Supplement to

www.windx.com
A FAWCETTE TECHNICAL PUBLICATION
AUGUST 1997 VOL. 7, NO. 9

VISUAL BASIC® PROGRAMMER'S JOURNAL

INCLUDES WINDOWS PROGRAMMING & VISUAL PROGRAMMING

Fifth Edition

Featuring

For VB Developers

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

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

These tips and tricks were submitted by professional developers using Visual Basic 3.0, Visual Basic 4.0, Visual Basic 5.0, Visual Basic for Applications, and Visual Basic Script. The tips were compiled by the editors at *Visual Basic Programmer's Journal*. 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, and indicate if it's for VBA, VBS, VB3, VB4 16- or 32-bit, or VB5. 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: Intermediate

COMBO BOX EVENT HANDLING

Two problems can arise when a confused user scrolls up or down with the mouse and then makes a selection with the Enter key. First, the down arrow fires two events: Change and Click. Second, the Enter key moves focus to the next tab stop, while the mouse click doesn't remove focus from the combo box. Therefore, if you place your action code in the Change event, an up or down arrow will fire it, which you don't want. On the contrary, if you place your action code only in the LostFocus event and the user clicks on a selection, focus won't move from the combo box, and the user is left staring at the selected text in the combo box, wondering why no action occurred.

This solution filters out Click events generated with arrow keys and forces the control to lose focus. In the Declarations section of the form, enter this code:

```
' Note: Use an Integer flag variable in
' VB3
Dim bNoise as Boolean
' True denotes a Noise Event which is to
' be ignored
```

Enter this code in the Form_Load event:

```
bNoise = False
```

Enter this code in the combo box KeyDown event:

```
Private Sub cbTest_KeyDown(KeyCode As _
     Integer, Shift As Integer)
     ' If the user is browsing with the
     ' arrows, ignore the Click Events
     If KeyCode = vbKeyDown Or KeyCode _
          = vbKeyUp Then bNoise = True
End Sub
```

Enter this code in the combo box Click event:

```
Private Sub cbTest_Click()
     If bNoise Then
          ' Ignore Noise events
          ' (up or down arrow)
          bNoise = False
     Else
          ' Force loss of focus
          SendKeys "{TAB}", True
     End If
End Sub
```

Write code that reacts to a new user selection in the combo box LostFocus event. Don't send a Tab keystroke because focus has already shifted, and the combo box's behavior is consistent regardless of how the user selects a new value.

—Les Smith, Concord, North Carolina

VB5
Level: Beginning

COMMENT AND UNCOMMENT BLOCKS OF CODE

Visual Basic 5.0 lets you comment a block of code in a snap and uncomment it later. This feature is useful in the debug phase, when you don't want to execute a number of statements, but don't want to physically delete them either. However, the Comment/Uncomment command pair isn't present in any menu of the environment, and you can only reach it by enabling the Edit toolbar. To do this quickly, right-click on any toolbar in the environment and select the Edit command.

—Francesco Balena, Bari, Italy

VB5
Level: Beginning

DON'T CREATE ALIASED VARIABLES

Never pass a global variable as an argument to a procedure that also accesses the variable directly. If you're 100 percent sure you adhered to this rule within your application, check the Assume No Aliasing option in the Advanced Optimizations dialog that you invoke from the Compile tab of the Project Properties dialog. If the native code compiler knows no aliased variables exist, it can freely cache variable values into faster CPU registers, and store them back to RAM memory only when leaving the procedure. This improves the execution speed of compiled programs.

—Francesco Balena, Bari, Italy

VB5
Level: Beginning

DEFAULT VALUES FOR OPTIONAL PARAMETERS

If you ever programmed under VB4, you probably took advantage of the powerful Optional parameters. VB5 enhanced them in several ways: they can be of any type now (not only Variants), and can appear in Property procedures. Interestingly, you can now state a default value for them:

```
Property Get Value (Optional index As Long = 1)
...
End Property
```

You can do this without an explicit (and slower) IsMissing test:

```
Property Get Value (Optional index As Long)
If IsMissing(index) Then index = 1
...
End Property
```

—Francesco Balena, Bari, Italy

VB5
Level: Beginning

CENTER FORMS ON SCREEN

A popular code snippet lets you center any form on the screen, regardless of the current screen resolution. You now can reach the same result by simply assigning the value vbStartUpScreen (=2) to the form's StartUpPosition new property. You can even center a form within its parent window by assigning the vbStartUpOwner (=1) value. You can set this property from the Property window.

When a form is supposed to be centered within its parent window, remember to add a second argument to the Show method:

```
Form2.Show vbModal, Me
```

—Francesco Balena, Bari, Italy

VB5
Level: Beginning

DON'T AUTO-OPTIMIZE FOR FAST CODE

If you take a look at VB's native code optimization options for the first time, you might be tempted to click on "Optimize for Fast Code" right away. Strange as it may sound, this does not always guarantee the best performance. Applications optimized for performance generally don't run much faster, but do have a larger memory footprint. This causes them to load slower, especially on memory-constrained machines, giving the user the impression that your app is actually slower than one optimized for compact code.

For the same reason, consider leaving your applications compiled as p-code. Especially for large, UI- and database-intensive applications, the performance gain of compiling to native code won't outweigh the increase in application size. To determine exactly which compilation option is right for you, use the VB Application Performance Explorer (APE) included on your VB CD.

—Michiel de Bruijn, Rotterdam, The Netherlands

VBA5
Level: Beginning

NOT ALL TEMPLATES ARE CREATED EQUAL

Unlike templates in other Office 97 products, Word 97 templates provide a business-application engine that remains separate from the documents that use that engine. Template-based Excel workbooks and PowerPoint presentations include a copy of that engine. In practice, all Word documents consist of two VBA projects: the first project is from the underlying template (all Word documents are based on a template), and the second project belongs to the Word document itself. On the other hand, Excel workbooks and PowerPoint presentations based on a template have only one VBA project. Every file contains its own copy of the project in the original template. Changes made to this copy don't affect the underlying template.

In Excel, PowerPoint, and Access, add-ins provide code engines that remain separate from the workbooks and presentations using those engines. To create an Excel or PowerPoint add-in, use the Save As command on the File menu and save the file as a specific type: the "Add-in" type. Each product uses a specific "Add-in" extension (XLA for Excel, PPA for PowerPoint, and MDA for Access).

There's no canonical location for storing add-ins, but to load add-ins automatically when you launch an Office product, store them in the XLStart folder or in the StartUp folder. You can load add-ins manually with the Add-ins command on the Tools menu, or you can automate the process in code.

To create an Access add-in, use the command "Make MDE file" available through the Database Utilities command on the Tools menu.

—Christine Solomon, New York, New York

VB5
Level: Beginning

CUSTOMIZE VB TOOLBARS

Here are a few simple ways you can customize your VB5 IDE:

• Add tabs to the custom control toolbox by right-clicking on the General button and selecting the Add Tab command. You can also move tabs around and delete them, as well as move control icons from one tab to the other by using the drag-and-drop method.
• Create toolbar buttons for any menu command by right-clicking on any toolbar and selecting the Customize command. Move to the Commands tab, select the menu command in the rightmost list box, and drag it onto the toolbar where you want to move it. Good candidates for this procedure are the Project-References, Project-Properties, and Tools-Add Procedure commands.
• Create a brand new toolbar in the Toolbars tab of the Customize dialog box. After you define a toolbar, add buttons using the procedure outlined above. When the Customize dialog box is active, right-click on any toolbar button to change its image, create a group divider, show/hide text, and more.

—Francesco Balena, Bari, Italy

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

VB5
Level: Beginning

HIDE ALL PROJECT WINDOWS

When working with multiple projects, it's easy to get confused by the many windows active on the screen at the same time. However, you can temporarily hide all the windows related to a given project by simply collapsing the project item in the Project Explorer window. You can disable this feature in the General tab of the Tools-Options dialog box.

—Francesco Balena, Bari, Italy

VB5
Level: Intermediate

FRIENDLY ENUMERATED VALUES

If you build an ActiveX control that exposes an enumerated property, you should define a Public Enum structure that gathers all the possible values for that property. Doing this helps the developer using your control because the enumerated values will be listed in a combo box in the Property window.

However, at first glance, it seems impossible to achieve the same behavior as most of VB's intrinsic controls, which expose enumerated properties with short descriptions and embedded spaces. Even if they're not documented in the language manuals, you can create enumerated items that embed spaces by enclosing their names within square brackets:

```
Public Enum DrawModeConstants
     Blackness = 1
     [Not Merge Pen]
     [Mask Not Pen]
     [Not Copy Pen]
     ...
End Enum
```

Then add a DrawModeConstants property to the ActiveX control. All the enumerated values appear in the Property window of the VB IDE, without the square brackets and with all the spaces you included. Use this technique to embed other forbidden characters, such as math or punctuation symbols.

—Francesco Balena, Bari, Italy

VB5
Level: Advanced

PROPERTIES THAT BEHAVE LIKE TEXT AND CAPTION

If you build an ActiveX control that exposes a Text- or Caption-like property, even under different names, you should modify its attributes in the Procedure Attribute dialog box after expanding it by using the Advanced button. This way, the Procedure ID is set to Text or Caption, respectively.

This causes your property to behave like standard Text or Caption properties. When the user modifies its value in the Property window, the effect of each new character is immediately reflected on the ActiveX control itself.

—Francesco Balena, Bari, Italy

VB4 16/32, VB5 (Enterprise Edition)
Level: Intermediate

STANDALONE TYPE LIBRARIES

If you create out-of-process OLE servers, Visual Basic embeds the companion type library into the EXE file and generates no TLB file. However, if you own the Enterprise Edition of VB4 or VB5, you can flag the Remote Server File check box to have Visual Basic create a standalone type library. In VB5, you can find this option in the Component tab of the Project-Properties dialog box.

—Francesco Balena, Bari, Italy

VB4 16/32, VB5
Level: Advanced

IMPLEMENTATION OF PUBLIC FORM AND CLASS VARIABLES

The implementation of Public variables in forms and classes changed with Visual Basic 5.0. VB4 implements public variables in forms and class modules as if they're regular variables, using pointers to data in memory. In VB5, public variables are more correctly implemented as a pair of hidden Get/Let property procedures. This approach slows down these properties when the program is ported from VB4 to VB5.

More important, if you have a VB4 program that passes such Public variables to a procedure using the ByRef keyword (or no keyword at all, which results in the variable being passed by reference) and that relies on the procedure to modify the value of this argument, this code won't work correctly when recompiled under VB5. In fact, under VB5, they're passed by value, and the original property is never affected. For more information on this issue, see article Q166928 in the Microsoft Knowledge Base.

—Francesco Balena, Bari, Italy

VB5
Level: Intermediate

USE OBJECT BROWSER TO DISCOVER UNDOCUMENTED FEATURES

If you right-click on the right-most pane of the Object Browser, you can issue the Show Hidden Members command. From this point on, the Object Browser shows all hidden properties and methods in any library, and you can use it to explore all object libraries in more detail.

For instance, the VBA library exposes a hidden class, appropriately named "_HiddenModule," which includes many well-known VBA functions plus three undocumented ones: ObjPtr, StrPtr, and VarPtr. ObjPtr returns the address of the private area of an object instance, StrPtr returns the address of the first character in a string, and VarPtr returns the address of a variable or a string descriptor, if you pass it a string variable.

—Francesco Balena, Bari, Italy

VB4 16/32
Level: Advanced

The Address of a Variable

VB5 includes a built-in VarPtr function (see tip "Use Object Browser to Discover Undocumented Features" on page 3), but this function isn't available in VB4. The VB4 runtime library does include this function, but you must declare it first:

```
#If Win16 Then
Declare Function VarPtr Lib _
    "VB40016.DLL" (variable As Any) As Long
#Else
Declare Function VarPtr Lib "VB40032.DLL" (variable As Any) _
    As Long
#End If
```

This function is useful when passing an external API routine a Type structure, and one of its fields is the address of another variable or record.

—Francesco Balena, Bari, Italy

VB4 16/32, VB5
Level: Intermediate

Cross Midnight Benchmarks

Traditionally, VB programmers benchmark their code using the Timer function. However, if your process might terminate on the following day, you must take into account that the value returned by that function is reset at midnight. If you're satisfied with one-second precision, you can simplify your code using the Now function:

```
Dim startTime As Date
StartTime = Now
' the code to be benchmarked
' ...
Print "elapsedSeconds = " & Format$ _
    ((Now - startTime) * 86400, "#####")
```

You need the Format$ function to round the result to the nearest Integer.

—Francesco Balena, Bari, Italy

VB5
Level: Intermediate

App.Path Might Return UNC Path Specifications

Unlike VB4, VB5's App.Path property might return a UNC path, such as "\\server\programs\...", depending on how the program started and if it's interpreted in the VB IDE or compiled as a standalone EXE. This change likely affects all applications that use App.Path to set the current directory when the program starts:

```
ChDrive App.Path
ChDir App.Path
```

In fact, because ChDrive cannot handle UNC paths, the code might raise a fatal runtime error and should be protected using an On Error Resume Next statement. This fix, however, doesn't protect you under every possible condition. The best approach is to give the end user the capability to set the application directory at run time, then save the entered value in the registry or in an INI file. For more information on this problem and its possible solutions, see article Q167167 in the Microsoft Knowledge Base.

—Francesco Balena, Bari, Italy

VB4 16/32, VB5
Level: Advanced

More Versatile Array Parameters

You can write a single procedure that accepts any type of array as an argument by using a variant parameter. Within the procedure, address the array using the usual syntax:

```
' return the number of items
Function ItemCount(anArray As Variant) As Long
ItemCount = UBound(anArray) - LBound(anArray) + 1
' the first element is anArray(LBound(anArray))
End Function
```

You can even pass a matrix with any number of dimensions; in order to understand how many dimensions, you must iterate on the UBound or LBound functions until an error occurs:

```
Function ItemCount(anArray As Variant) As Long
Dim items As Long, i As Integer
On Error Resume Next
items = UBound(anArray) - LBound(anArray) + 1
For i = 2 to 999
    items = items * (UBound(anArray, _
        i) - LBound(anArray, i) + 1)
    If Err Then Exit For
Next
ItemCount = items
End Function
```

—Francesco Balena, Bari, Italy

VB4 16/32, VB5
Level: Intermediate

Compact Your Code Using IIf and Switch

You can often replace an If...Then...Else block with a more compact IIf function:

```
' returns the max of two values
maxValue = IIf(first >= second, first, second)
```

Switch is a rarely used function, yet it can prove rather useful as a substitute for a lengthy If...ElseIf block:

```
' is "x" negative, positive or null?
Print Switch(x < 0, "negative", x > 0, _
    "positive", True, "Null")
```

Note the last test is True, because the three conditions are mutually exclusive and exhaustive.

—Francesco Balena, Bari, Italy

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

VB5
Level: Advanced

COMBINE DEFAULT WITH OTHER ATTRIBUTES

When building an ActiveX control, you can set a default property or method using the Procedure Attributes dialog box, after clicking on the Advanced button. However, if your default property also happens to require another special attribute—as is the case with Caption and Text properties—you're in trouble because the Procedure ID combo box only permits one selection.

Suppose your ActiveX control exposes a Caption property you want to behave as a regular property. For example, all keys typed in the Property window are immediately reflected in the control itself. In order to achieve this behavior, assign the Caption attribute to this property in the Procedure ID combo box (see tip "Properties That Behave Like Text and Caption" on page 3). If you also want to make it the default property, you must resort to a trick: declare another, hidden property that delegates to your Caption property, and set this new property as the default member of the ActiveX control. The name of this property is not important because the user never sees it:

```
Property Get DefaultProp() As String
     DefaultProp = Caption
End Property

Property Let DefaultProp(newValue As String)
     Caption = newValue
End Property
```

—Francesco Balena, Bari, Italy

VB3, VB4 16/32, VB5
Level: Beginning

SPEED UP YOUR CODE USING CHOOSE

You can often use Choose to replace an array and build tables of results evaluated at compile-time instead of run time. For instance, if you need to evaluate the factorial of a number in the range 1 to 10, try this function:

```
Function Factorial(number As Integer) As Long
Factorial = Choose(number, 1, 2, 6, 24, 120, 720, 5040, _
     40320, 362880, 3628800)
End Function
```

—Francesco Balena, Bari, Italy

VB5
Level: Intermediate

GOSUBS ARE SLOW IN COMPILED PROGRAMS

Because GoSubs make for less structured programming, many programmers avoid them. If you compile your VB5 applications to native code, you have one more reason to stay away from them because, curiously enough, GoSubs calls happen to be about five times slower than calls to a regular procedure or function.

—Francesco Balena, Bari, Italy

VB5
Level: Intermediate

"ARRAY" IS NOT A VALID VARIABLE NAME ANYMORE

If you've often used the "array" name for variables, you must revise your code when porting your applications to Visual Basic 5.0. This name has become a reserved keyword and cannot be used for variables. You can easily revise your code with the Replace command in the VB5 IDE—remember to check the "Find whole words only" option.

—Francesco Balena, Bari, Italy

VB4 16/32, VB5 Enterprise Edition
Level: Advanced

RUN AUTOMATION MANAGER AS A HIDDEN TASK

If you use OLE Remote Automation, you must start the Automation Manager on the server computer before any OLE remote communication occurs. By default, this application is visible, but you can hide it so it doesn't appear on the taskbar. To achieve this, change the shortcut to the Automation Manager so it includes the /Hidden switch:

```
C:\Windows\System\AutMgr32.Exe /Hidden
```

Alternatively, you can change the value of a key in the registry. For more information, see article Q138067 in the Microsoft Knowledge Base.

—Francesco Balena, Bari, Italy

VB4 16/32, VB5
Level: Advanced

PROBLEMS WITH POPUP MENUS

If you use popup menus in your applications, you should be aware of a bug present in VB4 16/32 and VB5. If you have two forms and the first one shows the second form modally from within a popup menu, the second form can't show any popup menu.

To fix this bug, use a timer on the first form. Instead of showing the second form from within the popup menu's Click event, activate the timer so it shows the second modal form after some milliseconds. For more information on this bug and its workaround, see article Q167839 in the Microsoft Knowledge Base.

—Francesco Balena, Bari, Italy

VB5
Level: Advanced

Constituent Controls Are Private to UserControl Modules

You cannot directly access constituent controls on a UserControl component from another module of the same project. Constituent controls behave differently from controls on forms, which you can access from any other module using the familiar "Form1.Text1" syntax. If you need to work around this limitation, have each UserControl component expose its controls using a Friend property.

For instance, if the UserControl1 module needs to expose one of its constituent controls, add this property procedure:

```
Friend Property Get TextControl() As TextBox
        Set TextControl = Text1
End Property
```

When you wish to modify the Text property of the control of a particular instance of UserControl1 in the BAS module, write something like this:

```
Sub ClearText(uc As UserControl1)
    uc.TextControl.Text = ""
End Sub
```

—Marco Losavio, Gioia del Colle, Italy

VB4 16/32, VB5
Level: Intermediate

Use a Collection to Filter Out Duplicate Values

This code illustrates how to use a Collection to automatically generate a unique set of values from a set of data containing duplicates. In this example, scan a string array and sort all unique items using a list-box control:

```
Sub Remove_Duplicates(arr() As String)
    Dim i As Long
    Dim RawData As String
    Dim DataValues As New Collection

    On Error Resume Next
    ' Specifically to ignore run-time
    ' error 457 - Duplicate key
    For i = LBound(arr) To UBound(arr)
        RawData = arr(i)
        DataValues.Add RawData, RawData
        ' If Run-time error 457 occurs,
        ' Duplicate key is ignored
    Next
    On Error GoTo 0

    ' Store in List Box
    ' (with Sorted property set to True)
    lstSortedData.Clear
    For Each DataValue In DataValues
        lstSortedData.AddItem DataValue
    Next
End Sub
```

—J.G. Hussey, Fareham, Hampshire, England

VB3
Level: Intermediate

Create "Remotely Controllable" Forms

Sometimes I need to control a VB form while focus is on another one. For example, I want form B to be resized when I press the "OK" button on form A. In every form that must be "remote controllable," I include an invisible text box, such as TextCommand, with the Change procedure containing code like this:

```
Sub TextCommand_Change ()
    Dim msg as string
    msg = Trim$(Me.TextCommand.Text)
    If Len(msg) = 0 Then Exit Sub

    Select Case msg
        Case "COMMAND_RESIZE"
            Call MyFormResize
        Case "COMMAND_REPAINT"
            Call MyFormPaint
        ...
    End Select
    Me.TextCommand = ""
End Sub
```

You can remotely control this form by sending the appropriate value to its TextCommand field:

```
Sub Command1_Click ()
    formB.TextCommand = "COMMAND_RESIZE"
    DoEvents
End Sub
```

Use this code to send messages from an MDI form to its child:

```
Dim f As Form
Set f = Me.ActiveForm
f.TextCommand = "COMMAND_RESIZE"
```

If you program under VB4 or VB5, you might wish to use Public form properties and methods instead.

—Alex Klikouchin, Toronto, Ontario, Canada

VB4 16/32, VB5
Level: Intermediate

Save Form Position and Size Using SaveSetting

SaveSetting and GetSetting make writing application settings a breeze. These two utility functions retrieve and store the current forms position:

```
Public Sub FormPosition_Get(F As Form)
' Retrieve Form F's position from an
' ini/reg file and position it accordingly
Dim buf As String
Dim l As Integer, t As Integer
Dim h As Integer, w As Integer
Dim pos As Integer

buf = GetSetting(app.EXEName, _
     "FormPosition", F.Tag, "")
If buf = "" Then
     ' defaults to centering the form
     F.Move (Screen.Width - F.Width) \ _
          2, (Screen.Height - F.Height) \ 2
Else
     ' extract l,t,w,h and move the form
     pos = InStr(buf, ",")
     l = CInt(Left(buf, pos - 1))
     buf = Mid(buf, pos + 1)
     pos = InStr(buf, ",")
     t = CInt(Left(buf, pos - 1))
     buf = Mid(buf, pos + 1)
     pos = InStr(buf, ",")
     w = CInt(Left(buf, pos - 1))
     h = CInt(Mid(buf, pos + 1))
     F.Move l, t, w, h
End If
End Sub

Public Sub FormPosition_Put(F As Form)
' Write form F's top,left,height and
' width properties to the reg/ini file
' for the application
Dim buf As String
buf = F.left & "," & F.top & "," & _
     F.Width & "," & F.Height
SaveSetting app.EXEName,_
     "FormPosition",  F.Tag, buf
End Sub
```

You should place these routines in a module and call them from the forms' Load and Unload events. You must place the name of the form in its Tag property for these utilities to work:

```
Sub Form_Load()
     FormPosition_Get Me
End Sub
Sub Form_Unload()
     FormPosition_Put Me
End Sub
```

—Rob Parsons, Dee Why Beach, Australia

VB4 16/32, VB5
Level: Beginning

Put VB Intrinsic Constants to Good Use

I've seen several tips using numeric values instead of the corresponding Visual Basic constants. Always use the Visual Basic predefined constants. For example, you can set up a message box using numeric constants:

```
rc = MsgBox(msg, 4 + 32 + 256, "Confirm Delete")
```

But isn't this easier to read?

```
rc = MsgBox(msg, vbYesNo + vbQuestion _
     + vbDefaultButton2, _
     "Confirm Delete")
```

Use these constants for the value of a check box:

```
     VbUnchecked =0
     VbChecked =1
     VbGrayed =2
```

Also, use the string constants instead of the corresponding chr$(ASCII value):

```
     vbTab instead of Chr$(9)
     vbCr instead of Chr$(13)
     vbLf instead of Chr$(10)
     vbCrLf instead of Chr$(13)+Chr$(10)
```

—Pedro Prospero Luis, Odivelas, Portugal

VB3, VB4 16/32, VB5
Level: Intermediate

Test for "File Exist" the Right Way

Dir$ raises a runtime error if you supply it an invalid drive. For example, Dir$ ("d:\win\himems.sys") crashes if drive d: doesn't exist. To check if a file exists, add an error handler:

```
Function FileExist(filename As String) _
     As Boolean
     On Error Resume Next
     FileExist = Dir$(filename) <> ""
     If Err.Number <> 0 Then FileExist _
          = False
     On Error GoTo 0
End Function
```

—Pedro Prospero Luis, Odivelas, Portugal

VB4 16/32, VB5
Level: Intermediate

Procedures that Act on a Group of Controls

You can use the almost-forgotten ability of Visual Basic to have a function or sub with an undetermined number of arguments that do something to a set of controls. For example, you can enable/disable many controls with one sub call:

```
EnableAll True, Text1, Text2, _
     Command1, Command2
```

This procedure iterates on all the controls passed as arguments:

```
Sub EnableAll(Enabled As Boolean, _
     ParamArray objs() As Variant)
          Dim obj As Variant
          For Each obj In objs
               obj.Enabled = Enabled
          Next obj
End Sub
```

—Pedro Prospero Luis, Odivelas, Portugal

VB3, VB4 16/32, VB5
Level: Intermediate

BETTER SCROLLING IMAGES

I enjoyed Joel Paula's "Scrollable Viewport for a Picture" tip ["101 Tech Tips for VB Developers," Supplement to the February 1997 issue of *VBPJ*, page 12], and I'd like to add some improvements to it. First, make the scrollbar's Scroll event update the picture position so it moves while you drag the scroll box. Second, declare these form-level variables:

```
Dim StartX As Long, StartY As Long
Dim Moving As Boolean
```

Finally, declare these three events for PicPicture:

```
Private Sub PicPicture_MouseDown_
    (Button As Integer, Shift As _
    Integer, x As Single, y As Single)
        StartX = x
        StartY = y
        Moving = True
End Sub

Private Sub PicPicture_MouseMove_
    (Button As Integer, Shift As _
    Integer, x As Single, y As Single)
        If Moving Then
            PicPicture.Move _
                PicPicture.Left + x - _
                StartX, PicPicture.Top + _
                y - StartY
        End If
End Sub

Private Sub PicPicture_MouseUp_
    (Button As Integer, Shift As _
    Integer, x As Single, y As Single)
        Moving = False
End Sub
```

Now you can scroll the image with your mouse. Don't forget to test for the borders of the image.

—Pedro Prospero Luis, Odivelas, Portugal

VB3, VB4 16/32, VB5
Level: Intermediate

ENCRYPTED PASSWORDS

These two small, simple, and effective functions easily encrypt/decrypt a text password. The functions take two parameters: a number in the range of 1 to 10 used to alternatively shift up or down the ASCII character by that amount, and the actual Password string.

The EncryptPassword function loops though each character of the DecryptedPassword, checks if its position is odd or even, and shifts the character up or down according to the Number parameter. This makes the encrypted string unreadable. The encrypted password is then scrambled once again using the XOR operator, which makes it even more unreadable.

I chose a limit of Number to be 10, so I don't have to check for invalid ASCII values. The DecryptPassword Function reverses the encryption process by first applying the XOR operator and then shifting:

```
Function EncryptPassword(Number As _
    Byte, DecryptedPassword As String)
Dim Password As String, Counter As Byte
Dim Temp As Integer

Counter = 1
Do Until Counter = _
    Len(DecryptedPassword) + 1
    Temp = Asc(Mid(DecryptedPassword, _
        Counter, 1))
    If Counter Mod 2 = 0 Then
        'see if even
        Temp = Temp - Number
    Else
        Temp = Temp + Number
    End If
    Temp = Temp Xor (10 - Number)
    Password = Password & Chr$(Temp)
    Counter = Counter + 1
Loop
EncryptPassword = Password
End Function

Function DecryptPassword(Number As _
    Byte, EncryptedPassword As String)
Dim Password As String, Counter As Byte
Dim Temp As Integer

Counter = 1
Do Until Counter = _
    Len(EncryptedPassword) + 1
    Temp = Asc(Mid(EncryptedPassword, _
        Counter, 1)) Xor (10 - Number)
    If Counter Mod 2 = 0 Then
'see if even
        Temp = Temp + Number
    Else
        Temp = Temp - Number
    End If
    Password = Password & Chr$(Temp)
    Counter = Counter + 1
Loop
DecryptPassword = Password
End Function
```

—Jeff Bogusz, received by e-mail

VB4 16/32, VB5
Level: Intermediate

FIXING A PROPER CASE TIP

If you use the left arrow key to go back to the beginning of a word and then enter a letter, you get two uppercase letters. Use this code, which takes advantage of the built-in VB4/VB5 StrConv() function, to automatically capitalize words upon entering:

```
Private Sub Text1_Change()
    If Text1.Tag = "" Then
        Text1.Tag = Text1.SelStart
        Text1.Text = StrConv(Text1.Text, vbProperCase)
        Text1.SelStart = Text1.Tag
        Text1.Tag = ""
    End If
End Sub
```

—Tim McBride, Redmond, Washington

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

VB4 32, VB5
Level: Intermediate

Trapping a Double Click for a Toolbar Button

VB4 supports the built-in Win95 Toolbar control, which allows users to add Buttons to the toolbar. The button has a ButtonClick event, but if you want to trap a double-click, there is no ButtonDoubleClick event. To work around this problem, declare two form-level variables:

```
Private mbSingleClicked As Boolean
Private mbDoubleClicked As Boolean
```

In the Toolbars ButtonClick event, add this code:

```
Private Sub Toolbar1_ButtonClick_
     (ByVal Button As Button)
Dim t As Single
t = Timer
If mbSingleClicked = True Then
     mbDoubleClicked = True
     MsgBox "Double Clicked"
Else
     mbSingleClicked = True
     ' allow the user to click the next
     ' time if he wants to double click
     Do While Timer - t < 1 And mbSingleClicked = True
          DoEvents
     Loop
     ' if the user has selected a double
     ' click end the sub.
     If mbDoubleClicked = True Then
          mbSingleClicked = False
          mbDoubleClicked = False
          Exit Sub
     End If
End If
If mbDoubleClicked = False Then
     MsgBox "Single Clicked"
End If

'you can do the processings here, e.g
'If mbDoubleClicked Then
'—— code
'ElseIf mbSingleClicked Then
'—— code
'End If

'when exiting from the sub please
'reintialize the variables, otherwise we
'will end up with the single clicks only
If mbDoubleClicked = False Then
     mbSingleClicked = False
     mbDoubleClicked = False
End If
End Sub
```

—Sushrut Nawathe, Pune, India

VB3, VB4 16/32, VB5
Level: Intermediate

Used Bytes in a Directory

This function returns the number of bytes used on the directory:

```
Function DirUsedBytes(ByVal dirName As _
     String) As Long
Dim FileName As String
Dim FileSize As Currency

' add a backslash if not there
If Right$(dirName, 1) <> "\" Then
     dirName = dirName & "\"
Endif
FileSize = 0
FileName = Dir$(dirName & "*.*")

Do While FileName <> ""
     FileSize = FileSize + _
          FileLen(dirName & FileName)
     FileName = Dir$
Loop
DirUsedBytes = FileSize

End Function
```

You can call the function passing the name of a directory:

```
MsgBox DirUsedBytes("C:\Windows")
```

—Isaias Martinez, Equifisa, Venezuela

VB4 32, VB5
Level: Advanced

Get Useful Disk Information

This function returns the hard disk free bytes, total bytes, percentage of free bytes, and used space. Before calling the function, set the first field of the DISKSPACEINFO structure ("RootPath") to the drive letter:

```
Dim dsi As DISKSPACEINFO
dsi.RootPath = "C:\"
GetDiskSpace dsi
```

The function returns all its results in the other field of the record:

```
' *** Declaratiosn Section ******
Declare Function GetDiskFreeSpace Lib _
     "kernel32" Alias _
     "GetDiskFreeSpaceA" _
     (ByVal lpRootPathName As String, _
     lpSectorsPerCluster As Long, _
     lpBytesPerSector As Long, _
     lpNumberOfFreeClusters As Long, _
     lpTotalNumberOfClusters As Long) _
     As Long

Type DISKSPACEINFO
     RootPath As String * 3
     FreeBytes As Long
     TotalBytes As Long
     FreePcnt As Single
```

```
    UsedPcnt As Single
End Type

' ****** Code Module ******
Function GetDiskSpace(CurDisk As _
    DISKSPACEINFO)
    Dim X As Long
    Dim SxC As Long, BxS As Long
    Dim NOFC As Long, TNOC As Long

    X& = GetDiskFreeSpace_
        (CurDisk.RootPath, SxC, BxS, _
        NOFC, TNOC)
    GetDiskSpace = X&

    If X& Then
        CurDisk.FreeBytes = BxS * _
            SxC * NOFC
        CurDisk.TotalBytes = BxS * _
            SxC * TNOC
        CurDisk.FreePcnt = ((CurDisk._
            TotalBytes CurDisk._
            FreeBytes) / CurDisk._
            TotalBytes) * 100
        CurDisk.UsedPcnt = _
            (CurDisk.FreeBytes / _
            CurDisk.TotalBytes) * 100
    Else
        CurDisk.FreeBytes = 0
        CurDisk.TotalBytes = 0
        CurDisk.FreePcnt = 0
        CurDisk.UsedPcnt = 0
    End If
End Function
```

As is, this routine works with drives with a capacity of 2GB or less; for larger disks, you should use Single variables instead.

—Isaias Martinez, Equifisa, Venezuela

VB4 32, VB5

Level: Advanced

Simulate Pressed Control Key for Multiple Selections in List Box

When selecting items in a normal list box with the MultiSelect property set to 1 - Simple or 2 - Extended, the user needs to press the Control key while clicking on the items in order to continuously select multiple items without also deselecting the items currently selected. This method lets the user select multiple items continuously without pressing the Control key. Place this code in a module:

```
Declare Function GetKeyboardState Lib _
    "user32" (pbKeyState As Byte) _
    As Long
Declare Function SetKeyboardState Lib _
    "user32" (lppbKeyState As Byte) _
    As Long
Public Const VK_CONTROL = &H11
Public KeyState(256) As Byte
```

Place this code in the MouseDown event procedure in a list box (List1) with MultiSelect property set as either Simple or Extended:

```
' Sets the control key state to
' "pressed"
GetKeyboardState KeyState(0)
KeyState(VK_CONTROL) = _
    KeyState(VK_CONTROL) Or &H80
SetKeyboardState KeyState(0)
```

Place this code in any procedure where the pressed Control key is to be released, such as the List1_LostFocus event procedure:

```
' release the control key state from
' "pressed"
GetKeyboardState KeyState(0)
KeyState(VK_CONTROL) = _
    KeyState(VK_CONTROL) And &H7F
SetKeyboardState KeyState(0)
```

—Shangzhi Ren, Omaha, Nebraska

VB3, VB4 16/32, VB5

Level: Intermediate

Get All Matching Files in a Directory Structure

Because this code doesn't use an API, you can easily port it between 16- and 32- bit applications. The DirWalk procedure lets you search an entire directory structure starting at whatever you specify as the argument:

```
ReDim sArray(0) As String
Call DirWalk("OLE*.DLL", "C:\", sArray)
```

The procedure accepts wildcards in the first argument, which is the search pattern for file names. You can even specify multiple search patterns using the semicolon as a separator, as in "OLE*.DLL; *.TLB." The second argument is the location of where to start, and the third argument is an array of strings.

The procedure recursively goes to the deepest level in the directory structure and gets all the matching file names with full path in the array sArray. This array is ReDimed from the function and has as many members as matches found.

To use DirWalk, put two extra controls, FileListBox and DirListBox, on the form. This procedure assumes it's on a form on which there are two controls: FileListBox with name File1, and DirListBox with name Dir1. Keep the controls invisible to improve the speed of the search. Putting these additional controls on a form doesn't cause any overhead because they're part of a basic library of controls for VB:

```
Sub DirWalk(ByVal sPattern As String, _
    ByVal CurrDir As String, sFound() _
    As String)
Dim i As Integer
Dim sCurrPath As String
Dim sFile As String
Dim ii As Integer
Dim iFiles As Integer
Dim iLen As Integer

If Right$(CurrDir, 1) <> "\" Then
    Dir1.Path = CurrDir & "\"
Else
```

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

```
    Dir1.Path = CurrDir
End If
For i = 0 To Dir1.ListCount
    If Dir1.List(i) <> "" Then
        DoEvents
        Call DirWalk(sPattern, _
            Dir1.List(i), sFound())
    Else
        If Right$(Dir1.Path, 1) = "\" _
            Then
            sCurrPath = Left(Dir1.Path, _
                Len(Dir1.Path) - 1)
        Else
            sCurrPath = Dir1.Path
        End If
        File1.Path = sCurrPath
        File1.Pattern = sPattern
        If File1.ListCount > 0 Then
            'matching files found in the
            'directory
            For ii = 0 To File1._
                ListCount - 1
                ReDim Preserve _
                    sFound(UBound(sFound) _
                    + 1)
                sFound(UBound(sFound) - _
                    1) = sCurrPath & _
                    "\" & File1.List(ii)
            Next ii
        End If
        iLen = Len(Dir1.Path)
        Do While Mid(Dir1.Path, iLen, _
            1) <> "\"
            iLen = iLen - 1
        Loop
        Dir1.Path = Mid(Dir1.Path, 1, _
            iLen)
    End If
Next i
End Sub
```

—Atul Ganatra, Omaha, Nebraska

VB4 32, VB5
Level: Advanced

CURRENT COMPUTER NAME ON WINDOWS 95/NT

You often want to know the name of the current computer running Win95 or Windows NT in your VB program. Use this simple wrapper function of a kernel32.dll API function to do the job:

```
Private Declare Function GetComputerNameA Lib "kernel32"_
    (ByVal lpBuffer As String, nSize _
    As Long) As Long

Public Function GetMachineName() As String
    Dim sBuffer As String * 255
    If GetComputerNameA(sBuffer, 255&) _
        <> 0 Then
        GetMachineName = Left$(sBuffer, _
            InStr(sBuffer, vbNullChar) - 1)
    Else
        GetMachineName = "(Not Known)"
    End If
End Function
```

—Jeff Hong Yan, Elmhurst, New York

VB3, VB4 16/32, VB5
Level: Intermediate

SHOW FONTS AS YOU SELECT THEM

To let a user change a font name, load all the fonts into a combo box:

```
Private Sub Form_Load()
    ' Determine the number of screen
    ' fonts.
    For I = 0 To Screen.FontCount - 1
        ' Put each font into list box.
        cboFont.AddItem Screen.Fonts(I)
    Next I
End Sub
```

Make this more useful by letting your users see what the font looks like immediately after selecting it without having to “test” it by typing something:

```
Private Sub cboFont_Click()
    'Set the FontName of the combo box
    'to the font that was selected.
    cboFont.FontName = cboFont.Text
End Sub
```

—Brian Lang, St. Cloud, Minnesota

VB4 16/32, VB5
Level: Intermediate

NEW SHORTCUTS FOR THE VB ENVIRONMENT

In VB5, pressing Ctrl-F3 when the cursor is over a word automatically searches to the next occurrence of that word, bypassing the search dialog. You need to be past the first character of the word for it to work properly.

Another shortcut is that VB4/5 Ctrl-Tab cycles through all your open windows in the IDE often quicker than going to the Window menu.

—Tim Jones, Castlemaine, Victoria, Australia

VB3, VB4 16/32, VB5
Level: Intermediate

SWAP TWO INTEGER VARIABLES

Use this algorithm to swap two integer variables:

```
a = a Xor b
b = a Xor b
a = a Xor b
```

—Alex Bootman, Foster City, California

VB4 32, VB5
Level: Intermediate

Trap Right-Clicks on TreeView's Nodes

The TreeView control gives your apps a good Windows 95 look and feel. However, the VB manual doesn't explain how to trap the right mouse button in a node. The Treeview_MouseDown event occurs before the NodeClick event. In order to display context menus over a node, use this code and define the Key for each node with a letter followed by a number:

```
+ Root (R01)      'the letter gives
|- Child 1 (C01)  'the indication to
|-+ Child 2 (C02) 'the context menu
|  |- Child 2.1 (H01)
|  |- Child 2.2 (H02)

Dim bRightMouseDown as Boolean

Private Sub Form_Load()
    bRightMouseDown = False
End Sub

Private Sub treeview1_MouseDown_
    (Button As Integer, Shift As _
    Integer, X As Single, Y As Single)
    If Button And vbRightButton Then
        bRightMouseDown = True
    Else
        bRightMouseDown = False
    End If
End Sub

Private Sub treeview1_MouseUp_
    (Button As Integer, Shift As _
    Integer, X As Single, Y As Single)
        bRightMouseDown = False
End Sub

Private Sub treeview1_NodeClick_
    (ByVal Node As Node)
    Select Case Left(Node.Key, 1)
        Case "R"
            If Not bRightMouseDown Then
                'do the normal node click,
                'so you must here the code
                'for the node code click
            Else
                'select the node
                treeview1.Nodes(Node.Key).Selected _
                    = True
                'show the popup menu
                PopupMenu mnuContext1
        End If

        Case "C"
            If Not bRightMouseDown Then
                'do the normal node click,
                'so you must here the code
                'for the node code click
            Else
                'select the node
                treeview1.Nodes(Node.Key).Selected _
                    = True
                'show the popup menu
                PopupMenu mnuContext2
        End If

        ' and so on with all other nodes
        ' ....
    End Select
End Sub
```

—Victor Raposo, Coimbra, Portugal

VB3, VB4 16/32, VB5
Level: Intermediate

Run VB Using the SendTo Menu

Adding a "Shortcut to VB.exe" and "Shortcut to VB32.exe" to your "Send To" menu lets you right-click on any VBP project and open it with your choice of VB4 16/32 or VB5.

Go to your VB directory, right-click on VB32.exe, and choose "Create shortcut." When the shortcut file is created, move it into the C:\Windows\Sendto directory—it will be there next time you use it. You might want to add one for WordPad, Word, Excel, or any program that takes an input parameter.

—Warren K. Egger, Houston, Texas

VB4 32, VB5
Level: Intermediate

Getting UserID on Windows 95 and NT

When you want to get the user ID of the current user on the machine running Windows 95 or Windows NT, use this simple wrapper function of an API function to do the job:

```
Option Explicit

Private Declare Function WNetGetUserA _
    Lib "mpr" (ByVal lpName As String, _
    ByVal lpUserName As String, _
    lpnLength As Long) As Long

Function GetUser() As String
    Dim sUserNameBuff As String * 255
    sUserNameBuff = Space(255)
    Call WNetGetUserA(vbNullString, _
        sUserNameBuff, 255&)
    GetUser = Left$(sUserNameBuff, _
        InStr(sUserNameBuff, _
        vbNullChar) - 1)
End Function
```

—Jeff Hong Yan, Elmhurst, New York

VB4 32, VB5
Level: Advanced

Show an HourGlass When Processing Data

Have you ever forgotten to add code to set the MousePointer back to its default at the end of a procedure or function? This technique simplifies showing and resetting the MousePointer without adding code to the end of a procedure or function.

When you create an object from a class, the Initialize event is generated. Any code in the event procedure for that event then executes. This is the first code to execute for the object, before you set any properties or invoke any methods. When the variable goes out of scope, all references to the object are released, the Terminate event is generated for the object, and any code in the Terminate event procedure for that object is executed:

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

' this is an example of a procedure that
' uses the CHourGlass class
Private Sub ProcessData()
     Dim MyHourGlass As CHourGlass
     Set MyHourGlass = New CHourGlass
     'Add processing code here
     Sleep 5000 'This simulates the
     'processing of data
     'Resume processing code here
End Sub

'Create a CHourGlass class with the
'following code:
Private Sub Class_Initialize()
     'Show HourGlass
     Screen.MousePointer = vbHourglass
End Sub

Private Sub Class_Terminate()
     'Restore MousePointer
     Screen.MousePointer = vbDefault
End Sub
```

—Kurt D. Crockett, Denver, Colorado

VB4 16/32, VB5
Level: Beginning

Evaluating Elapsed Minutes

You might need to keep track of the total minutes between one date or time and another. To get the total minutes, use a line like this:

```
lTotalMinutes = Minutes(Now) - _
     Minutes(datStartTime)
```

This function returns the number of minutes since 01/01/1900:

```
Public Function Minutes(d As Date) _
     As Long
     'Minutes since 1900
     Dim lPreviousDays As Long
     Dim lTotalMinutes As Long

     lPreviousDays = d - #1/1/1900#
     lTotalMinutes = _
          (lPreviousDays * 24) * 60
     lTotalMinutes = lTotalMinutes + _
          Hour(d) * 60
     lTotalMinutes = lTotalMinutes + _
          Minute(d)

     Minutes = lTotalMinutes
End Function
```

—Orville P. Chomer, Berwyn, Illinois

VB3, VB4 16/32, VB5
Level: Beginning

Please Stop Printing!

Sometimes I want to print data from a recordset to a report, reading and printing each record. However, it's hard to interrupt that process before it sends all recordsets to the printer queue. Use a Cancel button associated to a flag. Besides the button that starts the printing, create another one named Cancel. You can also set its Cancel property to True, so the user can stop printing by pressing the Esc key. Add a variable in a module:

```
Dim CancelNow As Integer
```

Put this code in the Click event of the Cancel button:

```
Sub cCancel_Click ()
     CancelNow = -1
     DoEvents
End Sub
```

You might even do without the button and simply intercept the Escape key. In this case, set the form's KeyPreview property to True and insert this code:

```
Sub Form_KeyPress (KeyAscii As Integer)
     'if user presses ESC
     If KeyAscii = (27) Then
          CancelNow = -1
          DoEvents
     End If
End sub
```

Finally, add a test for the flag inside the printing loop:

```
'... some code...
'printing a database recordset
Do While Not MyRecordSet.EOF
     Printer.Print MyRecordSet!SomeRecord
     MyRecordSet.MoveNext
     DoEvents
     'stop if Cancel button was clicked
     If CancelNow then Exit Do
Loop
Printer.EndDoc
'... more code...
```

—Carlos Cardoso, Salvador, Bahia, Brazil

VB3, VB4 16/32, VB5
Level: Intermediate

Evaluate Polynomials Faster

The well-known Horner schema lets you calculate polynomial expressions efficiently. To calculate A*x^N + B*x^(N-1) + ... + Y*x + Z (^ means power), simply write this expression as (...((A*x + B)*x + C)*x + ... +Y)*x + Z.

—Alex Bootman, Foster City, California

VB5
Level: Intermediate

Problems with ActiveX Controls Pasted from Clipboard

In VB5, the event is not called when an instance of your UserControl is copied into the clipboard and pasted again, creating a new one. This affects user controls that depend on the UserControl_Resize event to define the control's appearance. To check this behavior, start VB5, create a new ActiveX Control project, and add a text box in the middle of your UserControl area. Add this code to the UserControl Resize Event:

```
Private Sub UserControl_Resize()
    Text1.Move 0, 0, Width, Height
End Sub
```

Close your UserControl and add a Standard EXE project. Create an instance of your new UserControl. You should get a text box filling all your UserControl area. Now copy the UserControl to the clipboard and choose Paste. Note the new instance of your control doesn't resize the text box as it should. To work around this problem, add this code to the UserControl ReadProperties event:

```
Sub UserControl_ReadProperties_
    (PropBag As PropertyBag)
    Call UserControl_Resize
End Sub
```

—Miguel Santos, Aveiro Codex, Portugal

VB4 32, VB5
Level: Advanced

Format or Copy Diskettes Using the Windows API

The Win32 API includes a pair of functions that let you format and copy diskettes from your programs:

```
Private Declare Function SHFormatDrive _
    Lib "shell32" (ByVal hwnd As Long, _
    ByVal Drive As Long, _
    ByVal fmtID As Long, _
    ByVal options As Long) As Long
Private Declare Function GetDriveType _
    Lib "kernel32" _
    Alias "GetDriveTypeA" _
    (ByVal nDrive As String) As Long
```

Add two command buttons to your form, named cmdDiskCopy and cmdFormatDrive, and place this code into their Click events:

```
Private Sub cmdDiskCopy_Click()
    ' DiskCopyRunDll takes two
    ' parameters- From and To
    Dim DriveLetter$, DriveNumber&, _
        DriveType&
    Dim RetVal&, RetFromMsg&
    DriveLetter = UCase(Drive1.Drive)
    DriveNumber = (Asc(DriveLetter) - 65)
    DriveType = GetDriveType_
        (DriveLetter)
    If DriveType = 2 Then  'Floppies, _
        etc
        RetVal = Shell_
            ("rundll32.exe " & _
            "diskcopy.dll," _
            & "DiskCopyRunDll " & _
            DriveNumber & "," & _
            DriveNumber, 1)
    Else   ' Just in case
        RetFromMsg = MsgBox_
            ("Only floppies can be " & _
            "copied", 64, _
            "DiskCopy Example")
    End If
End Sub

Private Sub cmdFormatDrive_Click()
    Dim DriveLetter$, DriveNumber&, _
        DriveType&
    Dim RetVal&, RetFromMsg%
        DriveLetter = UCase(Drive1.Drive)
    DriveNumber = (Asc(DriveLetter) - _
        65)
    ' Change letter to Number: A=0
    DriveType = GetDriveType_
        (DriveLetter)
    If DriveType = 2 Then  _
        'Floppies, etc
        RetVal = SHFormatDrive(Me.hwnd, _
            DriveNumber, 0&, 0&)
    Else
        RetFromMsg = MsgBox_
            ("This drive is NOT a " & _
            "removeable drive! " & _
            "Format this drive?", _
            276, "SHFormatDrive Example")
        If RetFromMsg = 6 Then
            ' UnComment to do it...
            'RetVal = SHFormatDrive_
                (Me.hwnd, _
                '   DriveNumber, 0&, 0&)
        End If
    End If
End Sub
```

Add one DriveListBox control named Drive1:

```
Private Sub Drive1_Change()
    Dim DriveLetter$, DriveNumber&, _
        DriveType&
    DriveLetter = UCase(Drive1.Drive)
    DriveNumber = (Asc(DriveLetter) - _
        65)
    DriveType = GetDriveType_
        (DriveLetter)
    If DriveType <> 2 Then  _
        'Floppies, etc
        cmdDiskCopy.Enabled = False
    Else
        cmdDiskCopy.Enabled = True
    End If
End Sub
```

Be careful: this function can even format the hard disk.

—Duncan Diep, Etobicoke, Ontario, Canada

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

VB4 16/32, VB5
Level: Intermediate

CONSISTENT VERSION NUMBERS

For consistency's sake, use this routine wherever your version numbers appear in code:

```
Public Function GetMyVersion() As String
     ' Turn version info into something
     ' like "1.02.0001"
     Static strMyVer As String
     If strMyVer = "" Then
          ' Only call once for performance
          strMyVer = Trim$(Str$_
               (App.Major)) & "." & _
               Format$(App.Minor, "##00") _
               & "." Format$(App.Revision, _
               "000")
     End If
     GetMyVersion = strMyVer
End Function
```

—Kevin Sandal, Everett, Washington

VB3, VB4 16/32, VB5
Level: Beginning

RIGHT-ALIGN CONTROLS ON FORMS

When creating resizable forms, I like to place command buttons in either the upper-right or lower-right corners. For example, on data entry forms, I place record navigation buttons on the lower-left portion of the form along with an Add New Record button, Delete Record button, and a Find Record button. In the lower-right corner, I place buttons for print previewing reports and closing the form.

Create this subroutine in a module or general declarations section of a form. With Offset, you can vary the distance from the right edge of the form, so you can right-justify more than one button:

```
Sub ButtonRight(X As Control, _
     Frm As Form, Offset as Integer)
          X.Left = Frm.ScaleWidth - _
               X.Width - Offset
End Sub
```

Place two command buttons on the form. In the Form_Resize event, add this or similar code:

```
Private Sub Form_Resize()
     ButtonRight Command1, Me, 0
     ButtonRight Command2, Me, Command1.Width
End Sub
```

—James D. Kahl, St. Louis Park, Minnesota

VB5
Level: Intermediate

HOW OLD ARE YOU?

This function returns the difference between two dates on Years, Months, and Days:

```
Function GetAge(dtDOB As Date, _
     Optional dtDateTo As Date = 0) _
     As String
          'is dtDateto passed ?
          If dtDateTo = 0 Then
               dtDateTo = Date
          End If
          GetAge = Format$(dtDateTo - _
               dtDOB, "yy - mm - dd")
End Function
```

—Emmanuel Soheyli, Downey, California

VB3, VB4 16/32, VB5
Level: Intermediate

VAL DOESN'T WORK ON FORMATTED NUMBERS

Beware of the Val() function. It doesn't correctly recognize formatted numbers. Use CInt(), CDbl(), and so on instead:

```
FormattedString = Format(1250, _
     "General")
     ' = "1,250.00"
Debug.Print Val(FormattedString)
     ' prints 1 !
Debug.Print cDbl(FormattedString)
     ' prints 1250
```

—Peter Gabor, Tel Aviv, Israel

VB3, VB4 16/32, VB5
Level: Intermediate

A SMART ID GENERATOR

I wrote a unique error-proof number generator that greatly simplifies the checking of clients' account numbers or other IDs used by your application. I use it in conjunction with the CheckForValid functions. For example, the CheckForValid returns True for number "203931." The CheckFor Valid returns False for number "209331."

```
Function CheckForValid(Num As Long) _
     As Boolean
' Check for valid number
Result = Num Mod 13
If Result <> 0 Then
     CheckForValid = False
     ' if false then the number is wrong
Else
     CheckForValid = True
     'if true the number is OK
End If
End Function

Function Generate(Num As Long) As Long
'Generates the successor of a valid
'number
If CheckForValid(Num) Then
     Generate = Num + 13
     'if valid Generate
Else
     Generate = -1
     ' Otherwise return -1
End If
End Function
```

—Carlos Santos, Agualva, Portugal

VB4 16/32, VB5
Level: Intermediate

KEEP TRACK OF THE LAST FORM

In your MDI application, you might have many child forms and need a form to go back to the form that called it. In each child form, declare this variable:

```
Public Callingform As Form
```

Before calling a form, set CallingForm to the form that is calling it so that the form being called can remember which form called it. Use this call to call a form:

```
ShowForm frmNextForm, Me
```

frmNextForm is the form that you're calling, and ShowForm is this Global Procedure (declared in a BAS file):

```
Sub ShowForm(frmFormToShow As Form, frmFrom As Form)
     frmFrom.Hide
     frmFormToShow.Show
     Set frmFormToShow.Callingform = frmFrom
End Sub
```

To close a form and go back to the form that called it, use this call:

```
ExitForm Callingform, Me
```

ExitForm is this global procedure:

```
Sub ExitForm(Callingform As Form, ThisForm As Form)
     Unload ThisForm
     Callingform.Show
End Sub
```

Use this procedure for all child forms except when you need to make the call from the main MDI form. In this case, use this call:

```
ShowChild frmChild
```

frmChild is the MDIChild form you're calling, and ShowChild is defined in the MDI form:

```
Private Sub ShowChild(frm As Form)
     frm.Show
     Set frm.Callingform = frmMain
End Sub
```

It's a good idea to call procedures to do these tasks because you might want to put extra processing in these procedures. Note: you should have error-handing routines as well in the procedures.

—Sergio Walter Ruz, Allawah, New South Wales, Australia

VB4 32, VB5
Level: Advanced

RESIZE THE DROP-DOWN LIST AREA OF COMBO BOXES

VB doesn't provide a ListRows property, so if you need to display more than eight default items in a combo box drop-down list, use this procedure to increase the size of the combo box window:

```
Option Explicit

Type POINTAPI
     x As Long
     y As Long
End Type

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

Declare Function MoveWindow Lib _
     "user32" (ByVal hwnd As Long, _
     ByVal x As Long, ByVal y As Long, _
     ByVal nWidth As Long, _
     ByVal nHeight As Long, _
     ByVal bRepaint As Long) As Long
Declare Function GetWindowRect Lib _
     "user32" (ByVal hwnd As Long, _
     lpRect As RECT) As Long
Declare Function ScreenToClient Lib _
     "user32" (ByVal hwnd As Long, _
     lpPoint As POINTAPI) As Long

Public Sub Size_Combo(rForm As Form, _
     rCbo As ComboBox)
     Dim pt As POINTAPI
     Dim rec As RECT
     Dim iItemWidth As Integer
     Dim iItemHeight As Integer
     Dim iOldScaleMode As Integer

     'Change the Scale Mode on the form
     'to Pixels
     iOldScaleMode = rForm.ScaleMode
     rForm.ScaleMode = 3
     iItemWidth = rCbo.Width

     'Set the new height of the combo box
     iItemHeight = rForm.ScaleHeight - _
          rCbo.Top - 5
     rForm.ScaleMode = iOldScaleMode

     'Get the coordinates relative to the
     'screen
     Call GetWindowRect(rCbo.hwnd, rec)
     pt.x = rec.Left
     pt.y = rec.Top

     'then the coordinates relative to
     'the form.
     Call ScreenToClient(rForm.hwnd, pt)

     'Resize the combo box
```

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

```
     Call MoveWindow(rCbo.hwnd, pt.x, _
          pt.y, iItemWidth, iItemHeight, 1)
End Sub
```

—Keith Meulemans, Green Bay, Wisconsin

VB4 32, VB5
Level: Advanced

Show Free Memory Under Win32

If you want to show your users the available memory on the machine, and you're moving from 16 bits to 32 bits, you'll find the API function GetFreeSystemResources has been dropped. You can still do it in VB4/32 and VB5, though. You need to declare the API function and this type in a module:

```
Declare Sub GlobalMemoryStatus Lib _
     "kernel32" (lpBuffer As MEMORYSTATUS)

Type MEMORYSTATUS
     dwLength As Long
     dwMemoryLoad As Long
     dwTotalPhys As Long
     dwAvailPhys As Long
     dwTotalPageFile As Long
     dwAvailPageFile As Long
     dwTotalVirtual As Long
     dwAvailVirtual As Long
End Type
```

Fill the dwlength field with the MEMORYSTATUS type size. Long variables take four bytes, so the total size is 4*8=32 bytes:

```
Dim ms As MEMORYSTATUS

ms.dwLength = Len(ms)
GlobalMemoryStatus ms
MsgBox "Total physical memory:" & _
     ms.dwTotalPhys & vbCr _
     & "Available physical memory:" & _
     ms.dwAvailPhys & vbCr & _
     "Memory load:" & ms.dwMemoryLoad
```

You could also create a class to encapsulate this.

—Luis Ferreira, Oeiras, Portugal

VB3, VB4 16/32, VB5
Level: Intermediate

A Remainder You Can't Miss

I often have several programming irons in the fire at one time. Jumping back and forth from project to project, I sometimes lose track of where I left off within each program. To solve this problem, simply type a phrase without the comment character ('):

```
Left off here 5-5-97, 12:00.
```

The next time you bring up the project, use the "Start With Full Compile" <Ctrl-F5> option. As long as this is the first error within the project, the line will be highlighted and my memory refreshed.

—Mike Saeger, Spokane, Washington

VB4 16/32, VB5
Level: Intermediate

Create an Array on the Fly with the Array Function

The GetRows method retrieves multiple rows of a Recordset (JET) or rdoResultset (RDO) into an array. I often use this feature to transfer data between an OLE Server and client applications. This method uses a Variant type variable as a parameter to store the returned data. It is internally a two-dimensional array and it is treated like one on the client side, but in declaration of the custom method on the OLE server, it looks so much tidier as a variant.

I've tried to pass some additional information such as field names, types, and so on. Usual means of transportation such as collections and regular arrays are either too slow or destroy the symmetry in the declaration. Fortunately, the Array function returns a variant containing an array:

```
Dim A As Variant
A = Array(10,2)
```

—Dejan Sunderic, Etobicoke, Ontario, Canada

VB4 16/32, VB5
Level: Intermediate

Find the Selected Control in an Array of Option Buttons

Use this code to find the index of the currently selected control in an array of option buttons:

```
Function WhichOption(Options As Object) As Integer
' This function returns the index of the
' Option Button whose value is true.

    Dim i
    ' In case Options is not a valid object
    On Error GoTo WhichOptErr
    ' Default to failed
    WhichOption = -1
    ' check each OptionButton in the array. Note this
    ' fails if indices are not consecutive
    For i = Options.lbound To Options.ubound
        If Options(i) Then
            ' when the one set to true is found,
            ' Remember it
            WhichOption = i
            ' and stop looking
            Exit For
        End If
    Next
WhichOptErr:

End Function
```

Call the function with code like this, assuming that iCurOptIndex is an integer and Option1 is the name of an array of OptionButton controls:

```
iCurOptIndex = WhichOption(Option1)
```

Note the function parameter is an object. This function works only if the parameter is an object or a variant.

—Terry Conkright, Colbert, Washington

VB4 16/32, VB5
Level: Intermediate

PACKING CHECK-BOX VALUES INTO A SINGLE INTEGER

Use this code to find the binary representation of the currently checked check boxes:

```
Function WhichCheck(ctrl As Object) As _
     Integer
' This function returns the binary value
' of an array of controls where the
' value is 2 raised to the index of each
' checked control
' ie element 0 : 2 ^ 0 returns 1
'elements 0 and 2 : 2^0 + 2^2 returns 5

     Dim i
     Dim iHolder
     ' in case ctrl is not a valid object
     'default to failure return =0 on
     'fail
     On Error GoTo WhichCheckErr

     ' find the binary representation of
     ' an array of check box controls
     For i = ctrl.LBound To ctrl.UBound
          If ctrl(i) = 1 Then
               ' if it is checked add in its
               ' binary value
               iHolder = iHolder Or 2 ^ i
          End If
     Next
WhichCheckErr:
     WhichCheck = iHolder

End Function
```

Call the function with code like this:

```
iCurChecked = WhichCheck(Check1)
```

Check1 is an array of check boxes, and iCurChecked is an integer. Here's the "dual" routine that sets the state of all the check boxes in a control array given an Integer that holds their binary representation:

```
Sub SetChecked(ctrl As Object, _
     iCurCheck%)
' This sub sets the binary value of an
' array of controls where iCurChecked is
' 2 raised to the index of each checked
' control
     Dim i
     ' in case ctrl is not a valid object
     On Error GoTo SetCheckErr

     ' use the binary representation to
     ' set individual check box controls
     For i = ctrl.LBound To ctrl.UBound
          If iCurCheck And (2 ^ i) Then
               ' if it is checked add in its
               ' binary value
               ctrl(i).Value = 1
          Else
               ctrl(i).Value = 0
          End If
     Next
SetCheckErr:

End Sub
```

Call the sub with code like this:

```
Call SetChecked(Check1, iDesired)
```

Check1 is an array of checkboxes, and iDesired is a binary representation of the desired settings.

—Terry Conkright, Colbert, Washington

VB4 16/32, VB5
Level: Intermediate

CONDITIONALLY COMPILE YOUR CODE

Most developers know about VB4's Conditional Compilation feature, where you can declare Windows APIs for 16-bit and 32-bit operating systems:

```
#If Win#32 then
     'If running in 32-bit OS
     Declare SomeApi....
#Else
     'If running in 16-bit OS
     Declare SomeApi
#End IF
```

This same feature applies not only to Windows API statements, but also to your own functions:

```
#If Win32 Then
     Dim lRc&
     lRc& = ReturnSomeNumber(35000)
#Else
     Dim lRc%
     lRc% = ReturnSomeNumber(30000)
#End If

#If Win32 Then
     Private Function ReturnSomeNumber_
          (lVar&) As Long
          ReturnSomeNumber = 399999
#Else
     Private Function ReturnSomeNumber_
          (lVar%) As Integer
          ReturnSomeNumber = 30000
#End If

End Function
```

—Carl Denton, Marietta, Georgia

VB4, VB5
Level: Intermediate

REDUCE FLICKERING DURING FORM LOADING

When loading a form, reduce the "flicker" and "flash" of the GUI by using these Windows API functions:

```
'Declarations Section
#If Win32 Then
     Declare Function LockWindowUpdate _
          Lib "user32" _
          (ByVal hwndLock As Long) As Long
#Else
     Declare Function LockWindowUpdate _
          Lib "User" _
          (ByVal hwndLock As Integer) _
          As Integer
#End If

Public Sub LoadSomeForm()

     ' When loading a form lock the
     ' window update to stop the
     ' distracting flashing.

     'stop the updating of the GUI
     LockWindowUpdate frmTest.hWnd
     'Show the form
     frmTest.Show
     ' Load and populate form code here

     'Always,Always Undo the update lock
     LockWindowUpdate 0
End Sub
```

—Carl Denton, Marietta, Georgia

VB4 16/32, VB5
Level: Advanced

Hide the Record Selector in a Data-Bound Grid

To stop the selection bar on a data-bound grid from moving when navigating through records in the bound RDC or the rows on the grid, use the API call LockWindowUpdate(gridname.hwnd) before navigating the record pointer, and LockWindowUpdate(0) after the navigation:

```
'Declarations Section
#If Win32 Then
     Declare Function LockWindowUpdate _
          Lib "user32" _
          (ByVal hwndLock As Long) As Long
#Else
     Declare Function LockWindowUpdate _
          Lib "User" _
          (ByVal hwndLock As Integer) _
          As Integer
#End If

Private Sub cmdHideSelector_Click()
     LockWindowUpdate DBGrid1.hWnd
End Sub

Private Sub cmdShowSelector_Click()
     LockWindowUpdate 0
End Sub
```

—Colin Myles, Stratford-upon-Avon, Warwickshire, England

VB4 16/32
Level: Intermediate

Use Popup Menus in Windows without Title Bar

When you set the ControlBox property to False and BorderStyle as a fixed window, you can get a window without the title bar. If you add a menu to the window for use as a popup menu, the title bar appears again. You can place the menu on another form to avoid this problem:

```
Private Sub Command1_Click()
     Dim frm As New frmMenu
     Load frm
     frm.PopupMenu frm.mnutest
     'select specific code
     Unload frm
End Sub
```

This behavior has been fixed in VB5.

—Hou Yantang, Xi'an, China

VB3, VB4 16/32, VB5
Level: Intermediate

Compare Different Instances of the Same Component

When you get a cool sample of a VB component and you simply can't re-create all of its characteristics in your app, don't go nuts. Most of the time, you can detect the differences between the component in the sample and the component in your app by using Windows Notepad or some other editor. Simply open both of them and compare the properties corresponding to the components you're having problems with. In VB3, both forms must be in text format for the editor to see them. This isn't an issue with VB4 and VB5, which always save forms as ASCII files.

—Freud Jone Oliveira, Asa Norte, Brazil

VB3, VB4 16/32, VB5
Level: Intermediate

Get Date and Time Delimiters without API Functions

Use these easy algorithms to obtain the current Date, Time, and Decimal delimiters used by Windows without resorting to Locale Settings or API calls:

```
DateDelimiter = Mid$(Format(Date, _
     "General Date"), 3, 1)
TimeDelimiter = Mid$(Format(0.5, _
     "Long Time"), 3, 1)
DecimalDelimiter = Mid$(Format(1.1, _
     "General Number"), 2, 1)
```

—Rob Parsons, Dee Why Beach, Australia

VB4 16/32, VB5
Level: Intermediate

Prevent Errors when Using GetSetting

Using Visual Basic's GetSetting function might cause an error, particularly in certain situations when using it under 16-bit operating systems with INI files. If there's no specific entry in the INI file, you might get an error message such as "Invalid procedure call." You can use this routine, which wraps an error handler around the built-in function:

```
Public Function GetRegSetting(AppName _
     As Variant, Section As Variant, _
     Key As Variant, Optional Default _
     As Variant) As Variant

' the default value is not assumed to be
' an Object type otherwise you should
' use Set statement
Dim tmpValue As Variant

' set default value
' if no value was passed, this gives an
' empty variant
If Not IsMissing(Default) Then _
     tmpValue = Default

' this is for trapping possible errors
On Error Resume Next

' let's use VB's function
tmpValue = GetSetting(AppName, _
     Section, Key, tmpValue)

' after possible errors the call
' continues here with the preset default
' value
GetRegSetting = tmpValue

End Function
```

—Jussi Mattila, Helsinki, Finland

VB3, VB4 16/32, VB5
Level: Beginning

Duplicate Lines of Code without Syntax Errors

Many times when I code similar syntax with slight modifications on each line, I like to make a template of the core syntax, quickly paste a copy of it however many times I need it, and then go back and edit each line. Many times, however, the core syntax generates an error by the VB editor. You can get around this problem by commenting the core syntax line out before you paste the template. Once you finish editing the templates, simply go back and remove the comment delimiter. This is especially easy under VB5, which has a Block Uncomment command. For example, say you're reading a recordset to populate a collection:

```
While Not mRS.EOF
     oObject.FName = mRS!FName
     oObject.LName = mRS!LName
     oObject.Phone = mRS!Phone
     .
     .
     cCollection.Add oObject, oObject.FName
Wend
```

If your object has 20 or 30 properties, it would be quicker to code this core syntax:

```
'    oObject. = mRS!
```

Copy it, paste it 20 or 30 times, go back and type the property and field names in, and remove the comment delimiter. The comment delimiter lets you go back and edit each line in whatever order you like and not have to worry about generating a syntax error.

—Trey Moore, San Antonio, Texas

VB4 32
Level: Intermediate

A Shortcut to Load the Last Project into VB

Most of the time, I wish to start VB and resume the last project I was working on, but I don't like to litter my desktop with program icons for works in progress. As a solution install this program in compiled form on your desktop. You can probably adapt it to other versions of VB as well as to other programs that store this information in an INI file:

```
Option Explicit

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

Public Sub Main()
     Dim temp As String, rVal$, tmp _
          As Long
     rVal$ = String$(256, 0)
     tmp = GetPrivateProfileString_
          ("Visual Basic", _
          "vb32location", "", rVal$, _
          ByVal Len(rVal$) - 1, _
          "c:\windows\vb.ini")
     temp = Left$(rVal$, tmp)
     rVal$ = String$(256, 0)
     tmp = GetPrivateProfileString_
          ("Visual Basic", "RecentFile1", _
               "", rVal$, ByVal Len(rVal$) _
               - 1, "c:\windows\vb.ini")
     temp = temp & " """ & Left$(rVal$, _
          tmp) & """"
     Shell temp, 1
     End
End Sub
```

—Andrew Ladner, Delray Beach, Florida

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

VB4 32, VB5
Level: Intermediate

LISTVIEW CONTROLS DON'T ACCEPT NUMERIC KEYS

In a collection object, such as the ListItems collection from the ListView control, or simply a generic VB collection object, you can specify a key to uniquely identify the item. Documentation states the key can be any String expression. What if the key needs to be a numerical string? In a ListView control, you might not set a numeric key. Even if you try to set a key equal to Str$(<Numeric Variable>), you'll receive an error message. When displaying the results of a recordset with a ListView, the key would be the perfect place to hold the primary key for the row, if applicable. Because ListView doesn't have an ItemData property, the key is the only place to hold it. The solution is simple: append the string "key" to your Numeric Key, and use the Val function to retrieve its value:

```
Set itemX = lvPeople.ListItems.Add_
    (, , strName)
'Set the Key
itemX.Key = Str$(rstPeople!PersonID) & "key"
```

Use this code to retrieve the key:

```
lKey = Val(lvPeople.ListItems(nIndex).Key))
```

—Steve Danielson, Raleigh, North Carolina

VB4 16/32, VB5
Level: Beginning

SHOWING "&" CHARACTERS IN LABELS

If you want to show the character "&" instead of having it work as a marker for the access key, set the property "UseMnemonic" to False. This property is useful, for instance, when using Label controls to show data from a database. You can also get literal "&" characters by using double ampersands in the Caption property to display a single "&."

—S. Edwin Gnanaraj, Madras, India

VB3, VB4 16/32, VB5
Level: Beginning

CREATE TEMPORARY FILES

I'm developing a database program that deals with many auxiliary files at the same time. Everyone coding database programs must create some temporary files to produce an output from SQL or a temporary database to manipulate those records efficiently. I decided to create a FileAux function that returns one temporary file name. If I need to create more than one file into the same process, I simply store those names into variables dimensioned before:

```
Function FileAux(Ext As String) _
    As String
    Dim i As Long, X As String
    If InStr(Ext, ".") = 0 Then
        Ext = "." + Ext
    End If

    'Look for the previous files in the
    'HardDisk
    i = 0
    Do
        X = "Aux" + Format$(i, "0000") _
            + Ext
        If FileExists(X) Then
            i = i + 1
        Else
            Exit Do
        End If
    Loop
    FileAux = X
End Function
```

This function uses the "FileExists" function:

```
Function FileExist(filename As String) _
    As Boolean
    FileExist = Dir$(filename) <> ""
End Function
```

Here's an example of its usage:

```
Sub Test()
    Dim File1 As String, File2 As _
        String, File3 As String
    Dim DB1 As database, DB2 As DataBase
    Dim FileNum As Integer
    File1 = FileAux("MDB")
    Set DB1 = CreateDataBase(File1)
    File2 = FileAux("MDB")
    Set DB2 = CreateDataBase(File2)
    File3 = FileAux("TXT")
    FileNum = FreeFile
    Open File3 For OutPut As FileNum
    'Your code
    ' ...
    Close FileNum
End Sub
```

File1, File2, and File3 should be "Aux0001.MDB," "Aux0002.MDB," and "Aux0001.TXT," respectively.

—Luis Orlindo Tedeschi, Piracicaba, Brazil

VB3, VB4 16/32, VB5
Level: Beginning

MOUSE EVENTS DON'T FIRE IF ENABLE IS FALSE

MouseMove events don't occur when the control's Enabled property is set to False. My method tackles this problem and is useful when you want to display the Tooltips or Notes on the status bar, whether the control is enabled or disabled.

If the Enabled property is set to False, the control placed behind the control's MouseMove event will be fired when you move the cursor on the control. Duplicate code you write in the Command1_MoseMove in the Label1_MouseMove. Now it works even though your Command1 button is disabled. Place these controls on Form1:

- Command1(0), Command1(1)—Command1 is the control array.
- Label1(0), Label1(1)—Labels set behind the command1.

• SSPanel1—Acts as status bar.

Add this code:

```
Private Sub Form_Load()
Dim i As Integer
For i = 0 To 1
    Label1(i).Left = Command1(i).Left
    Label1(i).Top = Command1(i).Top
    Label1(i).Width = Command1(i).Width
    Label1(i).Height = _
        Command1(i).Height
Next i
Command1(0).enabled = false
Command1(0).Tag = "Button to Add"
Command1(1).Tag = "Button to Modify"
Command1(0).Caption = "&Add"
Command1(1).Caption = "&Modify"

End Sub

Private Sub Label1_MouseMove(Index As _
    Integer, Button As Integer, Shift _
    As Integer, X As Single, Y As _
    Single)
    SSPanel1.Caption = _
        Command1(Index).Tag
End Sub

Private Sub Command1_MouseMove(Index _
    As Integer, Button As Integer, _
    Shift As Integer, X As Single, Y _
    As Single)
    SSPanel1.Caption = _
        Command!(Index).tag
End Sub
```

—S. Edwin Gnanaraj, Madras, India

VB4 16/32, VB5
Level: Intermediate

Disable the Default Popup Menu on Text Boxes

Some controls in VB4 and VB5, such as the TextBox control, have a default context menu that appears when you right-click on the control. If you want to pop up your own context menu, no property or method of those controls exists to disable the default behavior.

To solve this problem, place code in the Mouse_Down event, which disables the target control. Pop up your context menu, and then re-enable the control. Here's the method in PopContextMenu:

```
Sub PopContextMenu(argoControl As _
    Control, argoMenu As Control)
        argoControl.Enabled = False
        PopupMenu argoMenu
        argoControl.Enabled = True
End Sub
```

Call it in the MouseDown event of a text box named Text1 for a menu called MyMenu:

```
Private Sub Text1_MouseDown(Button As _
    Integer, Shift As Integer, X As _
    Single, Y As Single)
        If Button = vbRightButton Then
            PopContextMenu Text1, MyMenu
        End If
End Sub
```

—William Jordan, Acworth, Georgia

VB3, VB4 16/32, VB5
Level: Intermediate

A Taskbar-Compliant Version of CenterForm

To center a form, you only need one API call, no UDTs, and two constants. This solution is based on the fact that GetSystemMetrics reflects real estate taken up by the taskbar and the Microsoft Office shortcut bar:

```
Public Const SM_CXFULLSCREEN = 16
Public Const SM_CYFULLSCREEN = 17

#If Win32 then
    Declare Function GetSystemMetrics _
        Lib "user32" _
        (ByVal nIndex As Long) As Long
#Else
    Declare Function GetSystemMetrics _
        Lib "User" _
        (ByVal nIndex As Integer) _
        As Integer
#End If

Public Sub CenterForm(frm As Form)
    frm.Left = Screen.TwipsPerPixelX * _
        GetSystemMetrics_
            (SM_CXFULLSCREEN) / 2 _
            - frm.Width / 2
    frm.Top = Screen.TwipsPerPixelY * _
        GetSystemMetrics_
        (SM_CYFULLSCREEN) / 2 _
        - frm.Height / 2
End Sub
```

—Dave Michel, Minneapolis, Minnesota

VB3, VB4 16/32, VB5
Level: Beginning

A String Cleaner

At times it's useful to have a function that cleans a string of unwanted characters. This small function accepts the string you want to clean and the characters you want to get rid of:

```
Function StringCleaner(s As String, _
    Search As String) As String
        Dim i As Integer, res As String
        res = s
        Do While InStr(res, Search)
            i = InStr(res, Search)
            res = Left(res, i - 1) & _
                Mid(res, i + 1)
        Loop
        StringCleaner = res
End Function
```

—Roop Datta, Palatine, Illinois

VB4 16/32, VB5
Level: Beginning

TEST OBJECTS USING TYPENAME

You can determine the class of an object using the TypeName function instead of the If TypeOf statement. Use the TypeOf statement to determine the type of object:

```
If TypeOf myObject is myType then
    ... do something
End If
```

You can do the same with this code:

```
if TypeName(myObject) = "myType" Then
    ....do something ....
End If
```

The advantage is that you don't have to include in your project all the classes (or OCXs) that you manage. This is a good approach when writing general-purpose routines, and moreover, you can use TypeName in compound tests and Select Case blocks.

—Andrea Adami, Lugagnano, Italy

VB4 16/32, VB5
Level: Intermediate

APPEND A STRING TO A TEXT BOX

Use this code to cause a TextBox control to automatically scroll down as you concatenate new text:

```
'Select the end of the text
MyTextBox.SelStart = Len(MyTextBox.Text)
'Put the new text there
MyTextBox.SelText = NewText$
```

—William D. Marquis, Apple Valley, California

VB3, VB4 16/32, VB5
Level: Beginning

DOUBLE CHECK VAL ARGUMENTS

When using the Val function, VB has a quirk that causes a type mismatch error. For example, Val("25%") correctly returns 25, whereas Val("2.5%") misinterprets the string and returns a type mismatch error. This happens only when there is a decimal point and a "%"or "&" character in the string. To work around this, remove these characters before passing the string to the Val function.

—Frank Barbato and Krystyna Zyzdryn, North Palm Beach, Florida

VB4 32, VB5
Level: Advanced

INTERNET SHORTCUTS

VB5 App Wizard can create a Web Browser form, but it works only with Microsoft Internet Explorer and you must distribute SHDOCVW.DLL. If you use Windows' ShellExecute function to execute an Internet Shortcut file, Windows executes the default browser and goes to the specified URL. This method works with both the Microsoft and Netscape browsers if they're properly registered in the Windows Registry, and you don't have to distribute any DLL:

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

' frm : ShellExecute needs a Window
' handle. You can use the handle of your
' main window
' sUrl : This is the name and path of a
' .url file (Internet shortcut file)
' that points to your page, for example
' c:\MyWebPage.url Use Internet Explorer
' to create the shortcut file

Public Sub GoToMyWebPage(frm as Form, _
    sUrl as string)
    Dim lRet as Long
    lRet = ShellExecute(frm.hwnd, _
        "open", sUrl, vbNull, _
        vbNullString, SW_SHOWNORMAL)
    If lRet <= 32 Then
    ' there was an error. Some of the
    'errors that can be returned by
    ' ShellExecute are:
        ' ERROR_FILE_NOT_FOUND = 2&
        ' ERROR_PATH_NOT_FOUND = 3&
        ' ERROR_BAD_FORMAT = 11&
        ' SE_ERR_NOASSOC = 31
        ' SE_ERR_OOM = 8
    Else
        ' your browser is running!
    End If
End Sub
```

—José Rodríguez Alvira, San Juan, Puerto Rico

VB3, VB4 16/32, VB5
Level: Intermediate

A Form-Level DataChanged Property

Ever wished you could use the Save parameter outside of a Validate event? Have you noticed that if a bound control is changed and you set DataChanged back to False, Save is still True on Validate? Solve both problems by adding a public function:

```
Public Function FormChanged(frm As Form)
     On Error Resume Next
     Dim i As Integer, dChanged As _
          Boolean
     FormChanged = False
     For i = 0 To frm.Controls.Count - 1
          'for controls with no DataChanged
          'property
          dChanged = frm.Controls(i)_
               .DataChanged
          If Err <> 0 Then
'almost certainly is
' OjectDoesntSupportThisPropertyOrMethod
               Err = 0
          Else
               If dChanged = True Then
                    FormChanged = True
                    Exit Function
               End If
          End If
     Next
End Function
```

Use this function any time by simply writing If FormChanged(Me) Then, or however you wish, and you can use it in the Validate routine with Save = FormChanged(Me).

—Matthew Brown, Cambridge, Illinois

VB4 32, VB5
Level: Intermediate

Open Help File Contents

Most programmers want their applications to look like commercial software, so they want to add their own Help files. How do you open the contents of a Windows Help file in your application? Simply paste in this code, which uses Win32 API function:

```
' -- Declares
Const HELP_CONTENTS = &H3&
' Display contents
Declare Function WinHelp Lib "user32" Alias "WinHelpA" _
     (ByVal hwnd As Long, _
     ByVal lpHelpFile As String, _
     ByVal wCommand As Long, _
     ByVal dwData As Long) As Long

' -- Code
Sub OpenHelpFile(HelpFileName As String)
     ' HelpFileName is the path of the
     ' help file.
     WinHelp hwnd, HelpFileName, _
          HELP_CONTENTS, 0
End Sub
```

—Huang Xiongbai, Shanghai, China

VB3, VB4 16/32, VB5
Level: Beginning

Enforce Design-Time Size for MDI Forms

Because MDI forms don't have a border property, the user can drag the borders and distort the size of the MDI form. If the user tries to resize the form, I want the form to revert to its design-time size. To accomplish this, use this procedure for the MDIForm_Resize() event:

```
Private Sub MDIForm_Resize()
     ' Stop resizing of MDI forms by
     ' dragging borders and reposition
     ' the MDI form. Only do this if the
     ' MDI form is displayed as a Normal Window
     If WindowState = 0 Then
          ' Your MDI form's design height
          Me.Height = 6900
          ' Your MDI form's design width
          Me.Width = 10128
          ' Your MDI form's design left
          ' position
          Me.Left = 1020
          ' Your MDI form's design top
          ' position
          Me.Top = 1176
          ' You may also use a Move method
          ' to change all properties in one
          ' single command
     End If
End Sub
```

—Joseph J. Janus, New Castle, Pennsylvania

VB3, VB4 16/32, VB5
Level: Beginning

Fast Database Lookups

Visual Basic doesn't have a procedure like the DLookUp function that Access has. You can use this procedure in VB to receive the Name of an object by ID:

```
Public Function MyDLookUp(Column As _
     String, TableName As String, _
     Condition As String) As Variant
Dim Rec As Recordset
On Error GoTo MyDlookUp_Err

' gCurBase is a global variable that
' holds the database that is currently
' opened
Set Rec = gCurBase.OpenRecordset_
     ("Select * From " & TableName)
Rec.FindFirst Condition
If Not Rec.NoMatch Then
     ' return the requested field if
     ' matching
     MyDLookUp = Rec(Column)
     Exit Function
End If

' return -1 if there is no match, or any
' other error
MyDlookUp_Err:
```

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

```
    MyDLookUp = -1
End Function
```

—Valery Belfor, Jerusalem, Israel

VB3, VB4 16/32, VB5
Level: Intermediate

Cheap Focus Tracking

The Lost_Focus and Got_Focus events are the most-used events for implementing validation and text highlighting. Wouldn't it be nice to respond instantly to these events and to do it from a single routine for all controls without the aid of a subclassing control?

Here's the answer. Place a timer control on your form, set its Interval property to 100 and set Enabled = True. Name the control tmrFocusTracking. Code its Timer event like this:

```
Private Sub tmrFocusTracking_Timer()
    Dim strControlName As String
    Dim strActive As String
    strControlName = _
        Me.ActiveControl.Name

    Do
        strActive = Me.ActiveControl.Name
        If strControlName <> strActive _
            Then
            Print strControlName & _
                " - Lost Focus", _
                strActive & " - Got Focus"
            strControlName = strActive
        End If
        DoEvents
    Loop
End Sub
```

To implement universal highlighting, replace the Print statement with this code:

```
Me.Controls(strActive).SelStart = 0
Me.Controls(strActive).SelLength = _
    Len(Me.Controls(strActive))
```

To implement validation, replace the Print statement with a call to a validation routine. Use strActive in a Select Case structure. At the moment where the Print statement would occur, strActive is equal to the control that just got focus, and strControlName holds the name of the control that just lost focus.

Don't place this routine in anything but a timer; otherwise, your program hangs once the routine is called. Even the timer here never makes it to a second interval. For a given control, don't write validation code both in the Got_Focus/Lost_Focus events, and in code called by this routine. Doing so might cause unpredictable results.

—John S. Frias, Santa Maria, California

VB3, VB4 16/32, VB5
Level: Beginning

A Form That Won't Close

If you set a form's ControlBox property to False, the Minimize and Maximize buttons also become invisible. Suppose you want to provide functionality to the user to maximize and minimize the form, but not to close the form using the control box. Simply add this code to the Query_Unload event:

```
' uncomment next line in VB3
' Const vbFormControlMenu = 0
Private Sub Form_QueryUnload(Cancel As _
    Integer, UnloadMode As Integer)
        If UnloadMode = vbFormControl_
            Menu Then
            Cancel = True
        End If
End Sub
```

—A.G.K. Kishore, New Delhi, India

VB3, VB4 16/32, VB5
Level: Beginning

Change a Property in a Group of Controls

You can easily make a group of controls visible or invisible. At design time, select all controls you want to make visible or invisible during the execution. Press F4, and assign the Tag property a name for the group, such as "Group1." When you wish to make that group visible, run this code:

```
For ind = 0 To Formname.Controls.Count _
    - 1
    If Formname.Controls(ind).Tag = _
        "Group1" Then
        Formname.Controls(ind).Visible _
            = True
    End If
Next
```

—Rogerio Zambon, Porto Alegre, Brazil

VB3, VB4 16/32, VB5
Level: Intermediate

Round and Format the Easy Way

Do you sometimes need to format rounded numbers to a specific number of digits? You can accomplish this in one step:

```
n = 12.345
Format(n, "0.00\0")
'returns   "12.350"
Format(n, "0.\0\0")
'returns "12.00"
Format(0.55, "#.0\0") 'returns   ".60"
```

—Peter Gabor, Tel Aviv, Israel

VB4 16/32, VB5
Level: Intermediate

QUICK JUMPS TO THE DECLARATION SECTION

I often want to get to the General code section of a form quickly, either to hack module-level variables or to grab the procedure pull-down to find that function whose name I forgot. Double-clicking on the form gives you Form procedures. Clicking on "View Code" from the menu or project box does the same unless no form is open.

To solve this, put an oval Shape control with an obnoxious color on all your forms and make it invisible. Of course, nothing is invisible in design mode, and because shapes have no event procedures, double-clicking on it takes you to the General/Declarations page every time.

Note that under VB4 and VB5, you might enforce Full Module View, in which case you can jump to the declaration section simply by pressing Ctrl-Home.

—Randal J. King, Downers Grove, Illinois

VB3, VB4 16/32, VB5
Level: Intermediate

BE AWARE, IT'S NOT C!

VB developers that program with C language might be confused by a feature in the language. Consider this code:

```
Dim x As Integer
Dim y As Integer
Dim z As Integer

x = 10
y = 20
z = 0

'Assume function max returns the maximum
'of  the two
if  (z = max(x, y)) > 0 then
    Msgbox CStr(z)
Else
    Msgbox "How Come?"
End if
```

In the code, you would expect the message box to display 20, as it would do in C. VB, however, compares "z" with the RHS (right-hand side) even before the assignment, irrespective of the brackets. Be careful.

—Baskar S. Ganapathy, Walnut Creek, California

VB4 16/32, VB5
Level: Advanced

PORTING VB-SPECIFIC OBJECTS TO ACCESS 8.0

Code portability is extremely important, especially now that VBA is available in so many products. Some problems might occur if you want to use VB4 or VB5 code in Access 8 without making any changes. If the code you are porting references some VB-specific object, object method, or property, Access complains when you compile the imported code. For instance, the "App" object exists in VB but not in Access. Therefore, this statement won't compile under Access:

```
Msgbox App.Title
```

To get around this problem, create a class in Access 8 to "replace" the VB4 object, method, or property not available in Access. In this case, create a clsVBShadow class in Access:

```
Option Compare Database
Option Explicit

Private strTitle As String
Private strEXEName As String
Private strMin As String

Private Sub Class_Initialize()
    strTitle = "Microsoft Access"
    ' Or whatever you want to return.
    strEXEName = "?"
    strMin = "?"
End Sub

Public Property Get Title() As String
    Title = strTitle
End Property

Public Property Get EXEName() As String
    EXEName = strEXEName
End Property

' complete with more properties...
```

In any module in your Access 8 application, declare this global object:

```
Public App As New clsVBShadow
```

At this point, any code you import from VB4 that accesses the App object still works unchanged, and you can replace each method or property of that object with what you want.

—Warren Roscoe, Cape Town, South Africa

VB3, VB4 16/32, VB5
Level: Intermediate

STRING SURPRISE

I was developing a CGI application that read a database and pieced together the fields into a string for representation as an HTML form. The surprise came when I found how slow it was—20 seconds was unacceptable for the task. I first suspected the database access and thought there was no way to improve it; however, upon investigating further, it turned out to be the loop that concatenated the fields into the string. A simple change took the run time of the routine down to about one second. In this simplified example, instead of writing this code:

```
For i = 1 To 10000
    strHTML = strHTML & strField & vbTab
Next i
```

Break it down into something like this:

```
For i = 1 To 100
    strTemp = ""
```

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

```
     For j = 1 to 100
          strTemp =  strTemp & strField _
               & vbTab
     Next j
     strHTML =  strHTML & strTemp
Next i
```

Admittedly, the number of concatenations is high, but the difference is astounding. In the 16-bit world, the second example is about 20 times faster than the first on my machine. However, in VB 32-bit, my Pentium 133 with 32 MB of RAM takes 48 seconds for the first case and less than one second for the second case. I hesitate to suggest a precise speed improvement because of the complexity of Windows 95 and different hardware, but if you must concatenate large numbers of strings, this could become a make-it-or-break-it problem.

—Nick Snowdon, Delta, British Columbia, Canada

VB3, VB4 16/32, VB5
Level: Intermediate

USE BACKQUOTES INSTEAD OF APOSTROPHES

Often when using Transact-SQL, I want to capture comments from a user in a text box and send them to the database. However, if the user types an apostrophe in the text box, a runtime error is generated when the update is processed because SQL Server thinks the apostrophe is being used to mark the end of a string.

To get around this problem, intercept the user's keystrokes in the KeyPress event and exchange the apostrophe with an "upside-down" quote mark (ASCII(145)) like this:

```
Private Sub Text1_Keypress(KeyAscii as Integer)
     If KeyAscii = 39 Then
          KeyAscii = 145
     End If
End Sub
```

Alternatively, you might decide to substitute all occurrences of single quotes into backquotes immediately before sending them to SQL Server.

—Mike McMillan, North Little Rock, Arkansas

VB4 16/32, VB5
Level: Intermediate

SPREAD UPGRADES OVER THE NETWORK

I design VB applications for approximately 300 employees in a networked environment. It's difficult to keep those PCs up to date with the most current version of an app, so I use VB's auto-incrementing version-numbering feature to have the app test if a newer version is available when it launches.

Set the app to auto-increment when it's compiled. Store the setup/upgrade files on a networked drive (be sure to use the UNC path rather than drive letters), and include an uncompressed INI file that lists the newest version available. Then embed this code into the Form_Load event:

```
Open IniFile$ For Input As #1
Line Input #1, sUpgradeVersion$
Close #1

If sUpgradeVersion > (Format(App.Major, "00") & "." & _
     Format(App.Minor, "00") & "." & _
     Format(App.Revision, “0000”)) Then
          ' shell out to networked upgrade
          ' installation
     End
End If
```

If the version in the networked INI file is greater than that stored within the running app, the app launches the upgrade program off the network and exits, so all files can be upgraded. This works especially well when you're in the early stages of a rollout and need to distribute multiple small incremental patches over a number of days.

—Rodney Samodral, Indianapolis, Indiana

VB4 16/32, VB5
Level: Intermediate

A WORKAROUND FOR BOUND IMAGELIST CONTROLS

When an image list is referenced by another object such as a toolbar, you can only add images to the image list. VB doesn't allow deletes or changes to the size of the images in the image list while the reference exists. Typically, the solution is to remove the references, edit the image list, and then reset the references again. However, this can become cumbersome when you might have 10 to 15 toolbar buttons referencing the image list.

To avoid the reference problem, simply select the object referencing the image list and "cut" it from your form. Then you can freely edit the image list because VB doesn't think it's referenced by any control. Once you finish editing the image list, simply "paste" the other control back on your form. All references will be restored, including the reference to the changed image list.

—Steve Dulzer, Madision Heights, Michigan

VB3, VB4 16/32, VB5
Level: Intermediate

CLOSE YOUR WINDOWS THE WINDOWS 95 WAY

Place this code in the declaration section of a module:

```
Public Sub Win95Shrivel(xForm As Form)
     ' Sets the form's window status to
     ' minimized
     xForm.WindowState = 1
End Sub
```

Call it from the Unload procedure within a form:

```
Private Sub Form_Unload(Cancel As Integer)
     Win95Shrivel Me
End Sub
```

Each time a form is unloaded, the form appears to fade to the task bar and then disappears. This tip also works on a Windows 3.1x machine as well.

—Andy McInturff, Erwin, Tennessee

VB3, VB4 16/32, VB5
Level: Intermediate

PREPARE A STRING FOR OUTPUT

If you want to show the contents of a string variable or a database field using a Label control, you should be aware that any embedded "&" character in the string is interpreted as a placeholder for the hotkey associated to the label. You can use this routine to avoid this unpleasant behavior by simply doubling each ampersand character:

```
Function AddAmpersand (InText As String)
     Dim NewText As String, i As Integer
     Do
          i = InStr(InText, "&")
          If i > 0 Then
               NewText = NewText + _
                    Left$(InText, i) + "&"
          End If
          InText = Right$(InText, _
               Len(InText) - i)
     Loop Until i = 0
     AddAmpersand = NewText + InText
End Function
```

Here's an example of its usage:

```
Label1.caption = AddAmpersand(InputText)
```

With VB4 and VB5, it's simpler to resort to the UseMnemonic property:

```
Label1.UseMnemonic = False
Label1.caption = InputText
```

—S.T. Wright, Albuquerque, New Mexico

VB4 16/32, VB5
Level: Intermediate

PROTECT YOUR WAV FILES

If you design programs that use sound files, use the Resource Compiler on the Visual Basic CD-ROM to compile your sounds into a RES file. This way, no one can take the WAV files included with your program. For more information on how to use Resource Files, consult the Resource.Txt file in the Resource subdirectory in the Tools directory on the CD-ROM.

—Joshua Stein, Wauconda, Illinois

VB3, VB4 16/32, VB5
Level: Intermediate

USING THE DIR$ FUNCTION

To quickly and easily determine the existence of a directory in Visual Basic, create this function:

```
Function IsValidPath(strPath As _
     String) As Boolean
     IsValidPath = Dir(strPath, _
          vbDirectory) <> ""
End Function
```

You can also modify the function and determine whether a file exists in VB. Simply change the "vbDirectory" constant to "vbNormal."

Learning and exploring all the Dir functions leads to many benefits:

• Navigate through all contents of a directory tree without using slow form controls or tricky APIs.
• Search for a file using wild cards (as in DOS).
• Detect all file attributes.
• Support long file names (32-bit only).

—John S. Frias, Santa Maria, California

VB3, VB4 16/32, VB5
Level: Beginning

BOOLEAN-BASED TOGGLES CAN BE MORE EFFICIENT

Many conditional statements constructed with If...Then could more efficiently use Boolean-based toggles. For example, instead of using this code for menu checking:

```
If mnuEditModeStatus.Checked = True _
     Then mnuEditModeStatus.Checked = False
Else
     mnuEditModeStatus.Checked = True
End If
```

You can easily replace it with this code:

```
mnueditModeStatus.Checked = _
     Not mnuEditModeStatus.Checked
```

You can shorten many similar constructs by using logic. In fact, the "IF condition = true THEN..." is the same as "IF condition THEN... ." Use a Boolean function or solution to replace many user variables, freeing up resources. Many times, the ability to check True/False situations exists within the scope of the VB environment, eliminating the need to generate lines of conditional statements. Try it on your If/Then and Select code—you might be pleasantly surprised at the resulting code reduction.

—Randall Arnold, Coppell, Texas