                (StringReturn, IntegerReturn)
            StringReturn = Right$ _
                (StringReturn, Len(StringReturn) - _
                InStr(StringReturn, ","))
            PortName = Trim$ (Mid$ (StringReturn, InStr_
                (StringReturn, ",") + 1))
            If InStr(PortName, " ") <> 0 Then
                Win95PrintBug = True
                ' VB 3.0 cannot print to this port
            Else
                Win95PrintBug = False
                ' VB 3.0 can print to this port
            End If
        Else
            Win95PrintBug = False ' Operating
                ' system is not Windows 95
        End If
    End If
End Function
```

For more information, check out the Microsoft Knowledge Base article Q130650.

—Clinton D. Hess, Ocoee, Florida

VB4 16/32
Level: Beginning

MANAGE FOCUS WITH MDI TOOLBARS

While developing MDI applications in VB 4.0, you probably have noticed that the cursor on the MDI child disappears when you click on the toolbar on the MDI parent. You cannot navigate through Tab or any other key. To avoid this problem, put this one line of code in the MouseUp event of every Toolbar object:

```
Private Sub tbMain_MouseUp(Button As Integer, Shift _
    As Integer, X As Single, Y As Single)
    me.setfocus
End Sub
```

—Rajeev Madnawat, Sunnyvale, California

VBA, VB4 16/32
Level: Intermediate

HARD-LOCK A TABLE

In many applications, I want to make absolutely sure that the data in a Jet table doesn't get modified under any circumstances. I hard-lock the table in addition to using any system-level security to protect it. Also, this hard lock stays with the MDB if it is issued with an application.

Place the expression True=False into the ValidationRule property of the table to lock it. The Jet evaluates this expression to False and blocks updates to the table:

```
'Declarations
Public MyDB As Database
Dim Dummy As Integer
'Sample calls
Dummy = HardLockTable("UnLock", "TestTable")
Dummy = HardLockTable("Lock", "TestTable")
Function HardLockTable_
    (ByVal whichAction As String, _
    ByVal aTable As String) As Integer
On Error GoTo HardLockTableError
'Default return
HardLockTable = True
Select Case whichAction
Case "Lock"
    MyDB.TableDefs(aTable).ValidationRule = "True=False"
    MyDB.TableDefs(aTable).ValidationText = _
        "This table locked via " & _
        "ValidationRule on " & Now
Case "UnLock"
    MyDB.TableDefs(aTable).ValidationRule = ""
    MyDB.TableDefs(aTable).ValidationText = ""
Case "TestThenUnLock"
    If MyDB.TableDefs(aTable)._
        ValidationRule = "True=False" Then
            MyDB.TableDefs(aTable).ValidationRule = ""
            MyDB.TableDefs(aTable).ValidationText = ""
    End If
End Select
HardLockTableErrorExit:
        'subFlushDBEngine
        'optional, see next suggestion
Exit Function
HardLockTableError:
        HardLockTable = False
        MsgBox Error$ & " error " & _
            "in HardLockTable trying " & _
            "to " & whichAction & " " & _
            aTable
        Resume HardLockTableErrorExit
End Function
```

—Stan Mlynek, Burlington, Ontario, Canada

VB4 16/32
Level: Intermediate

NEW FEATURE ALLOWS DATA STRUCTURES TO BE MORE COMPLEX

VB4 allows you to create data structures that are more dynamic than previous versions allowed, thanks to the possibility of having dynamic arrays within types. For example, a tree structure with three levels can be created with user-defined types and dynamic arrays:

```
Private Type LeafType
    'we'll let the leaves be integers
    Leaf As Integer
End Type
Private Type BranchType
    'a branch is an array of leaves:
    Branch() As LeafType
End Type
'a tree is an array of branches:
Private Tree() As BranchType
```

This next code sets up a tree with three branches. The first branch has three leaves, the second branch has two leaves, and the third branch has five leaves:

```
ReDim Tree(1 To 3)    'three branches
ReDim Tree(1).Branch(1 To 3)
'three leaves on the first branch
ReDim Tree(2).Branch(1 To 2)
'two leaves on the second branch
ReDim Tree(3).Branch(1 To 5)
'five leaves on the third branch
```

You can add other fields to allow parts of the tree other than the leaves to store data.

—Dave Doknjas, Surrey, British Columbia, Canada

VB3, VB4 16/32
Level: Intermediate

REGARDING "VALIDATING NUMERIC INPUT"

The tip you published on page 70 of the September 1996 issue of *VBPJ* on "Validating Numeric Input" was a good tip. However, the function failed to take into account the use of the comma (,) as the decimal separator. The United States uses the period, but Europeans tend to use the comma. To correct this problem, use the value of the "sDecimal" variable from "win.ini."

—David S. McBride, Scottsdale, Arizona

For even more tricks and tips go to
http://www.windx.com

VB3, VB4 16/32
Level: Intermediate

CODE ALL KEYPRESS EVENTS IN ONE MOVE

Do you need to tab through all those edit boxes? Are you tired of coding all of their KeyPress events? If so, set the tab indexes as you normally would. Then set the form key preview to True. In the form KeyPress event, enter this code:

```
If KeyAscii = 13 Then
    SendKeys "{Tab}"
    KeyAscii = 0
End If
```

This will tab to all the controls, so if you don't want to tab to a command button, set its tab stop to False.

—Mark Patenaude, Santee, California

VBA, VB3, VB4 16/32
Level: Intermediate

TEST FOR LEAP YEAR

The tip "Leap Year Rules" ["101 Hot Tech Tips for VB Developers," Supplement to the August 1996 issue of *VBPJ*, page 22] illustrated how to easily determine if any given year is a leap year. I believe I can offer a simpler, smaller function that does the same thing.

Simply pass in the year you are testing, append that year to 02/29/, and use the IsDate function to see if that is a valid date. If 02/29/xx is not a valid date, then you know it is not a leap year:

```
Function IsLeap(sYear As String) As Integer
    If IsDate("02/29/" & sYear) Then
        IsLeap = True
    Else
        IsLeap = False
    End If
End Function
```

—Michael Willeson, Tea, South Dakota

VB3
Level: Intermediate

PRINT A SINGLE SUB OR FUNCTION

Need a fast way to print only one sub or function in VB3 that also takes care of the problem some printers have in printing the last page?

Create a small form with one command button. Change the button's Caption to Print Sub. Add this code to the button's Click event:

```
Sub Command1_Click ()
    'Make sure VB has the focus.
    AppActivate "Microsoft Visual Basic"
    'Clear the Clipboard just in case.
    Clipboard.Clear
    'Select the Sub or Function and copy
    'it to the Clipboard.
    SendKeys "^{HOME}" & "^+{END}" & "%EC", True
    'Print the Clipboard
    Printer.Print Clipboard.GetText ()
    'Form Feed in case printer needs it
    'to print the last page.
    Printer.NewPage
    'End the print job.
    Printer.EndDoc
    'Unmark the Sub or Function.
    SendKeys "{HOME}", True
End Sub
```

Move the form to a spot you want it to appear in when running. Compile it to an EXE file. Run it along with Visual Basic. When you want to print a sub or function, make sure the cursor is in the Sub or Functions code box, and click on the Print Sub button.

Printer.EndDoc called immediately after Printer.NewPage ensures that no blank page is printed.

—Kenneth L. Creel, Lynden, Washington

VBA, VB3, VB4 16/32
Level: Intermediate

A FAST WAY TO CHANGE CASE

Do you need to convert user input to all upper case in VB3? All the ways I have seen to force user input to all upper case is to make calls to the Chr and UCase functions in the variant or string form, in addition to calling the Asc function.

Why not use integers only, eliminating calls to these functions? This method is easy because the only difference between lower and upper case letters is bit five. The number 223 conveniently has all bits except bit five:

Lower case a	= 01100001
Upper case A	= 01000001
Number 223	= 11011111

```
Sub AllUppers (KeyAscii As Integer)
    'If KeyAscii is in range, then
    'perform bit-wise comparison and assign result.
    If KeyAscii > 96 And KeyAscii < 123 Then KeyAscii = _
        (KeyAscii And 223)
End Sub
```

If you have multiple text controls needing this input, put the Sub in a code module, and call it from the KeyPress event of the Text controls whenever you need it.

Similarly, 32 = 00100000, so to convert from upper to lower case, use:

```
If KeyAscii > 64 and KeyAscii < 91 Then KeyAscii = _
    (KeyAscii Or 32)
```

—Kenneth L. Creel, Lynden, Washington

VBA, VB3, VB4 16/32

Level: Intermediate

TABLE CHECKER

I use a housekeeping routine to check the state of tables during system initialization. The routine tests whether the table is in a collection, whether the table has data, and whether the table is local or attached. The routine helps to establish the state of the application at startup.

For example, a table name that appears in the Tables collection can be either attached or local. If it is attached and the second MDB where it resides is not present, this state is only discovered when the application first attempts to access it. Usually, an error condition occurs in the middle of application logic. I prefer to test for this condition at the beginning as opposed to being surprised later on. I perform a trial read on the attached table:

```
Public MyDB As Database
Dim Dummy As Integer
Set MyDB = OpenDatabase("testjet3db")
'Sample call
Dummy = CheckTable("NameTable", "readarecord", "local")

'More sophisticated call to trial read an attached table
If Not CheckTable("AnotherTable", _
     "readarecord", "attached") Then
     'we've got trouble in River City
End If
Function CheckTable(ByVal whichTable As String, _
     ByVal whichTest As String, _
     ByVal whichAttach As String) As Integer
Dim ErrorStage As String
Dim aTable As Recordset
' This is a comprehensive table checker
' which tests for:
'    .is the table in the collection
'    .is it attached or local as the case may be
' NOTE: have found subtle differences in
' how DAO Find FindNext
'         commands behave wrt attached and
'         local tables
'          message boxes are useful in
'         debugging, can be removed
On Error GoTo CheckTableError
     'Set Default return condition
     CheckTable = True
     ErrorStage = " Test for table in collection "
     If Not CheckIfTableInCollection(whichTable) Then
          CheckTable = False
          MsgBox ErrorStage & whichTable _
               & " failed in CheckTable"
          Exit Function
     End If
     ErrorStage = "SetTable"
     Set aTable = MyDB.OpenRecordset(whichTable)

     ErrorStage = "Test read a record "
     If whichTest = "readarecord" Then
          aTable.MoveFirst
     End If
     ErrorStage = "Test attachment status "
     Select Case whichAttach
          Case "attached"
               If InStr(MyDB.TableDefs(whichTable).Connect, _
                    "DATABASE=") = 0 Then
                    CheckTable = False
                    MsgBox ErrorStage & whichTable & _
                         " failed " & whichAttach & _
                         " in CheckTable"
               End If
          Case "local"
               If MyDB.TableDefs_
                    (whichTable).Connect <> "" Then
                    CheckTable = False
                    MsgBox ErrorStage & whichTable & _
                         " failed " & whichAttach & _
                         " in CheckTable"
               End If
          Case "dontcare", ""
     End Select
CheckTableErrorExit:
     If ErrorStage <> "SetTable" Then
          aTable.Close
     End If
     Set aTable = Nothing
     Exit Function
CheckTableError:
     CheckTable = False
     MsgBox Error$ & " ErrorTrap " & _
          ErrorStage & "  " & whichTable _
          & " failed in CheckTable"
     Resume CheckTableErrorExit
End Function
Function CheckIfTableInCollection_
     (ByVal TableName As String) _
     As Integer
' This function checks that the tables
' collection has a table
' it looks for tables by name and
' returns true/false setting
Dim i%
     CheckIfTableInCollection = False
     For i% = MyDB.TableDefs.Count - 1 _
          To 0 Step -1
          DoEvents
          If TableName = Trim$(MyDB._
               TableDefs(i%).Name) Then
               CheckIfTableInCollection = True
          End If
     Next i%
End Function
```

—Stan Mlynek, Burlington, Ontario, Canada

VB3, VB4 16

Level: Intermediate

MANIPULATE THE HEIGHT OF THE LIST-BOX ITEM(S)

These two routines enable you to get the height of an item of a list box in terms of pixels, and they also enable you to set the height of all the list-box items to a given value in pixels:

```
Declare Function SendMessage Lib _
     "User" (ByVal hWnd As Integer, ByVal wMsg As Integer, _
     ByVal wParam As Integer, lParam As Any) As Long
Const WM_USER = &H400
Const LB_GETITEMHEIGHT = (WM_USER + 34)
Const LB_SETITEMHEIGHT = (WM_USER + ee)
Const WM_SETREDRAW = &HB
'Height is returned in terms of pixels
Function ListBox_getItemHeight (1st as Control) As Integer
     'Retrieve the height of the listbox
```

```
    'item
    ListBox_getItemHeight = SendMessage ((1st.hWnd), _
        LB_GETITEMHEIGHT, 0, &)
End Function
'Sets the height of the items of the
'listbox to a specified one.
Sub ListBox_setRowHeight (1st As Control, 1Height As Long)
    Dim ignore as integer
    ignore = SendMessage ((1st.hWnd), LB_SETITEMHEIGHT, 0, _
        ByVal 1Height)
    'Refresh the listbox
    ignore = SendMessage ((1st.hWnd), WM_SETREDRAW, True, 0&)
End Sub
```

—Vishwanath Khasarla, Waltham, Massachusetts

VBA, VB3, VB4 16/32
Level: Intermediate

QUICK TEST FOR WEEKEND

To test whether a date falls on a weekend, you might be inclined to do this:

```
nDay = weekday (sDate)
If (nDay = 1) or (nDay = 7) Then
    'It's a weekend
End If
```

However, you can use VB's MOD operator to perform the test in roughly half the time:

```
If (Weekday (sDate) MOD 6 = 1) Then
    'It's a weekend
End If
```

—Phil Weber, Tigard, Oregon

VB4 16/32
Level: Intermediate

SHORTCUT FOR 16/32-BIT DEVELOPMENT

Here is a tip that should make life simpler for all VB programmers doing mixed 16/32-bit development in VB4 under Windows 95. I have three additional items in my project menu. These items, Open Win16, Make Win16 EXE File, and Run Win16 Project, allow me to develop in either 16- or 32-bit versions of VB4 at the click of a mouse button.

To get this result, start REGEDIT and find the subkey VisualBasic.Projects under HKEY_CLASSES_ROOT. Then add three subkeys under this key. These keys are Open Win16, Make Win16 EXE File, and Run Win16 Project.

Under each of these keys, add another subkey and name it Command. Now set the value of Command. For Win16, set the value to "c\vb\vb.exe %1." Note that REGEDIT automatically adds the opening and closing quotes. For Win16 EXE File, set the value to "c:\vb\vb.exe /m %1." For Run Win16 Project, set the value to "c:\vb\vb.exe /r %1." These paths assume that the 16-bit version of VB4 is installed under c:\VB. If you have installed 16-bit VB4 in another folder, substitute its path for c:\vb\vb.exe. Now you can conveniently open, run, and make EXE files in either Win16 or Win32.

—Lalit Bhargava, Edmonton, Alberta, Canada

VB4 32
Level: Intermediate

CREATE NUMERIC-INPUT TEXT BOXES

In almost every application's project you come across, fields necessitate the use of numeric text boxes—text boxes that should accept only numeric values. You can deal with this situation by either using a Masked Edit control or by writing code in the KeyPress event procedure of every text box. Here is an alternative method for accomplishing the same objective that can be used with the 32-bit version of VB4. The advantage of this method is that it eliminates the need to write code in every text box. Also, because this method is applicable to the basic text-box behavior of control in VB4, you don't need to worry about the specific behavior of the Masked Edit control.

To see how this method works, load a new project and insert a new module, "Module1," into it. Now insert these statements in the declarations section of Module1:

```
Declare Function SetWindowLong Lib "user32" Alias _
"SetWindowLongA" (ByVal hwnd As Long, ByVal nIndex As Long, _
    ByVal dwNewLong As Long) As Long
Declare Function GetWindowLong Lib "user32" Alias _
    "GetWindowLongA" (ByVal hwnd As Long, _
    ByVal nIndex As Long) As Long
Sub NumericEdit(TheControl As Control)
    Const ES_NUMBER = &H2000&
    Const GWL_STYLE = (-16)
    Dim x As Long
    Dim Estyle As Long
    EStyle = GetWindowLong(TheControl.hwnd, GWL_STYLE)
    EStyle = EStyle Or ES_NUMBER
    x = SetWindowLong(TheConrol.hwnd, GWL_STYLE, EStyle)
End Sub
```

Next, insert a new form into the project. Add a text box, Text1, to this form and place this code in the Form_Load event procedure:

```
Call NumericEdit(Text1)
```

**—Vinit Budhiraja
Fountain Valley, California**

VB4 16/32
Rank: Beginning

DON'T FORGET YOUR REFERENCES

If you get the error "User-defined type not defined" on a line where you are declaring a database, go to Tools, then go to References, and select the appropriate DAO library.

—John Muller, Woodstock, Georgia

VB3, VB4 16
Level: Intermediate

SET CURSOR TO CONTROL BY DEFAULT

Sometimes it is useful to set a cursor to determine control when a form is loading—after the form is done loading, the cursor stays on the control you have ordered:

```
Type pointapi
    x As Integer
    y As Integer
End Type
'Sets the mouse cursor position in
'screen coordinates
Declare Sub SetCursorPos Lib "User" _
    (ByVal x As Integer, ByVal y As Integer)
'Converts client point to screen coordinates
Declare Sub ClientToScreen Lib "User" _
    (ByVal hWnd As Integer, lpPoint As pointapi)

Sub SetCursorToDefaultControl (Control As Control)
Dim Pnt As pointapi
Dim x As Integer
Dim y As Integer
Pnt.x = Pnt.y = 0
'Determine coordinates left top corner of Control
Call ClientToScreen(Control.hWnd, Pnt)
x = Pnt.x + Control.Width/ _
    (2 * (Scree.ActiveForm.Left + Control.Left)/Pnt.x)
y = Pnt.y + Control.Height/ _
    (2 * (Screen.ActiveForm.Top + Control.Top)/Pnt.y)
Call SetCursorPos(x, y)
End Sub
```

—Aleksandr Dvigubskiy, Brooklyn, New York

VB4 16/32
Level: Beginning

ADD NEW FILE TYPE INTO THE REGISTRATION ON THE FLY

In reference to "Unregister a DLL with the Right Mouse Button in Win95" [*VBPJ* March 1996, page 72], I would like to suggest a tip that is much easier to use and edit. Any Win95 or NT 4.0 user can easily customize this tip.

The tip given in the March issue had restrictions. It has no edit facility because the lines have to be coded and not all users can understand the code. Using Windows Explorer, you can easily perform all the operations.

First, open Windows Explorer and select Options in the View menu. You will see the tab, which has two options—View and FileTypes. Select the FileTypes tab, where you can see all the registered file types. They have three buttons—NewType, Remove, and Edit.

If you want to register an OCX file type, press the NewType Button. You will see an AddNewFileType dialog. Next, specify the Description of the file type, Associated extension, and Actions. Enter the description as "OCX File." In the Associated extension, type "OCX." Suppose you want to add a "Register" action—press the New button, and you get the NewAction dialog.

Next, specify the action name and browse to select the application file, which will perform the given action or give the command line with full path. Here you give the action name as "Register" and press Browse to select the regsvr32.exe file. Normally, Regsvr32.exe resides in the Windows directory. Now the entry looks like "C:\Windows\Regsvr32." Press OK to save the action. Then, for Unregister, give the action name as "Unregister" and press Browse to select the regsvr32.exe file. Add the parameter /u for unregister. Now the entry looks like "C:\Windows\Regsvr32 /u." Remember, when adding a parameter, the path name should follow the MS-DOS path convention. By pressing the right mouse button on any OCX file listed in the Explorer, you will see the Register and Unregister option. Select the appropriate action to perform. The same procedure can be done for any file type.

Using this option of Explorer, you can edit any existing action on any file type shown in the list, as well as remove the actions or the whole entry for the selected file type.

—System Builders International, New Dehli, India

VB3, VB4 16/32
Level: Beginning

SCROLLABLE VIEWPORT FOR A PICTURE

It's easy to make a scrollable viewport for a picture in both VB3 VB4. In addition to a picture, you can also use other objects that grow. First, create a new form. Place a picture box on the form, name it "picParent," and resize it. This will be the viewport. Place a picture box inside the other picture box (picParent) and name it "picPICTURE." The value of the Left and Top properties must be zero. This one will contain the picture you want to view.

Next, place a vertical scrollbar on the right of the picParent Picture. Name it "vsbPict." This scrollbar should have the same Top and Height values as picParent. Do the same with a horizontal scrollbar. Place it on the bottom of "picParent"—the Left and Width values should be equal to picParent. Name it "hsbPic." In both scrollbars, set the LargeChange property to the height and width of picParent, respectively. Then place this code on the declarations section of Form1. You can also copy this code to the Scroll event:

```
Private Sub hsbPIC_Change()
    picPicture.Left =hsbPIC.Value
End Sub
Private Sub vsbPIC_Change()
    picPicture.Top =vsbPIC.Value
End Sub
Private Sub picPicture_Resize()
    If picPicture.Height > picParent.Height Then
        vsbPIC.Max =picParent.Height - picPicture.Height
    Else
        vsbPIC.Max = 0
    End If
    If picPicture.Width > picParent.Width Then
        hsbPIC.Max =picParent.Width - picPicture.Width
    Else
        hsbPIC.Max =0
    End If
        vsbPIC.Value = 0
        hsbPIC.Value = 0
End Sub
```

Every time you load a new picture, the scrollbars will adjust to its size. Write the code to load the pictures, and write other code to deal with exceptions to the rules.

—Joel Paula, Carcavelos, Portugal

For even more tricks and tips go to
http://www.windx.com

VB3, VB4 16/32
Level: Beginning

AVOID THE VALIDATION ROUTINE WHEN THE USER PRESSES CANCEL

When a user presses the Cancel button, the control that previously had focus triggers its LostFocus event, which is where many programs do their validity checking. If the input wasn't filled, as is often the case when the user clicks on Cancel, the validation may display a message. The user won't expect this after clicking on Cancel, and shouldn't ever see it.

To avoid this problem, add a form-level variable to indicate if Cancel has been clicked on. In the Cancel's MouseDown event, set it to True. In the MouseUp event, set it back to False—this way the LostFocus event occurs between the two.

One more catch—if the user activates the Cancel button with a mouse-down click on the Cancel button, changes his or her mind, and moves the mouse off the Cancel button and then "mouse-ups," you still have it set to Cancel. So set it to False in the MouseMove event of the parent of the Cancel button. To avoid doing validations, check against your Cancel Pressed variable.

—Jed Walker, Denver, Colorado

VB3, VB4 16/32
Level: Beginning

CLEAN AND COMPACT YOUR CODE

The last thing you should do before creating your final EXE is clean your code. This tip explains how to clean the garbage from your project that is sometimes left behind from moving procedures in and out of different project files. Doing so will make your project files smaller and, in some cases, eliminate possible GPFs due to code being left behind by only deleting a pointer to the code. This routine works for forms and module files.

Start Visual Basic and load your project. Make sure that all your code windows are closed. Highlight the first file in your project window. Then select Save Text from the File menu. Click on OK to save the file in text format. Highlight the same file in the project window. Click on the View Code button in the project window. Select Load Text from the File menu. Find and highlight the previously saved file (FormName.txt). Click on Replace. Finally, close the code window. Repeat the same steps for each file in the project and then save your project.

This procedure will remove the transparent dead code left over from deleting code or moving procedures in and out of different project files. You will be able to see the results in the size of your project files after you run this routine for each file in your project and save your project. This cleaning can also prevent GPFs that are caused sometimes by this transparent dead code. This routine does not remove code that is moved to the general declaration section of a file because you changed a control's name. It removes old data you thought you deleted, but you actually deleted only a reference pointer to it. The code is still there, but it is transparent in the code windows.

—Stefan Klein, Ridgeland, Mississippi

VB4 16/32
Level: Beginning

AUTOMATICALLY RESIZE CONTROLS

Sometimes you need to change the size of controls by changing the size of a container. For example, try this:

```
Private Sub Form_Resize()
     Picture1.Align = vbAlignLeft
     Picture1.Align = vbAlignTop
End Sub
```

Every sizing event that changes the form's size also changes the size of the picture box. Make sure that the control you want to resize has the align property.

—Uwe Pryka, Bochum, Germany

VB4 16/32
Level: Intermediate

"ACCESSING SECURED DATABASES FROM VB4," REVISITED

The tip, "Accessing Secured Databases from VB4" [*VBPJ* September 1996, page 81], is not entirely accurate. There are several ways to secure an Access database. If the database itself is protected with a password, but no user access rights or passwords have been assigned, then the method described will not work when using the DAO3032.

In order to access an Access95 database protected with a password using the DAO3032, the workspace must be created with a blank password and user name. Then the database must be opened with a blank administrator, but with this parameter: PWD=[password]. This method opens the database properly without assigning users particular passwords. Generally, this is useful when a front end accesses a database located in a distributed manner, where the user access is actually stored in the MDB itself.

—David Schneider, Phoenix, Arizona

VB4 16/32
Level: Intermediate

EFFECTIVE ERROR HANDLING

While working with classes in VB 4.0, we found that the errors raised from our classes were not being trapped at the form level. We used "On Error Resume Next," but still the error was being raised in the class module. We changed the settings of the Advanced Tab from Options in the Tools menu to "Break Unhandled Error," and it worked as we needed.

So, if you want to raise an error from a class module in your application to the form, you must use the Err.Raise method to raise the error. The error will be handled in the form only if you set the option from Options in the Tools menu in the Advanced Tab to "Break On Unhandled Error." Otherwise, the error is raised in the class itself because the default setting is "Break On Class Module."

—System Builders International, New Delhi, India

VB3, VB4 16/32
Level: Intermediate

STORED ACCESS QUERIES SPEED RESPONSE TIME

In general, stored Access queries are faster than embedded SQL in your VB application. It can be a great benefit to be able to send parameters to the query directly from VB. Saving these queries in Access also reduces the amount of code you must maintain directly in your VB app. These code segments show how to send a parameter to a stored Access query. Here is the example Access SQL query:

```
PARAMETERS ID Text;
SELECT DISTINCTROW Host.IPAddress
FROM Host
WHERE Host.ID=[ID];
```

This code retrieves an IP address from an Access table, where the host ID in the table matches the host ID in the parameter. This code is stored in the Access database and has already been parsed and optimized. This is the key to gaining the performance benefits of stored queries. The Jet engine then uses this QueryDef when it is called from VB.

Assume that there is a form with a data control on it. Here is the corresponding VB code to send the parameter:

```
Dim db As Database
Dim rs As Recordset
Dim qd As QueryDef
Set db = dthosts.Database
'dthosts is a data control on a form
Set qd = db.QueryDefs("aq_host")
'aq_host is the MS Access query name
qd.Parameters("ID") = "MY_Host_ID"
'host ID is the parameter for this query
Set rs = qd.OpenRecordset()
'open the recordset
Set dthosts.Recordset = rs
'return the recordset to the data control
```

You can now use this data control in conjunction with a listbox control (or whatever control you like) to display the data. You can also add multiple parameters by adding more "qd.parameters" to the middle section of this code.

Insert, update, and delete queries (or "action queries," as they are called in Access) can use parameters in the same fashion.

—Al Gehrig, Jr., Laguna Hills, California

VB4 32
Level: Intermediate

A BETTER SUBSTITUTE FOR GETMODULEUSAGE

To synchronously shell an application from a 16-bit VB application, some people write code like this:

```
Const HINSTANCE_ERROR% = 32
Dim hInstChild As Integer
'Shell program, if Shell worked, enter loop
hInstChild = Shell(strExeName, intCmdShow)
If hInstChild >= HINSTANCE_ERROR Then
    While GetModuleUsage(hInstChild)
        DoEvents
    Wend
End If
```

This code relies on a 16-bit-only API function, GetModuleUsage. Several 32-bit workarounds have been published, but here is the simplest and most reliable solution I've seen:

```
Declare Function OpenProcess Lib "Kernel32" _
    (ByVal dwDesiredAccess As Long, _
    ByVal bInheritHandle As Long, _
    ByVal dwProcessId As Long) As Long
Declare Function WaitForSingleObject Lib "Kernel32" _
    (ByVal hHandle As Long, _
    ByVal dwMilliseconds As Long) As Long
Const SYNCHRONIZE = &H100000
Const INFINITE = &HFFFFFFFF
Private Function SyncShell(ByVal pathname As String, _
    windowstyle As Integer) As Boolean
Dim ProcessID As Long
Dim ProcessHandle As Long
' In VB4, an error occurs if Shell
' fails to start the program
On Error GoTo SyncShell_Error
' Shell the program, get its handle,
' and wait for it to terminate
ProcessID = Shell(pathname, windowstyle)
ProcessHandle = OpenProcess(SYNCHRONIZE, True, ProcessID)
WaitForSingleObject ProcessHandle, INFINITE
SyncShell = True
Exit Function
SyncShell_Error:
    On Error GoTo 0
    SyncShell = False
    Exit Function
End Function
```

—Bob Voges, Woburn, Massachusetts

VB3, VB4 16/32
Level: Beginning

PROTECT YOUR SCREEN SAVER FROM RESTARTING

When you create a screen saver in Visual Basic, you may sometimes utilize the command-line options such as "/s" to start a screen saver directly or "/c" to run the configuration. By simply using App.PrevInstance, your application may not detect the running screen saver and attempt to start another when the normal Windows screen saver time-out occurs. To ensure that Windows will detect the running screen saver, regardless of the use of command-line options, use this code:

```
Private Declare Function GetModuleHandle% Lib "Kernel" _
    (ByVal lpModuleName$)
Private Declare Function GetModuleUsage% Lib "Kernel" _
    (ByVal hModule%)
Sub StartUp()
Dim Inst%
    Inst = GetModuleHandle(App.Path + _
        "\" + CStr(Trim(App.EXEName) + ".scr"))
    If GetModuleUsage_
        (GetModuleHandle(App.Path + "\" _
        + CStr(Trim(App.EXEName) + ".scr"))) > 1 Then End
End Sub
```

For even more tricks and tips go to
http://www.windx.com

Call the sub like this:

```
Sub Main()
    Call StartUp
End
```

This method works well with Windows 3.11. You can use this basic idea with any application you write, not just screen savers.

—Uwe Pryka, Bochum, Germany

VB3, VB4 16/32
Level: Intermediate

USE AN ICO FILE AS A MOUSE POINTER

If you want to use an ICO file as a mouse pointer, set the MousePointer property to 99-Custom. Then load the icon into the MouseIcon property:

```
Text1.MouseIcon= LoadPicture("c:\vb\icons\elements_
    \earth.ico")
```

For the icon to appear as a mouse pointer, both of these properties must be set.

—Prashanti Doma, Inkster, Michigan

VB4 16/32
Level: Intermediate

OLE AUTOMATION VERSUS STANDALONE MODE

There you are, creating an OLE server in Visual Basic. You go to Options in the Tools menu, select the Project tab, choose "OLE Server," and click on OK. After compiling, you run the executable. Is it an OLE server? Unfortunately, it's not.

What makes a VB4 application an OLE server? You need at least one class module whose Public property is set to True and whose Instancing property is set to either "Creatable SingleUse" or "Creatable MultiUse."

At this point, however you still do not have an OLE server: you have an application that is OLE Automation-enabled. If you were to create an instance of the exposed class via OLE Automation from another application, then it would be an OLE server. If the application is run normally, it is a standalone application.

What is the purpose of the StartMode option in the Project options? Is this strictly for the design environment? When you are testing an OLE server with two copies of Visual Basic running, the one with "OLE Server" chosen can be started and will continue running even if no form ever loads. This feature allows the second copy of VB to test OLE Automation against the first. In addition, the first creates a temporary registry entry so that early binding may be tested as well.

On another note, don't forget about the StartMode property of the App object. You can determine whether the app was started as a standalone or as an OLE server. Excel, for example, makes use of this by showing itself if it is a standalone, or by hiding itself if it is run via OLE Automation.

—Peter W. DeBetta, Cary, North Carolina

VB3, VB4 16/32
Level: Intermediate

ODBC ERROR DISPLAY

I have seen many posters in the VB newsgroups that say they have received an ODBC error message such as 3146, but can't figure out why they got the message. I suspect that the vast majority of these posters did not update their error-handling procedures to use the DBEngine.Errors collection when VB4 was released. Using this collection usually gives them all the error messages they need to resolve the problem. I have created an error-handling subroutine that displays the correct error message based on whether the error is a DB error:

```
Public Sub ShowError()
Dim sError As String
Dim nI As Integer
Dim sTitle As String
    sError = ""
    ' Determine whether or not this is a
    ' database error
    If DBEngine.Errors.Count > 0 Then
        If DBEngine.Errors(DBEngine.Errors.Count - 1)._
            Number = Err.Number Then
            sTitle = "Database Error"
            For nI = 0 To DBEngine.Errors.Count - 1
                sError = sError & DBEngine.Errors(nI) & _
                    vbCrLf
            Next
            sError = sError & vbCrLf
        End If
    End If
    If sError = "" Then
        sTitle = "Error"
        ' add the error string
        sError = sError & Err.description & vbCrLf
    End If
    ' beep and show the error
    Beep
    MsgBox sError, , sTitle
End Sub
```

—Karl Costenbader, Sacramento, California

VB4 16/32
Level: Beginning

DIRECTORY REFERENCES IN CODE PROFILER

The Code Profiler does not properly handle relative directory references in the VBP file. If the file looks like this, Code Profiler aborts trying to process the CRW45 file:

```
Module=PASSMAN1; PASSMAN.BAS
Module=CRW45; ..\CM4\CRW45.BAS
```

Code Profiler does not understand the context of the file reference. Before running the Profiler, change all the "…" type relative references to full references:

```
Module=PASSMAN1; PASSMAN.BAS
Module=CRW45; C:\DEV\CM4\CRW45.BAS
```

—Timothy J. Hoffmann, El Segundo, California

VB3, VB4 16/32
Level: Beginning

MAKE TWO (OR MORE) LIST BOXES FOLLOW EACH OTHER

Suppose you have two list boxes side by side—one to display the name of a person, the other to display his or her e-mail address. If the boxes contain several entries, you want both boxes to scroll at the same time. For example, when the user uses the scrollbar of the Names list box, the lines of the e-mail list box remain at the level of the corresponding Names list-box lines.

Unfortunately, the Click event of list boxes doesn't let you monitor the use of its scrollbar. The solution is to use a Timer in conjunction with the TopIndex property of your list boxes to compare and adjust the level of both boxes frequently.

Set the Interval property of the Timer to 50, and type this simple line of code in the Timer event:

```
Private Sub Timer1_Timer()
     If List1.TopIndex <> List2.TopIndex Then _
          List2.TopIndex = List1.TopIndex
End Sub
```

Disable List2 so that the user is forced to use only the scrollbar of List1, but still sees the information displayed in List2.

—Eric Bernatchez, Montreal, Quebec, Canada

VB3, VB4 16/32
Level: Beginning

TRACK MODEM USE BY OTHER APPLICATIONS

With applications increasingly using modems, it is quite possible that users will start your application, forgetting that another application is already using the modem—like a fax machine, for example.

The MS Comm object that comes with VB has a PortOpen property that returns the state of the modem. However, this property seems to work only when the modem is opened by the VB application in which the command is issued. Otherwise, PortOpen returns False even if, for example, a fax machine is running and is holding the modem port open to monitor incoming faxes. If you try to open the port, it will cause an error message.

The only solution is to make an error-handling routine that detects if the port is open by provoking the error and handling it. In the handling routine, insert a MsgBox that tells the user to check for other applications and waits for an answer before resuming.

—Eric Bernatchez, Montreal, Quebec, Canada

VB3, VB4 16/32
Level: Intermediate

USE THE MS DRAW APPLET TO DRAW LINES AND SHAPES

Rather than using Visual Basic's line and shape objects when drawing complex graphic backgrounds on forms, use the MS Draw applet included with most Microsoft products to draw a Windows Metafile (WMF) graphic. Then paste the graphic into the form or image box.

Because there are fewer objects on the form, it loads faster, is less cluttered during development, and the embedded WMF graphic rescales with no fuss if you need to resize the form. MS Draw also supports more graphical objects, such as arcs and free-form objects, than Visual Basic supports.

To paste a WMF graphic into a form, select Drawing from the Insert menu of a Microsoft application, such as Works. MS Draw will start. Draw the background graphic, select all the objects by using the Select All command on the Edit menu, and copy them. In Visual Basic, select the form or image box, and paste the graphic into it.

—Frank Olivier, Jr., Jan Kempdorp, South Africa

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

GET OBJECT NAMES OUT OF JET COLLECTIONS AND INTO VB CODE

To maintain sanity, I use long descriptive names for tables and queries in Jet MDB files. This leads to errors when I type these names into VB code. To overcome this problem, I open the MDB using the appropriate version of Access. I then highlight the name I want in its collection.

Edit Rename (Access 95) or File Rename (Access 2) brings up a rename window. I highlight the long name and copy it to the Clipboard using the Ctrl-C key combination. Edit Copy won't take because the rename window is modal. I cancel the rename window and then paste the long name into the VB code.

—Stan Mlynek, Burlington, Ontario, Canada

VB3, VB4 16/32
Level: Advanced

SPEED TIPS

Reading data into and out of a VB program is straightforward. Most folks use the Input and Put statements in a For…Next loop. However, this straightforward technique takes time. Because I am impatient when it comes to machines—and that includes computers—I use a time-tested technique to speed along the input and output of data files. The technique involves the use of strings.

I use strings because Basic-type programs, including VB, make efficient use of strings. One distinct reason for using strings is that a specific size is not needed for the array definition. This provides some programming flexibility if you work with files of different lengths and formats. Also, and more to the point, data is input and output faster when the Input and Put statements move strings into and out of computer programs.

Speed is the big advantage of using strings in conjunction with the Input and Put statements, in lieu of embedding the statements in a For…Next loop. The speed increase is obvious with a few simple illustrations. Start with this program listing:

```
Open Path$ For Binary As #1
     Datarray$ = Input(31680, #1)
     Workarray$ = Datarray$
Close
```

This listing inputs a string named "Datarray" and copies that string to a work string named "Workarray." I do this to maintain the integrity of the original data. As you can see, the code opens, inputs, copies, and closes a complete file in just four computer instructions.

For comparison, look at a typical code listing to read this file. This is the simplified code listing:

```
Open Path$ For Binary As #1
     For Position = 1 to 31680
          Datarray$(Position) = Input(1,#1)
     Next Position
Close
```

This example inputs data with a For…Next loop. Although the number of programming lines may be relatively small, the number of computer instructions is large. To input the same file with a For…Next loop, the For, Next, and Input statements each need to be executed 31,680 times. That's a whole heap of instructions when compared to executing the Input statement just once. The speed implication is obvious.

My programming philosophy to output data follows the same technique—eliminate data loops where possible. This example shows how this is easily accomplished without a data loop:

```
Open Path$ For Binary As #1
     Put #1,,Workarray$
Close
```

Path$ is the output location and name of the file. Workarray is the string that holds the data to save.

In summary, it is important to limit the length of For…Next and Do loops in any language. Long loops for I/O are real speed killers.

—Donald L. Parrish, Huntsville, Alabama

VB3, VB4 16/32
Level: Intermediate

Is Mail Running?

Use this code to quickly determine whether Microsoft Mail is currently running:

```
Declare Function GetModuleHandle Lib _
     "Kernel" (ByVal lpModuleName As String) As Integer
Function IsMicrosoftMailRunning ()
On Error GoTo IsMicrosoftMailRunning_Err
IsMicrosoftMailRunning = GetModuleHandle("MSMAIL")
IsMicrosoftMailRunning_Err:
     If Err Then
          'Do whatever you need to here
     End If
End Function
```

You can also use this method to determine if just about any particular program is running. Simply replace the Mail references with the appropriate name you are looking for.

—Calvin Smith, San Francisco, California

VB4 16/32
Level: Intermediate

Use a Picture Control to Create a Beveled Panel

Occasionally, you may want to use on your form a beveled panel that allows you to designate a background bitmap. Rather than use a third-party control, you can use the Picture control provided as a core DLL function (non-OCX) in all versions of Visual Basic 4.

Place a Picture control on your form, and set the Appearance property to "1 - 3D" and the BorderStyle property to "1 - Fixed Single." This creates the bevel effect. Now set the Picture property to the desired image. You may now place controls directly on the picture control. Remember to set the "Background Style" property to "0 - Transparent" for all controls where you would prefer the background image to be displayed.

—Marc Mercuri, Nashua, New Hampshire

VB4 16/32
Level: Intermediate

VB 4.0 Add-In Error Messages

The ability to write add-ins under VB 4.0 is a significant addition to the developer's toolbox. However, writing and distributing add-ins can be fraught with many a mind-boggling experience. One of these is dealing with the mysterious message, " 'Add-In Name' could not be loaded. Remove it from the list of available Add-Ins?"

My experience has shown at least two causes for this error. The first reason is obvious. As in all good programs, you must prepare for any and all contingencies. Such preparation is even more necessary in an add-in. If your add-in encounters an untrapped error in the Connect event, the error is passed back to VB and the error message is displayed with no further explanation of what occurred. The obvious correction is to program for any and all errors.

The second reason for the error is not so obvious. Registration problems can be particularly hectic for add-ins. The normal procedure for registering an out-of-process add-in (EXE) is to execute it, say from the File Manager. The add-in includes code to register itself with VB (VB.INI entry), and VB itself registers the add-in (OLE server) in the Reg.Dat file. However, if you later move the add-in to another directory, and reexecute it, the error message can occur again because of conflicting entries in the Reg.Dat file.

I recently discovered REGCLEAN.EXE (32-bit) and REGCLN16.EXE (16-bit) that ship with VB 4.0 in the Tools\PSS Directory of the CD. After two years of installing and removing numerous software packages from my development system, my Reg.Dat file had become so large that REGEDIT would not run because of a lack of memory. I ran REGCLN16 on my Windows 3.1, and it found 729 errors in my Reg.Dat file. After I set it to automatically clean up the file, I could run REGEDIT again and the problem with "unable to load add-in" disappeared.

—Les Smith, Concord, North Carolina

VB3, VB4 16/32
Level: Intermediate

Copying Menu Objects Between Forms

One problem that occurs frequently is taking the menu structure from one form and placing it in another. Although my method of dealing with this dilemma requires a little effort, it does work well. Because forms are saved as text files (always in VB4, optionally in VB3), you can open up a form in Notepad or any other text editor. An excerpt may look something like this:

```
Begin VB.Menu mnuEdit
    Caption  =   "&Edit"
    Begin VB.Menu mnuEditItem
        Caption  =   "Cu&t"
        Index    =   0
        Shortcut =   ^X
    End
    Begin VB.Menu mnuEditItem
        Caption  =   "C&opy"
        Index    =   1
        Shortcut =   ^C
    End
    Begin VB.Menu mnuEditItem
        Caption  =   "&Paste"
        Index    =   2
        Shortcut =   ^V
    End
End
```

This is a simple Edit menu with cut, copy, and paste options. You can copy this section, for example, and paste it into another form you have open in Notepad. Words to the wise: back up the files first (always be safe), and don't randomly paste the section in the recipient form. Make sure it is between other objects defined there. For example, insert the menu where the asterisk is:

```
    Top  =   -30
    Width    =   7215
End
*  Begin MSComDlg.CommonDialog CMDialog1
        Left=   0
        Top  =   0
```

—Peter W. DeBetta, Cary, North Carolina

VB3, VB4 16/32
Level: Intermediate

Find the Name of the Windows or Windows System Directory

You can quickly find the name of the Windows or the Windows System directory. Just pass "WindowsDirectory" or "WindowsSystemDirectory" into this function:

```
Declare Function wu_GetWindows_
    Directory Lib "Kernel" Alias _
    "GetWindowsDirectory" _
    (ByVal lpBuffer As String, _
    ByVal nSize As Integer) As Integer
Declare Function wu_GetWindowsSystem_
    Directory Lib "Kernel" Alias _
    "GetSystemDirectory" _
    (ByVal lpBuffer As String, _
    ByVal nSize As Integer) As Integer
Function WinDirs$ (strDirNameNeeded$)
On Error GoTo WinDirs_Err
Dim strBuffer$
Dim iSize%
Dim iResult%
iSize = 256
strBuffer$ = Space$(iSize)
    Select Case strDirNameNeeded$
        Case "WindowsDirectory"
            iResult% = wu_GetWindows_
                Directory(strBuffer$, iSize)
        Case "WindowsSystemDirectory"
            iResult% = wu_GetWindows_
                SystemDirectory(strBuffer$, iSize)
    End Select
    WinDirs$ = Left$_
        (strBuffer$, iResult%)
WinDirs_Err:
    If Err Then
        ' Do whatever you need to here
    End If
End Function
```

—Calvin Smith, San Francisco, California

VBA, VB4 16/32
Level: Intermediate

Substitute the Array Function for the DATA Statement

My first encounter with Visual Basic (VB1) was a letdown because I had become accustomed to using the original Basic "DATA" statement for loading numeric values. The statement was also useful for testing programs that processed sequential data streams. Unfortunately, both QBasic and Visual Basic didn't/don't support it. I must have researched the manuals for hours before giving up in frustration.

The VB4 Array function allows an argument list to be assigned to a variant. The data can be retrieved as if it were being read sequentially. For anyone who needs to convert old style DATA statements to VB4, these code snippets may be of use:

```
'Declarations
Option Base 1
Public MyData As Variant
Public Sub Main()
    'Like the old DATA statement
    MyData = Array(1.1, 1.2, 1.3, 1.4, 1.5, _
        1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9)
    'Test the results
    Debug.Print MyRead(MyData), MyRead(MyData)
End Sub
Public Function MyRead(ByVal whichVariant)
Static intI As Integer
    intI = intI + 1
    MyRead = whichVariant(intI)
End Function
```

—Stan Mlynek, Burlington, Ontario, Canada

VBA, VB3, VB4 16/32
Level: Beginning

CREATE QUERIES ON THE FLY

Use VB's string-concatentation operator (&) to incorporate user input into SQL queries at run time. Assume, for example, that cboField is a combo box containing a list of fields in a table, and txtValue is a text box into which the user types a value to search for. Here's how to create a dynamic query that includes the user's input:

```
sField = cboField
sValue = txtValue
SQL = "SELECT * FROM Table WHERE " & _
     sField & " LIKE '" & sValue & "';"
```

Note that you must surround the search value with quotes if you are searching a text field; no quotes are necessary if the target field contains numeric data.

—Claudia Vega Cachú, Tepic, Nayarit, Mexico

VBA, VB3, VB4 16/32
Level: Beginning

ENTERING SCIENTIFIC FORMULAS

I have developed this technique to use when I program long formulas. First, I declare descriptive variables for the mathematical elements. I prototype the formula using the digit "1" as a place holder for the variables. This allows me to review the structure of the formula and make sure it compiles. I keep a copy of the formula as an inline comment.

I replace the place holders with the variable by using cut and paste. This is an example that calculates the volume of a cap of a sphere:

```
Public Const PI = 3.141592653589
Dim CapVolume As Single
Dim SphereRadius As Single
Dim CapHeight As Single
'Sample values
SphereRadius = 9.35
CapHeight = 3.33
'Formula as prototype which is left in
'the code for the volume of the cap of a sphere
'CapVolume = PI * 1 * 1 * (3 * 1 - 1) / 3
CapVolume = PI * CapHeight * CapHeight _
     * (3 * SphereRadius - CapHeight) / 3
'Reality check
Debug.Print CapVolume
```

—Stan Mlynek, Burlington, Ontario, Canada

VBA, VB3, VB4 16/32
Level: Beginning

A NEW DATA PROGRAMMING STYLE

While working on VB projects with associated Jet databases, I found that I was creating a lot of individual handling functions for reading, writing, and validating data. It was hard to keep track of changes made to the table structures and to their associated handling functions.

As a result, I've developed new programming style. My read / write and special functions are all in one routine selected by case logic based on descriptive strings. The strings are used only within the programs and are self documenting to some extent. Data is passed through a Public WorkVariant array:

```
Public MyDB As Database
Public WorkVariant(10) As Variant
Dim Dummy As Integer
Set MyDB = OpenDatabase("testjet3db")
'Sample calls
Dummy = NameTableIO("writerecord")
Dummy = NameTableIO("readrecord")
Public Function NameTableIO_
     (ByVal whichAction As String) As Integer
'Assumes navigation to the record has
'been made externally
Dim MyTable As Recordset
Dim ErrorStage As String
On Error GoTo NameTableIOError
     NameTableIO = True
     ErrorStage = "OpenRecordSet"
     Set MyTable = MyDB.OpenRecordset("NameTable")
     ErrorStage = "AfterOpen"
     Select Case whichAction
          Case "readrecord"
               WorkVariant(1) = MyTable![FirstName]
               WorkVariant(2) = MyTable![Age]
          Case "writerecord"
               MyTable.Edit
                    MyTable![FirstName] = WorkVariant(1)
                    MyTable![Age] = WorkVariant(2)
               MyTable.Update
Case "validateage"    'specialized routine
          'etc
End Select
NameTableIOErrorExit:

     If ErrorStage <> "OpenRecordSet" Then
          MyTable.Close
     End If
     Set MyTable = Nothing
Exit Function
NameTableIOError:

     NameTableIO = False
     MsgBox Error$ &_
          " Error trap in NameTableIO " & _
          "at " & ErrorStage & " Action: " & whichAction
     Resume NameTableIOErrorExit
End Function
```

—Stan Mlynek, Burlington, Ontario, Canada

VB3, VB4 16/32
Level: Beginning

SUPPRESS SPACES IN A TEXT BOX

To prevent users from typing spaces in a text box, include this code in the KeyPress event of the text box:

```
Private Sub Text1_KeyPress(KeyAscii As Integer)
     If KeyAscii = 32 Then
          KeyAscii = 0
     End If
End Sub
```

—Meena Swaminathan, received by e-mail

VBA, VB3, VB4 16/32
Level: Intermediate

BRANDING A JET MDB FILE
This routine allows a hidden table to be appended to the Tables collection in an MDB file. I use the table name as a brand or stamp. During development, I change the brand when design changes are made to the MDB file. Access tables with Msys and Usys preambles are treated as hidden objects by the Jet database engine. The default setting is hidden. The brand starts with UsysBRAND.

I prefer using the table name to using a record in a predefined table because it is more difficult for a casual user to delete a table than it is to delete a record from a table:

```
Public MyDB As Database
Public Const BrandString = "UsysBRAND"
Dim Dummy As Integer
Dim outputVariant
Set MyDB = OpenDatabase("testjet3db")
'Some test calls
Dummy = MDBrand("clearbrands")
Dummy = MDBrand("stamp", "TheTestProject")
If MDBrand("read", outputVariant) Then
    Debug.Print outputVariant
End If
If MDBrand("validate", "TheTestProject") Then
    Debug.Print "ProjectTest"
End If
Public Function MDBrand(ByVal whichAction As String, _
    Optional aString) As Integer
Dim MyTableDef As TableDef, _
    MyField As Field
Dim TempString As String
Dim i%
    On Error GoTo MDBrandError
    'Default
    MDBrand = False
    Select Case whichAction
        Case "clearbrands"
            For i% = MyDB.TableDefs.Count - 1 To 0 Step -1
                If Mid$(LTrim$(MyDB._
                    TableDefs(i%).Name), _
                    1, Len(BrandString)) _
                    = BrandString Then
                    MyDB.TableDefs.Delete _
                        MyDB.TableDefs(i%).Name
                    DoEvents
                    DBEngine.Idle
                    DoEvents
                    MDBrand = True
                End If
            Next i%
        Case "read"
            For i% = MyDB.TableDefs._
                Count - 1 To 0 Step -1
                If Mid$(LTrim$(MyDB._
                    TableDefs(i%).Name), _
                    1, Len(BrandString)) _
                    = BrandString Then
                    MDBrand = True
                    aString = Mid$(LTrim$(MyDB.TableDefs_
                        (i%).Name), Len(BrandString) + 1)
                End If
            Next i%
        Case "stamp"
        TempString = BrandString & aString & CStr(Now)
            If Len(TempString) > 40 Then
                TempString = Mid$(TempString, 1, 40)
            End If
            Set MyTableDef = MyDB.CreateTableDef_
                (TempString)
            Set MyField = MyTableDef._
                CreateField("MyDate", dbDate)
            MyTableDef.Fields.Append MyField
            MyDB.TableDefs.Append MyTableDef
            Set MyField = Nothing
            Set MyTableDef = Nothing
            MDBrand = True
        Case "validate"
            For i% = MyDB.TableDefs.Count - 1 To 0 Step -1
                If Mid$(LTrim$(MyDB.TableDefs(i%)._
                    Name), 1, Len(BrandString)) = _
                    BrandString Then
                    If InStr(MyDB.TableDefs(i%)._
                        Name, aString) Then
                        MDBrand = True
                End If
                End If
            Next i%
    End Select
MDBrandErrorExit:
    Exit Function
MDBrandError:
    MDBrand = False
    Resume MDBrandErrorExit
End Function
```

—Stan Mlynek, Burlington, Ontario, Canada

VB3, VB4 16/32
Level: Beginning

EXTRACT A FRAME'S VALUE
I sure wish that the VB frames containing option buttons behaved more like Access' option groups. In Access, if you have an option group containing several option buttons or toggle button controls, the option group's value property becomes the selected button's option value. Therefore, to check which button was selected, you need to check only the value of the option group itself, not the value of the individual controls.

In VB, however, the equivalent of the option group, the VB frame control, doesn't get a value. Instead, you are forced to check the value of each button to detect which one is currently selected. This process is bound to be slow and seems silly to me. For a simple workaround, create your option buttons as a control array. Attach code to the Click event of the option button array that sets the Tag property of the frame control to the Index of the control array.

For example, if your frame is named fraGender and your option button control array is named optGender, the control array's Event procedure would look like this:

```
Private Sub optGender_Click_(Index As Integer)
    fraGender.Tag = Index
End Sub
```

Then, when you need to determine which value was selected, simply query the Tag property of the frame like this:

```
    intGender = fraGender.Tag
```

—Paul Litwin, Seattle, Washington

VB4 32
Level: Intermediate

OVERLAY TRANSPARENCY FIX

The Overlay method (ImageList control) doesn't keep the transparent areas when using icons. If you execute this statement, you get ListImage #1 overlaid with ListImage #2, but the transparent areas are gone:

```
set Picture1.Picture = ImageList1.Overlay(1,2)
```

Here is my solution:

```
Set Picture1.Picture = _
    ImageList1.ListImages(1).Picture
Picture1.Scalemode = 3 'pixels
Picture1.PaintPicture ImageList1._
    ListImages(2).Picture, 0, 0
```

—Alberto Perandones, Aarhus, Denmark

VBA, VB4 16/32
Level: Intermediate

REMOVE COLLECTION ITEMS FASTER

You may have noticed that VB adds items to a collection much faster than it removes them. For example, on my machine, adding 20,000 random integers to a collection takes about four seconds, but removing them in a loop takes over 54 seconds:

```
For I = colTest.Count To 1 Step -1
    colTest.Remove I
Next
```

Because of the way VB indexes collections, it can remove items much more rapidly from the beginning of a collection than it can from the end. This code executes in only 1.5 seconds:

```
' Remove item 1, instead of item I
' (can loop in either direction)
For I = 1 To colTest.Count
    colTest.Remove 1
Next
```

And, of course, if you're clearing an entire collection, it's even faster to simply set it to Nothing:

```
' Runs in 0.35 seconds
Set colTest = Nothing
```

—Phil Weber, Tigard, Oregon

VB3
Level: Beginning

FAKING PUBLIC FORM METHODS

Frequently, I need to pass an array to a procedure outside of the form where the array is declared. Declaring the array as global is not always the best solution, but if I can make the procedure that passes the array public, I can call that procedure from outside the form. The way to do this is to add a Label control to the form, and add a handler to the Label1_Change event. You can also add a parser to parse out the function call to accept parameters:

```
Dim CommandArray_()
'This parses the function into its parameters
ParseCommand Label1.Caption, CommandArray_()
'CommandArray_(0) is the function name
'CommandArray_(1...n) are the parameters
Select Case CommandArray_(0)
    Case "Add2Array"
        Add2Array Internal_(), CommandArray_(1)
    Case "DeleteFromArray"
        DeleteArray Internal_(), _
            CommandArray_(1), CommandArray_(2)
    ...
End Select
```

When I need to add a row to my array from outside my procedure (from an MDI parent, let's say) I set it to this, and away it goes:

```
Form1.Label1.Caption = "Add2Array 1"
```

—Douglas Tarr, Berkeley, California

VB3, VB4 16/32
Level: Beginning

SMOOTHER CONTROL ANIMATION

If you ever tried making a Label scroll from the right side to the left side of the screen, you know how bad the result is. A traditional approach of doing so would be:

```
Public Sub Scrolling()
    Label1.Left = Screen.Width
    Do
    Label1.Left = Label1.Left - 15
        ' 15 is the equivalent of 1 pixel
    DoEvents
    Loop Until Label1.Left <= -(Label1.Width + 15)
End Sub
```

VB makes too many redraws in a small period of time, so you can hardly read the text of the Label. The solution is to slightly increase the duration of each appearance of the Label so the human eye has enough time to see it. To obtain the same scrolling speed, you must increase the steps of the Label from 15 to 30, or even 60. Here's how to do it:

```
Public Sub Scrolling()
    Label1.Left = Me.Width
    Do
    Label1.Left = Label1.Left - 60
        temp = Timer
        Do
            DoEvents
        Loop Until Timer - temp > 0.1
    Loop Until Label1.Left <= -(Label1.Width + 15)
End Sub
```

Doing so also makes the animation independent of the CPU's speed.

—Eric Bernatchez, Montreal, Quebec, Canada

VB3, VB4 16/32
Level: Intermediate

TRACK THE LAST FORM

I had to develop an application that worked in a similar way to an Internet browser, allowing the user to go back to the previous form. Although I could put the name of the last form into a global variable, the problem came in trying to show the form whose name was held in the variable, and the user would only be able to go back one form.

I got around this by setting the Tag property equal to the name of the form, and by defining this variable to pass the form name between displayed forms:

```
Global from_form_tag As String
```

In each of the forms to include this functionality, define these form-level variables to store the name of the form that this form was shown from:

```
Dim from_form_this As String
```

When you need to show a form from which you want to be able to return, include this code:

```
Sub btn_nextform_Click ()
from_form_tag = Me.Tag
frm_nextform.Show
End Sub
```

The code sets the Global variable to the name of the calling form. Then, in the new form you are showing, include this code:

```
Sub Form_Activate ()
If from_form_tag <> "" Then
    from_form_this = from_form_tag
End If
from_form_tag = ""
End Sub
```

This sets the form-level variable to the name of the previous form, which it retains until the form is unloaded, allowing a chain of forms to go back to. Finally, in order to return back to the previous form that was shown, include this code:

```
Sub btnback_Click ()
Dim i As Integer
Dim from_form As Form
For i = 0 To forms.Count - 1
    If forms(i).Tag = from_form_this Then
        Set from_form = forms(i)
        from_form.Show
        Exit For
    End If
Next    'i
End Sub
```

This segment of code goes through each loaded form, comparing the Tag value of that form to the value held in the form-level variable. When it finds the form, it sets a form variable to the found form and then shows it.

—Campbell Maffett, Ascot Vale, Victoria, Australia

VB3, VB4 16/32
Level: Beginning

EXTRACT DELIMITED STRINGS

I find this function quite useful:

```
Function field _
    (ByVal Source As String, _
    ByVal delim As String, _
    pos As Integer) As String

'This function accepts three parameters:
'Source - a source string with
'delimiters such as commas:
'first,second,third
'delim - the character used as a
'delimiter in the string:  ,
'pos - the ordinal position of the item
'to fetch:   3
'
'The function returns a null string
'if the delimited position specified by
'pos is empty, or the delimited position
'does not exist because the string is
'too short (reference 7th position when
'5 comma delimiters=6 fields).
'
'Examples:
'dayList$ = "Sun,Mon,Tue,Wed,Thu,,Sat"
'
'dayName$ = Field(dayList$,",",3)
'would return "Tue"
'dayName$ = Field(dayList$,",",8)
'would return ""
'dayName$ = Field(dayList$,",",0)
'would return "Sun"
'dayName$ = Field(dayList$,",",6)
'would return ""
'
'Useful for reading sub-items from INI
'strings or manipulating a delimited
'database.
'Ex: Read the INI line device from the
'WIN.INI windows section.  Use field()
'to retreive the printer port to use.
'port$ = field(iniline$,",",3)
'
'WIN.INI:
'[Windows]
'device=HP LaserJet IID,HPPCL,LPT3:
'--------------------


Source = Source & delim
If pos = 0 Then pos = 1

For s% = 1 To pos - 1
    delpos% = InStr(Source, delim)
    Source = Mid$(Source, delpos% + 1, _
        Len(Source))
Next s%

delpos% = InStr(Source, delim)
If delpos% = 0 Then delpos% = _
    Len(Source) + 1
field = Left$(Source, delpos% - 1)

End Function
```

—Joe Neubauer, received by e-mail

For even more tricks and tips go to
http://www.windx.com

VB3, VB4 16/32
Level: Intermediate

CLOSE MDI CHILD WINDOWS

In an MDI application, use this code to add a menu option that will close all MDI child windows. First, create the menu entry Close &All under the Windows menu. Then add the this code in the Click event:

```
Sub mnuCloseAll_Click ()
	Dim i As Integer
	For i = Forms.Count - 1 To 0 Step -1
		' Don't test main MDI form for
		' MDIChild property!
		If Not Forms(i) Is MDIMain Then
			If Forms(i).MDIChild Then
				Unload Forms(i)
			End If
		End If
	Next i
End Sub
```

—Abdul Semerkant, Flushing, New York

VB3, VB4 16/32
Level: Intermediate

GET ALL DIRECTORIES ON A DRIVE

This little routine retrieves the names of all directories on a drive into a list box using just a Drive ListBox and a DirListBox. The Drive ListBox is for convenience only, and the routine can be written to accept a passed drive letter. If the DirListBox is hidden, processing will be faster:

```
'Get all the directory names on a disk
'Useful when you want to search a disk for files
'Expand this code with a FileListBox to
'get all file names
Dim iLevel As Integer, iMaxSize As Integer
Dim i As Integer, j As IntegerReDim _
	iDirCount(22) As Integer
	'Maximum 22 levels of directories
ReDim sdirs(22, 1) As String
'drive1 is a drive control using which
'user selects the drive
'dir1 is the directory control
iLevel = 1
iDirCount(iLevel) = 1
iMaxSize = 1
sdirs(iLevel, iDirCount(iLevel)) = _
	Left$(drive1.Drive, 2) & "\"
Do
'iterate till no more sub-directories exist
iLevel = iLevel + 1
iDirCount(iLevel) = 0
For j = 1 To iDirCount(iLevel - 1)
	dir1.Path = sdirs(iLevel - 1, j)
	dir1.Refresh
	If iMaxSize < (iDirCount(iLevel) + dir1.ListCount) Then
		ReDim Preserve sdirs(22, _
			iMaxSize + dir1.ListCount + 1) As String
		iMaxSize = dir1.ListCount + iDirCount(iLevel) + 1
	End If
	For i = 0 To dir1.ListCount - 1
		iDirCount(iLevel) = _
			iDirCount(iLevel) + 1 'count
			'of sub-dirs
			sdirs(iLevel, iDirCount_
				(iLevel)) = dir1.List(i)
	Next i
Next j
'Load all the directory names in a list box List1
list1.Clear
If iDirCount(iLevel) = 0 Then
'When no more sub-directories exist
	For i = 1 To iLevel
		For j = 1 To iDirCount(i)
			list1.AddItem sdirs(i, j)
		Next j
	Next i
	Exit Do
End If
Loop
```

—Kumar Prabodh, received by e-mail

VB4 32
Level: Beginning

REPLACE TEXT IN A RICH TEXT BOX

The RichTextBox in VB4/32 is a real gem. Among other great features, it allows files of any size, breaking the annoying 64K limitation of 16-bit operating systems. The new control also has a useful method: Find. As its name implies, it is used to find strings in text files directly. There is no need to use the tedious workaround of coding with the InStr function. Unfortunately, the Find method is missing its natural pair, Replace. Therefore, you must write Replace code to complement the Find method.

The main difficulty in writing Replace procedures is the length of both the FindString and ReplaceWithString. Obviously, there are three cases: =, >, and <. If the two strings are of equal length, no special care is needed. But in the other two cases, which occur frequently, you must do some extra work with the FindString (the OldString in my code). You need to add to it extra spaces if it is shorter than the ReplaceWithString. This way, you won't cut off characters from the next word in the file.

If the FindString is longer than the ReplaceWithString, you must eliminate the trailing characters. This way, the ReplaceWithString does not have trailing spaces:

```
If Len(NewString) > Len(FindString) Then
	' NewDummyString is a temporary
	' string with extra spaces ("     ")
	NewDummyString = DummyString + _
		String(Len(NewString) - Len(findstring), " ")
	Mid(NewDummyString, 1, Len(NewString)) = NewString
	TempString = NewDummyString
ElseIf Len(NewString) < Len(FindString) Then
	NewDummyString = DummyString + _
		String(Len(FindString) - Len(NewString), " ")
	Mid(NewDummyString, 1, Len(FindString)) = NewString
	TempString = Left(NewDummyString, Len(NewString))
Else
	' the strings are of equal length
	TempString = NewString
End If

rtfBox.SelStart = Pos
rtfBox.SelLength = Len(FindString)
rtfBox.SelText = TempString
```

—Ion Saliu, Cashtown, Pennsylvania

VB3, VB4 16/32
Level: Beginning

An InBetween Function

This function returns True if the first number in the argument is in between the next two arguments. Use this function to detect collisions, and for many other useful numeric functions:

```
Function InBetween_
     (ByVal Num As Variant, ByVal X As Variant, _
     ByVal Y As Variant) As Boolean
     InBetween = False
     If IsNumeric(Num) And IsNumeric(X) _
          And IsNumeric(Y) Then
          If X < Y Then
               If Num > X And Num < Y Then
                    InBetween = True
               End If
          ElseIf Y < X Then
               If Num > Y And Num < X Then
                    InBetween = True
               End If
          End If
     End If
End Function
```

For example:

```
X = IsBetween(1, 5, 3)     'X returns True
X = IsBetween(2, 12, 6)    'X returns False
```

—Tony Keck, Brookville, Indiana

VB3, VB4 16/32
Level: Intermediate

Stack 'Em Up

A status panel on an MDI form is an ideal way to keep your user informed about what the system is currently doing. Often, you may have several nested processes that may write to that status panel as each one executes.

Maintaining an accurate status through these nested processes can be messy, if not impossible. This code allows you to maintain a "stack" of statuses to allow for this type of nesting:

```
Global StatusPanelStack(1 To 10) As String
Global StatusPanelSize As Integer
Sub PushStatusPanel ()
     StatusPanelSize = StatusPanelSize + 1
     If StatusPanelSize > UBound(StatusPanelStack) Then
          Redim Preserve StatusPanelStack_
               (1 To UBound(StatusPanelStack) + 10) As String
     End If
     StatusPanelStack(StatusPanelSize) _
          = MDIMain.pnlMain.Caption
End Sub
Sub UpdateStatusPanel (StatusText As String)
     PushStatusPanel
     MDIMain.pnlMain.Caption = Trim$(StatusText)
End Sub
Sub PopStatusPanel ()
     StatusPanelSize = StatusPanelSize - 1
     If StatusPanelSize = 0 Then
          MDIMain.pnlMain.Caption = "Ready"
     Else
          MDIMain.pnlMain.Caption = _
                    StatusPanelStack(StatusPanelSize)
End If
End Sub
```

—Jeff Trader, Liberty, Missouri

VB3, VB4 16/32
Level: Intermediate

"Determining If an Object Has Been Set," Revisited

I have a way to determine if an object has been set that is simpler and faster than the method offered in the tip, "Determining If an Object Has Been Set" ["101 Hot Tech Tips for VB Developers," Supplement to the August 1996 issue of *VBPJ*, page 23]:

```
myObj Is Nothing
'where myObj is the object to test
'whether it has been set or not

'example:
if myObj is Nothing then
        MsgBox ("myObj is not set")
else
        MsgBox ("myObj is set")
end if
```

—Ari Singh, Columbus, Ohio

VB3, VB4 16/32
Level: Advanced

Set up ODBC Data Source During Installation

Use this code to set up an ODBC data source during the installation procedure. If you create the setup program yourself, you can put this code in the setup1 A.frm. If you have ODBC installed on your computer, it provides a silent ODBC data-source setup:

```
'Declare in your VB module
Declare Function SQLConfigDataSource Lib "ODBCINST.DLL" _
     (ByVal hWnd As Integer, ByVal)
fRequest As Integer, ByVal lpszDriver As String, _
     ByVal lpsAttrib$) As Integer
'and then specify you data source info:
Dim iRet as Integer
Dim lpszAttributes As String
lpszAttributes ="DSN=Data source name" & Chr$(0)
lpszAttributes =lpszAttributes & _
     "Description=New  system on SQL6" & Chr$(0)
lpszAttributes =lpszAttributes & _
     "Server=yourserver" & Chr$(0)
'91for TCP/IP
lpszAttributes =lpszAttributes & _
     "Address=162.66.125.65,7024" & Chr$(0))
lpszAttributes =lpszAttributes & _
     "UseProcForPrepare=Yes" & Chr$(0)
lpszAttributes =lpszAttributes & _
     "Database=your database name" & Chr$(0)
lpszAttributes =lpszAttributes & _
     "Language=us_english" & Chr$(0)
lpszAttributes =lpszAttributes & "OEMTOANSI=No" & Chr$(0)
'network library
lpszAttributes =lpszAttributes & _
     "Network=wdbnovtc" & Chr$(0) & Chr$(0)
iRet =SQLConfigDataSource(0, 1, "SQL _
```

For even more tricks and tips go to
http://www.windx.com

```
    Server", lpszAttributes)
'if you already have data source with
'the same name, display confirmation
'message; otherwise, setup ODBC
If iRet <> 1 Then
    iRet =SQLConfigDataSource(setup1.hWnd, 1, _
    "SQL Server", lpszAttributes)
    If iRet <> 1 Then MsgBox _
        "Error during setup of the " & _
        "Data Source. Install data " & _
        "source manually"
End If
```

Remember that some of the attributes depend on the way your SQL Server is set up. For example, if you exclusively use "Named Pipes," the address parameter will not be included.

—Eugene Dvorkin
Weehawken, New Jersey

VB3, VB4 16/32
Level: Beginning

FILTER USER KEYBOARD INPUT

This code removes the need for the Masked Edit control in most situations. It works with VB3 and VB4, although you need only the MouseDown and MouseUp events in Win95 to deal with built-in context menus. However, the user will not be able to paste legitimate numbers.

Add this code as appropriate to your text box and users will be able to put in only numbers. You can modify it to work with any specific set of characters:

```
Option Explicit
Dim tText As String
Private Sub Text1_KeyPress(KeyAscii As Integer)
    Dim tKeyAscii As Integer
    'Define some stuff
    tKeyAscii = 0
    If KeyAscii >= Asc(0) And KeyAscii <= Asc(9) Then _
        tKeyAscii = KeyAscii
        'Make sure it's a digit

    If KeyAscii = 8 Then tKeyAscii = 8
        'or the backspace key
        KeyAscii = tKeyAscii
        'Give them our filtered key.
End Sub
Private Sub Text1_MouseDown_
    (Button As Integer, Shift As _
    Integer, X As = Single, Y As Single)
    ' Relies on form-level variable to
    ' prevent changes via clipboard
    tText = Text1.Text
End Sub
Private Sub Text1_MouseUp_
    (Button As Integer, Shift As _
    Integer, X As = Single, Y As Single)
    'Ignore their changes!
    Text1.Text = tText
End Sub
```

—Holland Rhodes, Martinez, California

VB4 16/32
Level: Beginning

ITERATING CONTROLS COLLECTION TO SET PROPERTIES

There are many cases when a VB form has quite a few text-box controls—any data entry form, for example—and you want to clear the text of these text boxes frequently. For instance, every time you enter new data, you have to clear all text boxes to allow the user to enter new data. Thanks to the VB4 Controls Collection, you can do this in only one shot and save yourself a lot of time trying to figure out the names of those text boxes and setting values to an empty string for each of the text boxes. This subroutine does it all in only two lines of code. You only need to pass the Form name as the parameter to this subroutine (that is, Call gClearForm(Me)):

```
Public Sub gClearForm(FName As Form)
    Dim MyControl As Control
    For Each oMyControl In FName.Controls
        If TypeOf MyControl Is TextBox Then
            MyControl.Text = ""
        End If
    Next MyControl
End Sub
```

You can also use this tip to do many other things, such as making all command buttons disabled, making List Indexes of all combo boxes -1, and so on. You only need to modify the If condition and property of that control.

—Adesh Jain, Newington, Virginia

VBA, VB3, VB4 16/32
Level: Intermediate

SWAP VALUES USING ONLY TWO VARIABLES

This code swaps values using two variables only:

```
Sub Form_Load ()
    SwapThis Rnd(4) * 100, Rnd (5) * 100
End Sub
```

This routine swaps the values of A and B. There is no need to use a third variable:

```
Private Sub SwapThis (A As Integer, B As Integer)
    'before the swap
    MsgBox "A = " & Str(A) & " B = " & Str(B)
    A = A + B
    B = A - B
    A = A - B
    'After the swap
    MsgBox "A = " & Str(A) & " B = " Str(B)
End Sub
```

Note that this code crashes with an overflow error if (A+B) > 32K.

—Edwin M. de Guzman, Houston, Texas

VB3, VB4 16/32
Level: Beginning

REPLICATE MENUS

Press the Menu Design icon in VB. Make the menu follow these steps:

```
caption:     Menu 1
name:        mnuMain1
index:       0
```

Press the icon with the right arrow to create the menu popup. Add this code:

```
caption:     Item 1
name:        mnuItem1
idex:        1
caption:     Item 2
name:        mnuItem1
index:       2
caption:     Item 3
name:        mnuItem1
idex:        3
```

Repeat the second step, changing the Menu title from “Menu 1” to “Menu 2.” When you see the menu in your form, save it. Call Notepad or Write to see how VB saves the form. Copy these lines:

```
Begin Menu mnuMain1
     Caption  =   "&Menu 1"
     Index    =   0
     Begin Menu mnuItem1
          Caption  =   "Item 1"
          Index    =   1
     End
     Begin Menu mnuItem1
          Caption  =   "Item 2"
          Index    =   2
     End
     Begin Menu mnuItem1
          Caption  =   "Item 3"
          Index    =   3
     End
End
Begin Menu mnuMain2
     Caption  =   "M&enu 2"
     Index    =   0
     Begin Menu mnuItem2
          Caption  =   "Item 1"
          Index    =   1
     End
     Begin Menu mnuItem2
          Caption  =   "Item 2"
          Index    =   2
     End
     Begin Menu mnuItem2
          Caption  =   "Item 3"
          Index    =   3
     End
End
```

Open the other file and paste it. Save the new file with another name, such as “mnuHead.”

Each time you need to make a menu, you simply go to “mnuHead,” copy these lines, open the form, and paste it. Remember to save the form in text mode. If I need five, I only copy and paste five. Remember to insert the “End” of “Begin Menu mnuMain1.” Rename each of the items before you save the form.

These easy steps will save you a lot of work.

—Gonzalo Medina Galup, Miami, Florida

VB4 16/32
Level: Intermediate

COLOR STATUS INDICATOR

Have you ever wanted to create a status bar that gave color indication in addition to the percentage? Let’s say you desire this effect:

>=0 and <50	Green
>=50 and <75	Yellow
>=75 and <=100	Red

Place a Sheridan 3D Panel on your form, change the name to pn3Status, clear the Caption property, and set the FloodType property to 1 (Left to Right). Then place this code on your form:

```
Private Sub SetStatus(value%)
     If value < 0 Or value > 100 Then Exit Sub
     pn3Status.FloodPercent = value
     pn3Status.FloodColor = _
          RGB(-255 * (value >= 50), _
                -255 * (value < 75), 0)
End Sub
```

This not only updates the percentage, but it changes the FloodColor accordingly. You can also put this into a Property procedure and make it public so that the status can be set from elsewhere in the project.

To test this function, place this in a command button Click event:

```
For i = 1 To 100
     SetStatus (i)
Next
```

—Peter W. DeBetta, Morrisville, North Carolina

VB4 16/32
Level: Intermediate

CALL PROPERTY PROCEDURES WITH MULTIPLE PARAMETERS

Property procedures, which are new in VB 4.0, have a distinct calling convention. You can send any number of arguments to property procedures with only one restriction: the Property procedure Get should have one argument less than the Property procedure Let.

Property procedures don’t support Named arguments, and they cannot be called as normal procedures with multiple parameters. The only way to call these procedures is to have n-1 parameters on the left side and the nth argument on the right side of the assignment:

```
Property Let Test (arg1 As String, _
     arg2 As String, arg3 As Integer)
End Property
'calling the above property Test:
Test(arg1,arg2) = arg3
```

The last argument should be on the right side of the assignment.

—K. Ramesh, Kottur, Madras, India

For even more tricks and tips go to
http://www.windx.com

VBS
Level: Intermediate

Dynamic Web Pages with VBScript

Although most Visual Basic developers are accustomed to placing code only inside a sub or function, VBScript actually lets you write code that can be executed without a function call. Code written outside of a function is executed immediately when a Web page is displayed.

This technique, known as "immediate execution," is extremely useful for creating simple dynamic Web pages. This VBScript code shows a new "Tip of the Month" automatically when the Web page is loaded. Simply place the code directly in a Web page where you want the tips to appear:

```
<SCRIPT LANGUAGE="VBS">
<!—
    Dim strTip
    Select Case Month(Now)
        Case 1
            strTip="Tip 1"
        Case 2
            strTip="Tip 2"
        Case 3
            strTip="Tip 3"
        Case 4
            strTip="Tip 4"
        Case 5
            strTip="Tip 5"
        Case 6
            strTip="Tip 6"
        Case 7
            strTip="Tip 7"
        Case 8
            strTip="Tip 8"
        Case 9
            strTip="Tip 9"
        Case 10
            strTip="Tip 10"
        Case 11
            strTip="Tip 11"
        Case 12
            strTip="Tip 12"
    End Select
    Document.Write "<CENTER><H3>" & _
        strTip & "</H3></CENTER>
—>
</SCRIPT>
```

**—Scot P. Hillier, New Technology Solutions Inc.
http://www.vb-bootcamp.com**

VB3, VB4 16
Level: Intermediate

Monitor DLL Usage

Many times, you want to monitor the usage of a DLL. You need to know how many instances of the DLL are loaded and used. This is especially useful during DLL development. During debugging, you must always make sure that no previous instance is loaded and that you are debugging the correct instance:

```
Declare Function GetModuleUsage Lib "Kernel" _
    (ByVal hModule As Integer) As Integer
Declare Function GetModuleHandle Lib "Kernel" _
    (ByVal lpModuleName As String) As Integer
Sub Form_Load ()
Dim Handle As Integer
Dim Counter As Integer
Handle = GetModuleHandle("CommDlg.dll") ' Get the handle of
    'the DLL you want to monitor
Counter = GetModuleUsage(Handle)
' Get the number of usage.
Debug.Print Str(Counter)
' Print on the debug window
End Sub
```

—Ramy Habashy, Agouza, Giza, Egypt

VBA, VB3, VB4 16/32
Level: Beginning

Cut-and-Paste Trick for Power Programmers

As a power programmer, I cut and paste incessantly. I end up with my right hand on the mouse clicking and highlighting, and my left hand on the keyboard playing the Ctrl, X, C, and V keys. My eyes are glued to the screen, and I can't afford to look at my left hand.

Invariably, I make mistakes. The most common mistake I make is using Ctrl-X (delete) instead of Ctrl-C (copy). Luckily, Windows has the Undo function. But these mistakes still break my concentration and throw me off.

To reduce these finger errors, I now "home" my left forefinger to the C key. I've placed a piece of tape on it to give it a different feel. I use the forefinger for C and V and the middle finger for Z and X. I'm considering hot-gluing a metal nut to the C key!

—Stan Mlynek, Burlington, Ontario, Canada

VBS
Level: Intermediate

Using Collections in VBScript

Just like Visual Basic, VBScript provides collections that allow easy access to the controls in a form. In VBScript, the controls collection is called the Elements Array. However, it behaves in a manner similar to the controls collection in VB. This code snippet shows how you can use a VBScript routine to easily change controls in the collection. In this case, you are simply clearing all the text boxes in a form:

```
<SCRIPT LANGUAGE="VBS">
<!—
    Sub Clear
        For i=0 To Document.Forms(1).Elements.Count-1
            Document.Forms(1).Elements(i).Value=""
        Next
    End Sub
—>
</SCRIPT>
```

**—Scot P. Hillier, New Technology Solutions Inc.
http://www.vb-bootcamp.com**

VBS
Level: Intermediate

SIMULATING CONTROL ARRAYS IN VBSCRIPT

VBScript does not inherently support the concept of control arrays. This decent workaround, however, cuts a lot of excess coding. Use the Event attribute for any intrinsic HTML control to call a common routine. For example, this code passes an index to a common routine for an array of five intrinsic HTML button controls:

```
<SCRIPT LANGAUGE="VBS">
<!—
    Sub cmdArray_Click(intIndex)
        MsgBox "You pushed the " & intIndex " button!"
    End Sub
—>
</SCRIPT>

<BODY>
<FORM>
<INPUT TYPE="BUTTON" ONCLICK="cmdArray_Click(0)">
<INPUT TYPE="BUTTON" ONCLICK="cmdArray_Click(1)">
<INPUT TYPE="BUTTON" ONCLICK="cmdArray_Click(2)">
<INPUT TYPE="BUTTON" ONCLICK="cmdArray_Click(3)">
<INPUT TYPE="BUTTON" ONCLICK="cmdArray_Click(4)">
</FORM>
</BODY>
```

—Scot P. Hillier, New Technology Solutions Inc.
http://www.vb-bootcamp.com

VB3, VB4 16/32
Level: Beginning

USE DOEVENTS TO SIMULATE MULTITHREADING

It's possible for VB apps to perform multiple tasks simultaneously if you put each task inside a DoEvents loop. Consider this code:

```
Do While bProcessing
    Me.Print Int(Rnd * 10)
    DoEvents
Loop
```

As long as the form-level variable bProcessing is True, this loop generates and prints random numbers. The DoEvents at the bottom of the loop yields control to other processes, and to other code within the same app, allowing it to perform other activities simultaneously.

—Ion Saliu, Cashtown, Pennsylvania

VB3, VB4 16/32
Level: Beginning

USE A LIST BOX TO NAVIGATE A RECORD SET

Here's how to use a list box to find the first matching record in a record set. First, fill the list box with the unique values in a specific field:

```
Dim rs As Recordset
Dim SQL As String
SQL = "SELECT DISTINCT Field " & FROM Table;"
Set rs = db.OpenRecordset(SQL, dbOpenSnapshot)
If rs.RecordCount Then
    Do Until rs.EOF
        lstBox.AddItem rs!Field
        rs.MoveNext
    Loop
End If
```

Then add this code to the list box's Click event:

```
Private Sub lstBox_Click()
    Dim sFind As String
    Dim sCriteria As String
    ' Get selected item
    sFind = Trim$(lstBox.Text)
    If Len(sFind) Then
        ' Show first matching record in
        ' bound controls on form
        sCriteria = "[Field]='" & sFind & "'"
        datCtl.Recordset.FindFirst sCriteria
    End If
End Sub
```

—Claudia Vega Cachú, Tepic, Nayarit, Mexico

VB3, VB4 16
Level: Intermediate

API SELECTS ITEMS IN COMBO BOX

One day I found myself having a lot of trouble with a combo box. I was getting error messages when I tried to save the ID in the list index from that particular combo to the record set. I would get errors like "Invalid Use of Array" in the line "rs!id = cbo1.ItemData(cbo1.ListIndex)." Sometimes, I would not even get the value of the list index after using the "If cbo1.ListIndex = 1 Then" before my save event would work.

Now, I use the Windows API SendMessage. In the General Declarations section of the form, declare:

```
Private Declare Function SendMessage _
    Lib "User" (ByVal hWnd As Integer, _
    ByVal wMsg as Integer, _
    ByVal wParam As Integer, _
    Iparam As Any) As Long
Private Const CB_SelectString = &H14D
```

Within the calling routine, declare:

```
Dim X As Integer
Dim returnvalue As Integer
```

Place these two lines in your save procedure:

```
returnvalue = SendMessage(cbo1.hWnd, _
    CB_SelectString, X, cbo1.Text)
rs!id = cbo1.ItemData(cbo1.ListIndex)
```

—Marcia Silva, San Diego, California