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

WELCOME TO THE 3RD EDITION OF THE VBPJ TECHNICAL TIPS SUPPLEMENT!

VBPJ's *101 Tech Tips are back again, in full force. These tips and tricks were submitted by professional developers using both Visual Basic 3.0 and Visual Basic 4.0, and compiled by the editors at* Visual Basic Programmer's Journal. *To save yourself from typing in the code, 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 74774.305@compuserve.com. Please include a clear explanation of what the technique does and why it is useful, indicate if it's for VB3, VB4, or both, and try to limit code length to 20 lines if possible. Don't forget to include your e-mail and mailing addresses; we'll pay you $25 if we publish your tip.

VB4

VB SHORTCUTS WITH WINDOWS 95

With the release of VB4 and the new development environment for 32-bit operating systems, I use three versions of Visual Basic in developing VB applications. Some of my clients won't accept apps built with VB4. Some will not soon upgrade to a 32-bit operating system. After installing both the 16-bit and 32-bit VB design environments on my Windows 95 development machine, I discovered that any project named *.VBP would be opened with the 32-bit VB design environment. This quick way to fire up the correct VB design environment on my development machine works pretty well:

- Store all files for a project in the same folder.
- Add a shortcut to the proper VB design environment to the folder or on the desktop if you wish.
- Drag the project file (*.VBP) on top of the VB design environment shortcut and (*voila!*) open the project with the proper design environment.

—Joe Sytniak

VB4

DON'T LET GO WITHOUT NOTICE

Users may unexpectedly quit your application by clicking the close menu in the control box or by clicking the close button under Windows 95. Add a procedure to the QueryUnload event on the main form of your application to prevent the problem.

If the UnLoad attempt to your application's form is not issued explicitly by your code, the UnLoadMode parameter is not equal to 1 (VB4 Constant - vbFormCode). You can cancel this attempt by setting the Cancel parameter to True. Visual Basic Help explains the details of QueryUnload event:

```
Private Sub Form_QueryUnload(Cancel _
    As Integer, UnLoadMode As Integer)

'check if the unload attempt is issued
'explicitly by code
If UnLoadMode <> 1 Then
    Cancel = True
    'prompt message box for
    'confirmation, and exit or
    'continue according to user input
End If

End Sub
```

—Jiyang Keven Luo

VB3 VB4

PROGRAMMATICALLY DIFFERENTIATE BETWEEN DESIGN MODE AND RUN TIME

This code enables and/or disables functions during design and testing. The code can remain during initial deployment without affecting the end user. Make sure the path string being searched is part of your project path and is not in your final application's directory:

```
If InStr(App.Path, "VB") Then
    ' Set Test Locations
    ' Modify Settings without changing
    ' ini or registry settings
    ' Defeat security
    ' Set options
End If

'A simple variant of this technique is

If InStr(App.Path, "VB") Then Stop
```

You can insert this during debugging, but if you forget, it won't cause users problems.

—John Bailey

VB4

NEW REGISTRY FUNCTIONS

There is a list of our new functions in Visual Basic 4.0 for working with an application's Windows registry entries (or .INI file on 16-bit Windows platforms):

- GetSetting(appname, section, key[, default])
- GetAllSettings(appname, section)
- SaveSetting(appname, section, key, setting)
- DeleteSetting(appname, section[, key])

The new functions eliminate the need to declare any Windows API calls. For more information, search in VB Help for the function name or "Additional Information on VBA Registry Functions."

—Denis Basaric and Norbert Steinhoefel-Carqueville

VB3

LOAD A VB4 FORM IN VB3

You cannot load a VB4 form in VB3 directly. You must modify the form definition. When you open a VB4 form, the file will resemble the example:

```
VERSION 4.00
Begin VB.Form Form1
     Caption  =    "Form1"
     ClientHeight =    5940
     '
     '
End
Attribute VB_Name = "Form1"
Attribute VB_Creatable = False
Attribute VB_Exposed = False
Option Explicit
```

Change the VERSION 4.0 statement to VERSION 2.0. Remove all the postfix "VB." from the form which you can see in the Begin VB.Form Form1 statement. Remove all the Attribute declarations. Save the form and load it in VB3.

—Saji Varghese

VB3 VB4

HOW TO CALCULATE THE X, Y OF ANY POSITION ON A CIRCLE

The DegreesToXYsubroutine, part of CodeBank's standard procedure library, calculates the X (horizontal) and Y (vertical) coordinates of any point, measured in degrees, on the circumference of a circle or ellipse. As you can see, it's a pretty simple routine, but extremely valuable when drawing graphics or placing objects:

```
Public Sub DegreesToXY(CenterX As _
     Long, CenterY As Long, degree _
     As Double, radiusX As Long, _
     radiusY As Long, X As Long, Y _
     As Long)
Dim convert As Double
     convert = 3.141593 / 180
     'pi divided by 180
     X = CenterX - (Sin(-degree * _
          convert) * radiusX)
     Y = CenterY - (Sin((90 + _
          (degree)) * convert) * radiusY)
End Sub
```

Pass the subroutine the center X, Y of your ellipse, the degree position, and the horizontal and vertical radii (if they are equal, you're specifying a circle, if not, it is an elongated ellipse). DegreesToXY will return the coordinates in the X and Y parameters. This routine has any number of uses, limited only by your imagination. For example, various CodeBank routines rely on it when drawing arched and rotated text, triangles, polygons, and jagged "smash" ovals when implementing motion effects, and much more.

—Ward Hitt, author of Visual Components Inc.'s CodeBank

VB4

PARSING A VARIABLE NUMBER OF ARGUMENTS

Optional parameters are a great new feature in VB4, but the ParamArray keyword is an overlooked addition to function and subroutine declarations. The ParamArray keyword allows an unspecified number of Variant parameters to be passed to a routine as an array. This subroutine uses the ParamArray keyword to parse a string into a variable number of arguments, handling text and numerics as it goes:

```
Public Sub SplitIt(ByVal S As String, _
     ByVal Delim As String, ParamArray _
     Out() As Variant)

     'Dim xIndex As Integer, xPos As
     'Integer, xNext As String

     'For xIndex = LBound(Out) To
          'UBound(Out)
          xPos = InStr(S, Delim)
          If xPos = 0 Then
               xNext = S
               S = ""
          Else
               xNext = Left$(S, xPos - 1)
               S = Right$(S, Len(S) - xPos _
                    + 1 - Len(Delim))
          End If
          Select Case VarType(Out(xIndex))
               Case vbInteger To vbCurrency
                    Out(xIndex) = Val(xNext)
               Case vbString
                    Out(xIndex) = xNext
          End Select
     Next

End Sub
```

Now you can use SplitIt to split any text string into any variables at each delimiter. For example:

```
Dim X As String, Y As Double, Z As Integer

SplitIt "A,5.5,6", ",", X, Y, Z
```

returns:

```
X="A", Y=5.5, Z=6,
```

While:

```
SplitIt "AxxxBxxxC", "xxx", , X
```

returns:

```
 X="B"
```

—Andy Brundell

VB3 VB4

CHECK FOR NULLS RETURNED FROM DLL CALLS

After a call to a DLL, the return value often contains a null. One way to eliminate the null is to look for Chr$(0), as in the example:

```
Dim CheckForNull As Integer
CheckForNull = Instr(YourString, _
     Chr$(0))
If CheckForNull > 0 then YourString = _
     Left$(YourString, CheckForNull - 1)
```

—Marc Mercuri

VB4

LICENSE ERRORS IN VB4

I ran across an interesting problem trying to install VB4 Enterprise Edition in the Windows 3.1 environment. The new version of VB uses the Registry database, which in Win 3.1 is limited to 64K.

As a computer consultant with a big-six firm, I have an extensive list of software installed on my laptop to work in various client environments.

I had previously installed Microsoft Office, MS Project, and the standard Lotus Suite software. The REG.DAT file was apparently just about maxed out.

When I installed VB4, I got no indications of errors during setup, but when I tried to run VB and use certain custom controls, I got "License" errors.

A call to Microsoft verified the problem and brought this workaround:

1. Manually remove VB4.
2. Manually remove all OCX's and OCA's from \windows\system.
3. Manually remove OC25.DLL from \windows\system.
4. Rename Reg.Dat to Reg.Old.
5. Remove all items from the Windows Startup Group.
6. Remove all programs auto started in the WIN.INI file with the "Load" or "Run"sections.
7. Remove all TSRs from the Autoexec.Bat file.
8. If you are running a compressed drive, free up 6MB of space in a non-compressed volume.
9. Exit Windows and reboot.
10. Start up Windows and re-install VB4.
11. Restore system settings.

Technical Reviewer's Note: The license errors happened to me in Windows 95. An uninstall of VB and reinstall of VB corrected the problem. Thanks for the tip!

——Jim Gilligan

VB4

RETURN VALUES NOT REQUIRED!

You do not have to use the return value of any function! This new behavior is in the manual (under CALL), but is a shocker, and a little dangerous:

```
Private Sub Form_Load()
    dice
End Sub

Function dice() As Integer
        dice = Int(Rnd * 6) + 1
        InputBox "Don't bother typing, _
            I DON'T CARE ABOUT IT!"
End Function
```

—Andy Rosa

VB4

UPDATING BOUND CONTROLS FROM A LIST OR COMBO BOX

When you want your bound controls to be updated in reaction to a user click in a list box or combobox, add this code to the click-event (or double-click event) of the list or combo control:

```
Data1.Recordset.Bookmark = _
     DBCombo1.SelectedItem
```

As a result, your current record is now the record of which the key is presented in the list part of your DBCombo/List. All bound controls are updated automatically.

You need to set only the RowSource and the ListField properties of the bound combo/list to achieve this behavior. It saves a lot of trouble converting text strings or other data-type conversions.

—Peter Klein

VB3 VB4

DOES AN OBJECT HAVE A VALUE?

You cannot use the IsEmpty function to determine if a variable of type Form or any Object has been assigned a value. You can, however, use this method to determine if the Object has ever been assigned a value other than Nothing:

```
If Not frmChild Is Nothing Then
    frmChild.Unload
End If
```

—Arn Cota

VB4

CENTER FORMS — REVISITED

I use this procedure to center my forms. With frmParent, the last loaded form is centered against the parent form. Otherwise, it's centered against the screen. I always center forms in the Load event and often forget to put Me as the parameter. To center any form, put CenterForm in the Form_Load event:

```
Public Sub CenterForm(Optional _
    frmParent)
If Forms.Count = 0 Then Exit Sub
If IsMissing(frmParent) Or Not TypeOf _
    frmParent Is Form Then
    Forms(Forms.Count - 1).Move _
        (Screen.Width - _
        Forms(Forms.Count - _
        1).Width) / 2, _
        (Screen.Height - Forms(Forms. _
        Count - 1).Height) / 2
Else
    Forms(Forms.Count - 1).Move _
        (frmParent.Width - Forms(Forms. _
        Count - 1).Width) / 2, _
        (frmParent.Height - _
        Forms(Forms.Count - _
        1).Height) / 2
End If
End Sub
```

—Denis Basaric

VB3 VB4

GET RID OF LEADING ZEROS

Eliminate "leading-zeros" in a text string in this interesting way:

```
instring$ = "00030"
' set the string with some leading-zeros
' now to get rid of them....
instring$ = CStr(Cint(instring$))
' now instring$ should contain "30"

' Another way...

instring$ = "00030"
' set the string with some leading-zeros
' now to get rid of them....
instring$ = Val(instring$)
' now instring$ should contain "30"
```

—Brad Herbert

VB3 VB4

CONVERTING IDENTIFIERS INTO LABELS AND COLUMN HEADINGS

Programmers are in the habit of creating meaningful identifiers by concatenating the words of a "title case" phrase that describes the identifier, such as LastName or FinalPaymentDate.

Often, you can use these names to create labels or column headings at run time.

SpaceName takes such an identifier and inserts spaces appropriately. Thus, X$ = SpaceName "FinalPaymentDate") returns "Final Payment Date":

```
Function SpaceName (src As String) _
    As String
    Dim i As Integer, tgt As string
    tgt = Left$(src, 1)
    For i = 2 To Len(src)
        Select Case Mid$(src, i - 1, 1)
            Case "a" To "z"
            Select Case Mid$(src, i, 1)
                    Case "A" To "Z":tgt = _
                        tgt & " "
                End Select
        End Select
        tgt = tgt & Mid$(src, i, 1)
    Next i
    SpaceName = tgt
End Function
```

—Pat Dooley

VB3 VB4

THE MID STATEMENT AND FUNCTION

You're probably familiar with the Mid function, which returns a substring of a specified number of characters from its string argument. But are you aware that Mid can also be used to replace characters in the middle of a string? The Mid statement is a bit of an oddity in VB because it alters one of its arguments, but this saves a lot of string concatenation code:

```
Dim Str1 As String
Str1 = "SOME STRING"
If Mid(Str1, 2, 1) = "O" _
    Then Mid(Str1, _
    2, 1) = "A"
MsgBox Str1
```

—William Storage

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

VB3 VB4

GET THE LENGTH OF THE LONGEST WORD IN A STRING

LenLongestWord finds the length of the longest word in a string where a word is defined to be bounded by spaces or the ends of the string.

It also illustrates a case where a recursive implementation works well in VB. For example, you would use it to decide how to spread a column heading over multiple lines:

```
Function LenLongestWord (ByVal src _
     As String) As Integer
     Dim i As Integer, j As Integer
     i = InStr(src, " ")
     If i > 0 Then
          j = LenLongestWord(Mid$(src, i _
               + 1))
          If j > i - 1 Then _
               LenLongestWord = j Else _
               LenLongestWord = i - 1
     Else
          LenLongestWord = Len(src)
     End If
End Function
```

—Pat Dooley

VB3 VB4

WHEN TO USE SENDKEYS

Use the SendKeys function to exploit the delete functionality of many grid and spreadsheet controls that have a delete functionality. Use this functionality without writing code to delete each row if your form has an option that deletes all highlighted rows. This method works much faster than writing code to check for each highlighted row in the grid, and then deleting that row.

Remember not to use SendKeys to send a key value without first setting the focus of the control to which you are sending the key value.

Call this procedure to simplify the process:

```
'Pass the Key value to send and the
'Control to which the value to be sent

Sub SendKeyTo(KeyValue As String, cCnt _
     As Control)

      cCnt.SetFocus
      SendKeys KeyValue

End Sub
```

—Saji Varghese

VB3 VB4

MONITOR RESOLUTION THROUGH API

There is a simple way to get Monitor resolutions through a WinAPI call. Declare this in a module:

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

And make a call to this API in Form_Load or Form_Resize event:

```
Sub Form_Resize()

dim xRes as integer
dim yRes as integer

xRes = GetsystemMetrics(0)
yRes = GetsystemMetrics(1)

If xRes < 1024 and yRes < 768 Then
         ' Write your control resize
         ' and reposition code here
Else
         Exit Sub
End If
```

—Sanjay Mawalkar

VB3 VB4

UNLOAD ALL FORMS

```
Public Sub UnloadAll()
Dim f As Integer
f = Forms.Count
Do While f > 0
     Unload Forms(f - 1)
     ' If form isn't unloaded exit (User had canceled...)
     If f = Forms.Count Then Exit Do
     f = f - 1
Loop
End Sub
```

—Denis Basaric

VB4

LOST LONG FILE NAMES?

Never, never, never assume that a given API call or control/OCX works the same under Win95 and Windows NT. For example, the 32-bit file common dialog handles long file names, right? Well, it does in Win95, but under NT 3.51, long file names/directory names show up as short names if that name contains an embedded space. The solution? If your program needs to run on both platforms, check for the operating system in your code. If it is Win95, use the common dialog. If it is NT, call your own form that imitates the common dialog (the DirListBox and FileListBox show long filenames with embedded spaces just fine). Or, just tell your users to wait for NT 4.0.

—L.J. Johnson

VB4

ELASTIC FONTS

Designing monitor resolution-independent applications is a frequent problem Visual Basic programmers face. The simplest solution is to design forms at the 640 by 480 resolution found in most lap-top computers. Such a form design, however, looks awkward in desktop computers which have resolutions of 1024 by 768.

Solutions to resolve this include VSElastic control from VideoSoft's VS-OCX and FarPoint's Tab Pro VBX. In VSElastic, you can paste all child controls on the Elastic control. You must adjust the control's two properties, Align = 5 (Fill Container, here the form itself) and AutosizeChildren = 7 (Proportional).

Should your forms require tabbed folders (in case of large number of controls), use Tab Pro and set the AutoSize = 5 (Fill Parent) and AutosizeChildren = 3 (size and location of control).

The controls now will automatically resize and reposition when the resolution of the monitor changes or when the form is resized. The remaining problem is that these custom controls cannot resize or reposition child controls that are pasted on frames, picture boxes, or panel controls (option buttons). As the custom controls do not alter the fontsize of the controls, captions are truncated. To overcome this problem, you need to use a true-proportional font such as Arial and insert this code in the Form_Resize event:

```
Sub Form_Resize()

dim i as integer, j as integer
dim curFormHeight as integer
dim curFormWidth as integer
dim DefaultFontSize as integer
dim orgFormHeight as integer
dim orgFormWidth as integer

On Error GoTo FormResizeError
DefaultFontSize = 8
' Or whatever fancies you
orgFormHeight = 8000
' In twips, or whatever your desired
' height is
orgFormWidth = 8000

curFormHeight = Me.Height
' Get current form height
curFormWidth = Me.Width
' Get current form width

For i = 0 to Controls.Count -1
         Controls(i).FontName = "Arial"
         Controls(i).FontSize = _
              DefaultFontSize * _
              (curFormHeight / _
              orgFormHeight)
Next i

' If the form contains option buttons or
'check box control group then
For j = 0 To Option1().Count - 1
         Option1(j).Height = 200 * _
(curFormHeight / orgFormHeight)
         Option1(j).Width = 1000 * _
(curFormWidth / orgFormWidth)
         Option1(j).Top = 250 * _
              (j + 1) * (curFormHeight _
              / orgFormHeight)
         Option1(j).Left = 250 * _
              (curFormWidth / orgFormWidth)
Next j

FormResizeError:
If Err = 438 Then
         Resume Next    ' If the form
' contains a control whose Font
' properties do not exist
End If
```

—Sanjay Mawalkar

VB4

SUBCLASSING CHDIR

If your application's current directory is D:\OldDir, the call ChDir(C:\NewDir) will change the C Drive's default directory to NewDir, but the application's current directory will remain D:\OldDir. It seemed to me that ChDir should change the application's current directory in all cases. This subclassed ChDir subroutine handles drive changes, too:

```
Sub ChDir(Path As String)
     Dim TargetDrive As String

     ' if 2nd and 3rd letters of target
     ' are ":\"

     If Mid(Path, 2, 2) = ":\" Then
          TargetDrive = Left(Path, 3)
          If TargetDrive <> _
               Left(CurDir, 3) Then
               ChDrive TargetDrive
          End If
     End If

' Call VB's ChDir function
     VBA.ChDir Path

End Sub
```

—Bruce Hamilton, Centric Development

VB3 VB4

FADING COLORS

Use this as a fast way to paint the background of any form with a really cool "fading" color (lighter at top to darker at bottom). To specify base color, pass True/False for Red, Green, and Blue. Use combinations to fade blue, red, green, yellow, purple, gray, and so on:

```
Sub FadeForm (frm As Form, Red%, _
      Green%, Blue%)
```

```
    Dim SaveScale%, SaveStyle%, _
        SaveRedraw%
    Dim i&, j&, x&, y&, pixels%

    ' Save current settings.
    SaveScale = frm.ScaleMode
    SaveStyle = frm.DrawStyle
   SaveRedraw = frm.AutoRedraw

    ' Paint screen.
    frm.ScaleMode = 3
    pixels = Screen.Height / _
        Screen.TwipsPerPixelY
    x = pixels / 64# + .5
    frm.DrawStyle = 5
    frm.AutoRedraw = True
    For j = 0 To pixels Step x
        y = 240 - 245 * j \ pixels
        'can tweak this to preference.
        If y < 0 Then y = 0
        'just in case.
        frm.Line (-2, j - 2)- _
            (Screen.Width + 2, j + _
            x + 3), RGB(-Red * y, -Green _
                * y, -Blue * y), BF
    Next j

    ' Reset to previous settings.
    frm.ScaleMode = SaveScale
    frm.DrawStyle = SaveStyle
    frm.AutoRedraw = SaveRedraw
End Sub
```

For blue fading, just put this in the Form_Load procedure:

```
FadeForm Me, False, False, True
```

—Timothy L. Birch

VB3 VB4

FILE EXISTS?

One way to test whether a specific file exists is to open the file for input and check VB's error flag. An error will occur if the file does not exist.

Use VB's 'Dir$' function to accomplish the same task. Call 'Dir$' with the complete filespec. 'Dir$' will return the exact file name if that file exists or a null string if it does not. For example:

```
    If Dir$("C:\WINDOWS\WIN.INI") _
        <> "" Then
        'file Win.ini exists!
    Else
        'file Win.ini does not exist!
    End If
```

—Chuong Van Huynh

VB3 VB4

PADDING A DATE STRING

Some functions return dates in single units (1st May 1996 may be returned as 5-1-96). This makes formatting difficult where you have dates like 5-1-96 and 12-15-96 on the same column.

The function below returns a string preceded by "0" if the length of the string is less than two. If "5" is passed to the function, it returns a "05." Passing "15" to the function returns "15":

```
Function cto2d (u As String) As String

    If Len(u) < 2 Then
        cto2d = "0" & u
    Else
        cto2d = u
    End If

End Function
```

—Segun Oyebanji

VB4

CONSTANT FAILURE? DON'T DO THAT!

Most things in VB4, including the VBA and VB engines, are OLE objects. In some cases, objects can expose properties or methods with the same name as in another object. Theoretically, the object that is highest in the references list will take priority, and the VBA and VB objects will take priority over any add-ins. But if you do run into this problem, the solution is easy. For example, if a built-in VB or VBA function doesn't seem to work and nothing else is obviously wrong, try prefixing it with VB or VBA (VBA.Left$ instead of Left$). Note that this should not happen in the case of the VBA and VB objects, but it does.

Also, it is possible to redefine the built-in VB and VBA constants to some other value, and you will get no compile error. But when you actually use the redefined constant, it will fail in some really neat ways. As the doctor said to the patient who reported a pain when he raised his arm above his head, "Then don't do that."

—L.J. Johnson

VB3 VB4

HAVE A 3-D LINE BETWEEN A PULLDOWN MENU AND A TOOLBAR

Draw an SSPanel with a height of 30. You can't set the height by hand, so you must draw it, a difficult but possible task. Delete the caption, set BevelOuter to 1-Inset, border width to 1, and align Top. Draw the Toolbar and make the pulldown menu.

—Mario Manuel Mourao Coelho

VB3 VB4

LOGGING USER ACTIVITY WITH VB AND ACCESS

A while back, I wrote a simple orders application that allowed users to query and update order information. The application worked fine, but lacked a user activity log. It was difficult for me to determine what a user had done with a particular order.

By creating a table (I'll call it UserLog) with fields such as DATE, TIME, USERID, ACTION _TYPE (A for accepted entry or Q for queried database), SQLSTRING (for the actual SQL command issued against the database), and other data element fields, I was able to capture user activity at key points in the application. The AddItem method allowed me to quickly add records to the UserLog table when the user queried the database or when the user accepted data.

Not only is the user log concept useful for tracking user activity, it's also useful for analyzing trends in data such as who is updating the data, how often, and what type of queries are being performed on the database.

—Brian Prebola

VB3 VB4

LIST BOX CUMULATIVE SEARCH

By default, VB list boxes use any keyboard input to find the first item beginning with the pressed letter. Where the number of items in a list box is large (>100), this is not helpful. Substituting a binary or hybrid search for the sequential search below speeds up the algorithm substantially for very large lists (>1000 items). The code below searches the items in a sorted list box based on all the characters typed by the user:

```
Global ListBoxSearch$

Sub ListBox_KeyPress (KeyAscii As _
    Integer)
    'Return KeyAscii in case you
    'want VB to handle certain chars
KeyAscii = SearchListBox_
    (Me.ActiveControl, KeyAscii)
End Sub

Function SearchListBox _
    (CurrentActiveControl As Control, _
    KeyAscii As Integer)
    Dim ListBoxIndex As Integer
    Dim Result As Integer
'Result of string compare

    ListBoxSearch$ = ListBoxSearch$ & _
        Chr$(KeyAscii)
    ListBoxIndex = CurrentActive_
        Control.ListIndex
 While ListBoxIndex < CurrentActive_
    Control.ListCount
    Result = StrComp(UCase$(List_
        BoxSearch$), UCase$(Left$_
        (CurrentActiveControl.List_
        (ListBoxIndex), Len(ListBox_
        Search$))))

    If Result <= 0 Then
        CurrentActiveControl.List_
            Index_= ListBoxIndex
        SearchListBox = 0
        Exit Function
    End If
    ListBoxIndex = ListBoxIndex + 1
 Wend
 ListBoxIndex = CurrentActive_
    Control.ListCount - 1
 CurrentActiveControl.ListIndex_
    = ListBoxIndex
 SearchListBox = 0
End Function
```

Note: Clearly, the code above is not complete and needs to be augmented to suit the individual developer. For example, when to clear ListBoxSearch$ is a subject of much consideration (active control changes, user presses arrow keys, and so forth) I implemented the above scheme on a list of >1500 items with a separate ClearLBSearch subroutine and a binary/sequential hybrid searching algorithm.

—Joe Wilson

VB3 VB4

MAKING USE OF SHORTCUT KEYS

"Send a Message" (Windows Programming, by Jonathan Zuck, *VBPJ* December 1995), a great example of code usable in a text editor, mentions three ways to code the Clipboard functions using SendMessage, VB's Clipboard object, or SendKeys. The Clipboard object requires hand-coding with the SelText property. SendMessage and the Clipboard object require you to specify which text box is being edited, demanding the ActiveControl property if you have more than one text box on the form. Clearly the easiest method is to use SendKeys for Undo (^Z), Delete (Del), Cut (^X), Copy (^C), and Paste (^V), the functions Windows implements natively.

In the Edit menu you'll probably want to assign Shortcut keys to these functions, so the control codes are shown to the right for users as in most Windows applications' Edit menus:

```
Undo     Ctrl+Z
**************
Delete   Del
Cut      Ctrl+X
Copy     Ctrl+C
Paste    Ctrl+V
```

The functions no longer work if your Edit menu has Shortcut keys assigned like this and the Edit subs use SendKeys to access the built-in Windows text-box editing functions. The ^X, ^C, ^V, Del, and ^Z keystrokes are captured by the Edit menu shortcuts and so are never processed. You could use SendMessage, mimic the functions in VB codes using the Clipboard object, or leave off the Shortcut keys.

All this seems unnecessary. Windows has the functions built

in and I want only to show the shortcuts on the menu so users know what they are. An easy solution is to launch the Notepad, type a Tab, then copy the Tab to the clipboard. Next, open the Menu Design Window and paste the Tab character followed by the shortcut text in the menu's Caption property:

```
Caption: &Copy[paste tabchar]Ctrl+C
```

or:

```
Caption: &Undo[paste tabchar]Ctrl+Z
```

Set the Shortcut to (none). Now put the simple SendKeys "^C" in Sub mnuEditCopy and it works!

A design-time problem with this method occurs if you leave that menu item and return to it. VB then chops off the Caption at the Tab character and you lose your shortcut text. You must repaste the Tab and retype the control code. Or, you could use a menu array and Load menu items at run time with embedded Tab characters:

```
Load mnuEditArray(1)
mnuEditArray(1).Caption = "&Undo" _
    & Chr$(9) & "Ctrl+Z"
```

I dislike doing anything in code that can be done at design time, so I prefer the former method and avoid clicking on the affected menu items.

—Jon Jensen

VB3 VB4

AN IMPROVED USER TIP

The User Tip submitted by Nick Bulka in the January 1996 issue of *VBPJ* was interesting, but, as with so much code, improvable:

```
Sub TextHiLite( t as TextBox )
        t.SelStart = 0
        t.SelLength = len(t)
End Sub
```

My improvement is to pass the TextBox control as a parameter. When VB passes a control as a parameter, it passes the Reference to it, not a copy of it. With the Reference available, you can change the properties and so forth without the time and resource overhead of declaring and instantiating a new object. Besides, it could be appropriate to change some characteristic for a control that does not have the focus. By passing it as a parameter, the programmer is not restricted to the active control.

VB3 can be lax about cleaning up after its objects. Several *VBPJ* articles suggest doing clean-up. This little routine I put in all my <form>_Unload events really helps keep the resources in check:

```
        If <object_variable> Is _
            Nothing then
        Else
                Set <object_variable> = _
                    Nothing
        End If
```

This modification is for database result-set (tables, dynasets, and so forth) objects. The code is more straightforward to create and to read with an empty 'Then' section and going right to Else:

```
        ' It's time to close the object:
        If <object_variable> Is Nothing _
            Then
        Else
                .<object_variable>.Close
                Set <object_variable> = _
                    Nothing
        End If
```

—Richard A. Stiles

VB4

CREATE MULTIPLE LEVELS OF DIRECTORIES

If your users have the option to type a directory where they want files installed, they may type a directory that doesn't exist or a directory that is several levels below another directory.

This procedure checks to see if the directory exists and creates it if it doesn't. No matter how long the request, it will create any and all directories. This procedure allows users to create long directory names in VB4, but truncates each directory to 8.3 characters in VB3. The only requirements are that sDrive be passed as "<drive>:" (i.e. "C:") and that sDir be formatted as "\<directory>\" (i.e. "\Dir1\Dir2\"). Add your own error handling to make sure these elements exist properly.

Here is an example of creating a directory several levels deep and using a long directory name:

```
Sub CreateLongDir(sDrive As String, _
    sDir As String)
    Dim sBuild As String

    While InStr(2, sDir, "\") > 1
        sBuild = sBuild & Left$(sDir, _
            InStr(2, sDir, "\") - 1)
        sDir = Mid$(sDir, InStr(2, _
            sDir, "\"))
        If Dir$(sDrive & sBuild, 16) = _
            "" Then
            MkDir sDrive & sBuild
        End If
    Wend
End Sub

Sub Test()
    Call CreateLongDir("C:", _
        "\Test\TestDir\Long Directory _
        Name\")
End Sub
```

—Jeffrey Renton

VB3 VB4

MOVE AND RESIZE CONTROLS WITH ACCURACY

Ever tried to align controls just right, or resize them to just the exact size? It can be tricky with the mouse, and modifying the properties can be time consuming. This tip works in Access 2 and 95.

When you want to resize a control:

1. Select the control.
2. Hold down the SHIFT key, and use the arrow keys to change the size.

When you want to move a control:

1. Select the control.
2. Hold down the CTRL key, and use the arrow keys to change the size.

—Chris Kunicki (forwarded by John Chmela, Visual BASIC Developer's Network)

VB4

32-BIT GETMODULEUSAGE WORKAROUND

I have found a solution to the problem of GetModuleUsage not working in 32-bit VB4. The TaskID returned by Shell can be used by AppActivate like this:

```
    TaskID = Shell("DOSAPP.EXE", _
        vbNormalFocus)
    On Error GoTo finished
    While True
        DoEvents
        AppActivate TaskID
    Wend
finished:
    On Error GoTo 0
```

—John Muir (forwarded by John Chmela, VBDN)

VB3 VB4

ACCURATE TIME DELAY

This routine allows you to put a fairly accurate time delay, very useful for I/O routines and nice, graphic-delayed drawing routines in your code:

```
Sub Delay (milliseconds As Integer)
' This Routine uses a Timer to trigger
' a fixed Time Delay.
'=====================================
Dim Temp As Integer
If (milliseconds > 0 and milliseconds _
< 32767) Then
    TimeExpired = False
    Main.Delay.Interval = milliseconds
    Main.Delay.Enabled = True
    While (TimeExpired = False)
        Temp = DoEvents()
    Wend
End If
End Sub

Sub Delay_Timer ()
'   This Routine is the Timer Event. That is, when
'   the timer expires, it is disabled and the global
'   variable TimeExpired = set to True.
    TimeExpired = True
    Delay.Enabled = False
End Sub
```

—Gary Sinde

VB3 VB4

STREAMLINE YOUR API DECLARES, PART 1

Most Windows API routines are functions and must be declared as such, but in many cases we are not really interested in their return value. The SendMessage function, for example, depending on the message sent, might not return any significant value. Even in this case, however, we are compelled to call it as a function, writing something like:

```
Dim dummy As Integer
dummy = SendMessage(Text1.hWnd, _
    WM_PASTE, 0, 0&)
```

In other words, you are forced to declare more variables and make your code less readable only because of a syntactical constraint of Visual Basic. Luckily, you can add an aliased Declare which converts SendMessage to a Sub:

```
Declare Sub SendMessageSub Lib "User" _
    Alias "SendMessage" _
    (ByVal hWnd%, ByVal msg%, ByVal _
    wParam, lParam As Any)
```

Now you can call SendMessage and discard its return value:

```
SendMessageSub Text1.hWnd, WM_PASTE, _
    0, ByVal 0&
```

Note that your code will also be slightly faster because you save an assignment and do not waste any time dimensioning a dummy variable.

—Francesco Balena

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

VB4

CREATING DATABASE PASSWORDS IN VB4

Jet 3.0 (32-bit VB4 only) includes a new security system based on database passwords that you may use instead of the more complex, more secure, workgroup security system. This system allows you to set a single, all-user password to open a database. While much simpler to implement and use, this system is very easily compromised because all users use the same password. In addition, it doesn't let you track individual user activity in a shared database. However, you can use both workgroup-based security and database passwords at the same time.

Set a database password in VB using the NewPassword method of the database object, using code like this:

```
Dim wrk As Workspace
Dim dbPwd As Database

Set wrk = DBEngine.Workspaces(0)

' You must open the database
' exclusively (note 2nd parameter).
Set dbPwd = _
wrk.OpenDatabase_
     ("MyData.MDB", True)

' Set the database password which
' currently is blank to "NewPass".
dbPwd.NewPassword "","NewPass"
```

—Paul Litwin

VB4

OPENING PASSWORD-PROTECTED DATABASES

Use the connect parameter of the OpenDatabase method to open a database that is password-protected in VB4-32. For example, this code opens the Secure database with a password of "dirt":

```
Dim wrk As Workspace
Dim dbTest As Database

Set wrk = DBEngine.Workspaces(0)

' Open Secure database with a
' password of "dirt".
Set dbTest = wrk. _
     OpenDatabase("Secure.mdb", _
     False, False, ";PWD=dirt")
```

The connect parameter is case sensitive. In addition, and contrary to the VB documentation, you must set the exclusive and read-only parameters (the second and third parameters) when using the connect parameter.

—Paul Litwin

VB3 VB4

HOW TO AUTOMATICALLY RESIZE FORMS

The ElasticForm subroutine, part of CodeBank's standard procedure library, automatically repositions and sizes all controls on an SDI form when the user or code resizes the form. To use the routine, simply lay out the form as you normally would in design mode. Then call the subroutine once in the Form_Load event, with the Init parameter set to true, so that it can record the initial positions of the controls. Call the sub in the Form_Resize event, with Init set to False, to automatically resize and reposition the controls. The procedure accommodates any number of nested containers, any ScaleMode, and all types of controls. Note, however, that the procedure relies on the Tag property to store position information for each control, and so cannot be used if Tag is being used for another purpose (see ElasticFormArray). Also, MDI child forms are resized by VB before the Form_Load or Form_Initialize events, so the proportions of each form will be distorted. With MDI children, you must use ElasticFormArray (available in CodeBank's standard library) and specify the optional DesignWidth and DesignHeight parameters:

```
Public Sub ElasticForm(frm As Form, _
    Init As Integer)
On Error Resume Next
Dim ctl As Object

If Init = True Then
    For Each ctl In frm.Controls
        ctl.Tag = Format$(ctl.Left _
            / frm.ScaleWidth, ".0000") _
            & Format$(ctl.Top / frm.Scale_
            Height, ".0000") & Format$_
            (ctl.Width / frm.ScaleWidth, _
            ".0000") & Format$(ctl._
            Height / frm.ScaleHeight, _
            ".0000")
    Next ctl
Else
    For Each ctl In frm.Controls
        ctl.Move Val(Mid$(ctl.Tag, 1, _
            5)) * frm.ScaleWidth, _
            Val(Mid$(ctl.Tag, 6, 5)) * _
            frm.ScaleHeight, Val(Mid$_
            (ctl.Tag, 11, 5)) * frm._
            ScaleWidth, Val(Mid$(ctl.Tag, 16, _
            5)) * frm.ScaleHeight
    Next ctl
End If

End Sub
```

—Ward Hitt, author of Visual Components Inc.'s CodeBank

VB4

POSITIONING A COMMON DIALOG

If you are you unhappy with Microsoft's comment in the help ("Note: You cannot specify where a common dialog box is displayed"), but like the idea of common dialog controls, try this.

Start a hidden dummy form instead of calling the open dialog box directly from your main form:

```
(frmDummy_OpenSaveAs.Hide),
```

Define the Left and Top properties as you wish and then start the common dialog box from this form. On a Windows 95 system using the 32-bit version of Visual Basic, the open dialog box appears exactly over the left/top coordinates of the form that called the dialog box. This also works if the calling form is hidden and not visible to the user.

—Reinhard Salchner

VB4

PROVIDING CONTEXT MENUS FOR YOUR UI OBJECTS

Much of the ease of use of Windows 95 comes from the fact that its user interface objects have their own context menus, which can be accessed through a simple right-click of the mouse. In keeping with this theme, you can provide context menus for the interface objects in your applications too. Making and responding to a context menu is a pretty straightforward and simple process. These steps illustrate how this is done using a standard list box called lstSample as the interface object:

1. Define the context menu. The context menu is really a standard menu item which has submenu items just like your Help menu item would. Unlike your Help menu item, however, a context menu item will have its Visible property set to False so that the user never sees it on the form's menu. For this example, open a new form and use the Menu Editor to make a new top-level menu item and give it the name mnu_lstSampleContextMenu. The caption will never be seen by the user, but should be something descriptive that reminds you what the menu is used for, such as "Context Menu For lstSample Control." Set the Visible check box for this menu item to False. Now, define the sub-menu items that will appear when the user right-clicks on the control: "&Clear," "Clear A&ll," "&Add Item," "&Remove Item," and so forth.

2. Write the code that will show the context menu when the user right-clicks the control. This is done by invoking VB's PopupMenu method in the control's _MouseDown event. Here is a code sample:

```
Private Sub lstSample_MouseDown(Button _
    As Integer, Shift As Integer, X As _
    Single, Y As Single)

    ' if the user right clicked on
    ' control then show the popup menu
    ' for this control
    If Button And vbRightButton Then _
        PopupMenu _
        mnu_lstSampleContextMenu

End Sub
```

3. All that is left to do is to write the code in the click event for each of the context menu's submenu items. The PopupMenu method can also do neat things like bold a menu item, place the menu at a specific location, and so forth. For more information on the PopupMenu method, see the VB help file.

—Hassan Davis, MicroHelp Inc.

VB3 VB4

STREAMLINE YOUR API DECLARES, PART 2

While we are speaking of SendMessage, there is another trick you may find interesting enough to include in your programming habits. When used with some particular messages, the lParam argument is really considered as two words combined. The EM_LINESCROLL can scroll a multilined text box, the low word of lParam contains the number of lines to scroll vertically (positive values scroll up), and the high word contains the number of lines to scroll horizontally (positive values scroll left); other, similar messages are EM_SETSEL for text boxes, CB_SETEDITSEL for combo boxes, and LB_SELITEMRANGE for list boxes. In such cases you need to prepare the long value to be passed to the routine, which slows down your code and makes it less readable. It seems that a simple multiplication should do the work:

```
' scroll a multilined textbox "HO"
' lines horizontally
' and "VE" lines vertically
' beware: this approach does NOT
' work properly
longValue& = HO * 65536& + VE
```

The above code does not work correctly when HO is positive and VE is negative, and for a more general scrolling routine you have to resort to a more convoluted and slower method:

```
tmp$ = Right$("000" & Hex$(HO), 4) & _
    Right$("000" & Hex$(VE), 4)
longValue = Val("&H" & tmp$)
```

The solution is declaring an aliased function that splits the lParam into two distinct parameters of Integer type, as in:

```
Declare Sub SendMessageSub2 Lib _
    "User" Alias "SendMessage" _
    (ByVal hWnd%, ByVal msg%, ByVal _
    wParam, ByVal lParam1%, _
    ByVal lParam2%)
```

Now the call is much simpler:

```
SendMessageSub2 Text1.hWnd, _
    EM_LINESCROLL, 0, HO, VE
```

The trick works because a push of a Long value onto the stack has the same effect as the push of its high word followed by the push of its low word.

—Francesco Balena

VB3 VB4

SAVE MEMORY WITH A PICTURE BOX

Set the AutoRedraw property to True and the benefits and trade-offs include much faster repaints and some wasted memory. If your form is resizable, you waste a *lot* of memory, because the persistent bitmap used by AutoRedraw is as large as the maximum dimensions of the form to reveal the hidden output when the user resizes or maximizes the window. If the graphic output you want to preserve is limited to a relatively small portion of a large and/or resizable form, you may save some precious memory drawing your graphic output in a picture box with AutoRedraw = True and possibly BorderStyle = 0, while leaving the AutoRedraw property of the form set to False.

—Francesco Balena

VB3 VB4

REMEMBER SWAP?

I was surprised to learn that the SWAP command was not implemented in Visual Basic when I read a letter in the February 1996 issue of *VBPJ* requesting that Microsoft bring to Visual Basic the Qbasic Command SWAP.

In a routine I use to sort a file, this code performs the swap. I use character strings for this example, but the logic will work with other data types.

Here is an example:

```
Option Explicit

Private Sub Form_Load()
     Dim a, b, c As String * 4
     a = "AaAa"
     b = "BbBb"
     Debug.Print a; Spc(5); b
     'Before the swap
     c = a
     a = b
     b = c
     Debug.Print a; Spc(5); b
     'After the swap.
End Sub
```

—David Ferber

VB3 VB4

A TALE OF THREE BEEPS

Are your programs not executing instructions in VB4 that were executing in VB3? Try this in Qbasic, VB3, and VB4:

```
BEEP: BEEP: BEEP
```

If you F8-step through this very complex code, you will hear three beeps except when running on VB4, which will produce only two beeps. Reserved words, such as Beep, and Cls are treated as labels. Notice the colon.

Using VB4:

```
BEEP
BEEP
BEEP
```

will give the expected three beeps.

—David Ferber

VB3 VB4

MORE ON NULL CONVERSION

This tip has been published more than once on pages of *VBPJ* (See "99 of the Hottest Tech Tips For VB Developers," Supplement to the February 1996 issue of *VBPJ*, page 17).

This code is recommended to convert Null values in the numeric database fields to zeroes:

```
Dim nVar As Integer
nVar = 0 & rs!nField
' assumed that nField
' is a numeric field in
' the recordset
```

The expression *0 & rs!nField* actually returns a string, not a number. If, say, *rs!nField* contains 1, *0 & rs!nField* results in "01."

The code above works due to the automatic conversion of types. If, however, you need to assign the numeric value, not to a numeric variable, but to a grid cell or a text box, you do not get what you want, and additional data formatting is required.

You might consider three other ways to get rid of Nulls:

```
' This will work, and in VB4 you do
' not have to include MSAFINX.DLL
' with your project
' (as you did in VB3). However, the
' expression might look a bit too long...
nVar = IIf(IsNull(rs!nField), 0, _
     rs!nField)
```

This will work, both in VB3 and VB4:

```
nVar = Val("" & rs!nField)
```

Or:

```
nVar = Val(0 & rs!nField)
```

I always use:

```
"" & n
```

instead of Str$(n) when I do not need (and I hardly ever need it!) the leading space produced by Str$() function.

—Garold Minkin

VB4

DETERMINE THE CLASS OF YOUROBJECT WITH THE TYPEOF STATEMENT

In VB4 the TypeOf statement works with any valid object type, not just forms and controls. This allows you to pass objects of different classes to the same procedure for a class-specific implementation. Here is an example:

```
'This procedure prints information
'specific to the object referenced
'in the YourObject parameter
Public Sub PrintObjectInfo(YourObject _
    As Object)

    If TypeOf YourObject Is CDesk Then
    ' tell what type of object this is
        Print "Object Type: Desk"
        ' print the number of legs that
        'this object has
        Print "Number of legs: " & Your_
            Object.NumberOfLegs
    ElseIf TypeOf YourObject Is CHouse _
        Then
    ' tell what type of object this is
        Print "Object Type: House"
        Print "Number of doors: " & Your_
            Object.NumberOfDoors
    End If
    ' both classes have the following
    ' properties
    Print "Height: " & YourObject.
        Height & " ft."
    Print "Width: " & YourObject.Width _
        & " ft."
    Print "Built by: " & YourObject._
        BuilderName
    Print "Purchase date: " & Your_
        Object.PurchaseDate
    Print "Purchase price: $" & Your_
        Object.PurchasePrice
        .
        .
        .
        .

End Sub
```

—Hassan Davis, MicroHelp Inc.

VB4

USE MULTIPLE OBJECT TYPES TO REDUCE DUPLICATE CODE

In the "Determine the Class of YourObject with the TypeOf Statement" tip, the Desk and House objects have many properties in common. Instead of duplicating code by having a separate Print method for each object, we can write a generic routine to handle printing of each object type. Note that the advantage of this implementation rises as the number of object types and/or the number of similarities between object types increases.

—Hassan Davis, MicroHelp Inc.

VB3 VB4

REMOVE THAT MOVE!

Sometimes it is desirable to prevent a user from moving a form. A little known feature of Windows is that for any form, functionality that doesn't appear in the control menu (ControlBox in VB terms) is not available to that form. Therefore, remove the Move command from that form's control menu to prevent a form from being moved:

```
Declare Function GetMenu% Lib "User" (ByVal hWnd%)
Declare Function RemoveMenu% Lib "User" (ByVal hMenu%, _
    ByVal nPosition%, ByVal wFlags%)
Const SC_MOVE = &hF010, MF_BYPOSITION = &H400
' This deletes the Move command from the form's
' control menu
Dim Res%
Res = RemoveMenu(GetMenu(Form.hWnd), SC_MOVE, MF_BYCOMMAND)
```

—Phil Parsons

VB4

JET 3.0 QUERY OPTIMIZATION, GET WITH THE PLAN

If you need to analyze the performance of a Jet 3.0 query by the query execution plan, you may do so by adding this registry key and running regedit:

```
\\HKEY_LOCAL_MACHINE\SOFTWARE\_
    Microsoft\Jet\3.0\Engines\Debug
```

For the new Debug key, add a name value of JETSHOWPLAN (all capital letters) and a data value of ON. Jet will generate a file called SHOWPLAN.OUT, which will display query execution plans associated with your application. Because these files can get to be big quickly, please remember to set the data value to OFF once you're done.

Well-designed queries and databases will generate query execution plans that refer to steps of the query execution that utilize table indices and/or Rushmore technology as much as possible. Poorly designed ones often exhibit query execution steps where table scanning (reading rows sequentially) occurs.

—Rob Karatzas

VB4

TO FLASH OR NOT TO FLASH

Often, to create status-bar help, you place code in the mouse-move event procedures of controls to place text describing the control on the status bar. Every time you update the text property, the control repaints.

Because the mouse-move event will fire many times over one control, the repainting causes flashing. To avoid the strobe light effect, check the text property of the status bar to determine if the appropriate text is already shown. Then the repainting is done only once for every control.

This subroutine is handy for use with all controls:

```
Public Sub StatusText(NewText As String)
'If status text is already correct,
'don't change it
If FrmMain.StatusBar.Panels(1).Text <> _
    NewText Then
FrmMain.StatusBar.Panels(1).Text = _
    NewText
End If
End Sub
```

To use the subroutine, add this code to the mouse-move event procedure of the controls you wish to have status-bar help:

```
Private Sub _
    CmdEnterValue_MouseMove(Button As _
        Integer, Shift As Integer, X As _
        Single, Y As Single)
    Call StatusText("Press here to _
        change the current steps value.")
End Sub
```

—Dave Robins

VB3

MAINTAINING CONSTANTS

I like to use the Constant.txt file Microsoft has been kind enough to provide. For a new project, copy Constant.txt to a new file such as MyConst.txt. Place MyConst.txt in the same directory as the MAK file and then include MyConst.txt in the project. In debug mode open MyConst.txt and do a FIND on the string Global and replace with 'Global. MyConst.txt is changed to a large comment file.

When a new constant is required, check MyConst.txt to see if Microsoft has defined it. If it is there, then create it by removing the quote. This keeps all of the Microsoft constants in context and makes the program easier to maintain.

—Stan Mlynek

VB3 VB4

CLOSE ALL FORMS BEFORE THE END OF A PROGRAM

It is well known that VB3 is not always as conscientious as it should be about unloading all the forms in an application when the application terminates. Because unloading all the forms for an application manually can be tricky, I have developed a small routine that can be called as a prelude to the End statement:

```
Sub Main ()
'
' Blah, blah, blah...our code here....
'
CloseForms
End

End Sub

Sub CloseForms()
    Dim iFormCount As Integer
    Dim i As Integer
    'Nothing's gonna stop us now....
    On Error Resume Next

    'Store the number of forms NOW
    'because the count will change as
    'we close them
    iFormCount = Forms.Count - 1

    'Step downward through the Forms
    'Collection to ensure we get
    'ALL of them...
    For i = iFormCount To 0 Step -1
        Unload Forms(i)
        Set Forms(i) = Nothing
    Next
End Sub
```

The key here is to get the count of the number of open forms and then loop from that number down to make absolutely sure that all forms in the application are unloaded.

—Joe Ackerman

VB3 VB4

OVERCOME ODBC ERRORTIP CORRECTION—SAME OL' STORY

A tip on page 48 of the March 1996 issue of *VBPJ* states that an ODBC error message occurs when using VB3 with the Access 2.0 compatibility layer. An error occurs if DB_SQLPASSTHROUGH is used to create a record set and then another record set is created without fully populating the previously opened one. The article concludes by saying that the error does not occur in VB4.

I currently use the 32-bit version of VB4 which uses Jet 3.0, and this error still occurs. I use the above method to get around it. If you create a record set that is only going to return one row you do not need to do this, and you can use DB_FORWARDONLY, which speeds up the query.

—Dave Barraclough

VB3 VB4

APPLICATION PATH INCONSISTENCY

Be aware that when using the Path property of the Application object (App.Path), if your EXE is run from the root directory of a drive, App.Path will return the drive letter and a backslash (C:\), while an EXE in a subdirectory returns only the directory name (C:\DIR). Your application should, therefore, test for the presence of the backslash before adding a file name to it:

```
MyPath=App.Path
If Not Right(MyPath,1)=Chr(92) +_
    Then MyPath=MyPath & Chr(92)
```

—Clint Walker

VB3 VB4

AVOIDING COPY AND PASTE FUNCTIONS IN A TEXT BOX

Copy (Ctrl+C) and Paste (Crtl+V) functions are already included in a text-box control, but what if you want to avoid these functions? You might suppose that you can handle Ctrl key combinations with the KeyDown event, but Crtl+C and Ctrl+V are not detected in the KeyDown event. The answer is to capture these key combinations in the KeyPress event. Type this line in the KeyPress event of the text box:

```
If keycode = 3 or keycode = 22 _
    then keycode = 0
' Where keycode = 3 is the combination
' of Ctrl+C and keycode = 22 is
' Ctrl+V.
```

—Pedro Velazquez Davila

VB3 VB4

COLOR MY WORLD—IN DEFAULT!

Just before performing your final 'make EXE,' switch your desktop color scheme and see just how many backgrounds you have hard-coded! (Sadly, several excellent shareware controls fail here, *VBPJ* contributors are guilty of this, and even the VB4 help file examples slip up!)

Try the desert desktop scheme in Win95 as a starting point, or, better still, create a new ugly scheme just to test all your control colors.

VB4 provides 24 system colors though a set of constants (search help for Color Constants or VBTranslateColor) for use in your code, many of which are used as a default for VB controls, but if you've changed the property setting with the VB palette, you must retype (or paste) the hexadecimal number shown next to the color into the property text box, substituting the 0x for &H. Alternatively, you could click on another control that already has the correct color value, copy the property value, and paste it into your control. Then you won't be so gray.

Technical Reviewer's Note: In VB4, there is a new Color Palette View Window with a DEFAULT button. Select each Form or Control one at a time, select the DEFAULT button and the control will be set to the Windows Default. Be aware that the Appearance property should be set to 3-D for best results.

—Clint Walker

VB4

PRINTING PROBLEMS

Here is a tip to help with missing or mispositioned text. This failing code works fine with VB3:

```
Cls
Print Spc(10); "Enter Your Name";
currentx = 0
currenty = currenty + 1
Print Spc(10); "Enter Your Name";
```

The messages will be far to the right in VB4. Check the last semicolon in the first Print line to correct this. To show this, precede each Print statement with "debug":

```
Debug.Print Spc(10); "Enter Your Name"

CurrentX = 0
CurrentY = CurrentY + 1
Debug.Print Spc(10); "Enter Your Name";
```

or change the statement to:

```
Print Space$(10); "Enter Your Name";
```

—David Ferber

VB4

USE CODE PROFILER FOR DEBUGGING

Sometimes a runtime error only manifests itself after an EXE is made and not in debug mode. This can be hard to track down, even with message-box statements. Use the code profiler add-in to find the offending line of code. Following these steps should lead to your error:

1. First back up your code.
2. Select the code profiler add-in.
3. Select the source code file (or all source code files).
4. Select the Line Hit Count option.
5. Select the Add Profiler Code button.
6. Compile the code into an EXE.
7. Run the code to the error.
8. Go back to the code profiler and select View Results from the File menu.

Look for the last line that was executed in the offending module. You may have to trace through your code in debug mode at the same time that you look for the last line executed in the profiler.

—Rich Spencer

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

VB4

LISTVIEW COLUMN SORTING

Give your ListView control in report view the sort functionality of the Win95 Explorer. This code will allow you to sort by any column. If your list is already sorted by the column header you press, the sort order is toggled:

```
Private Sub ListView1_ColumnClick_
    (ByVal ColumnHeader As ColumnHeader)
    With ListView1
        'If current sort column header is
        'pressed then
        If (ColumnHeader.Index - 1) = _
            .SortKey Then
            'Set sort order opposite of
            ' current direction.
            .SortOrder = (.SortOrder + _
                1) Mod 2
        Else
            'Otherwise sort by this column
            '(ascending)
            .Sorted = False
            'Turn off sorting so that the
            ' list is not sorted twice
            .SortOrder = 0
            .SortKey = _
                ColumnHeader.Index - 1
            .Sorted = True
        End If
    End With
End Sub
```

—Joe Tuttle

VB3 VB4

WHERE'S THE BEEP?!

This code will prevent a beep when you hit [ENTER] or [TAB] in a text box whose maximum character limit has been reached:

```
Sub Form_KeyPress (keyascii As Integer)
'Eliminate beep if {ENTER} or {TAB}
If keyascii = 13 Or keyascii = 9 Then _
    keyascii = 0
End Sub
```

—Lonnie Broadnax, Michael Ottomanelli, & Preston Werntz

VB3

HOW TO DESELECT ALL ITEMS IN A LIST BOX

A quick way to deselect everything in a multiselect listbox is:

```
    list1.Selected(-1) = False
```

This doesn't work in VB4.

—John Muller

VB3 VB4

AUTOMATIC TAB TO THE NEXT FIELD WITH MAXIMUM CHARACTER LIMIT IN A TEXT BOX

This need was established by developing VB applications for 3270 mainframe apps. A user typing in a 3270 edit field is automatically placed into the next field once reaching the maximum number of characters for the field:

```
Code:
Sub Text1_KeyUp (keycode As Integer, _
    Shift As Integer)
If keycode > 47 And keycode < 123 _
    Then
'Only advance if they have just typed
'a number or character.
If Len(Me.ActiveControl.Text) = _
    (Me.ActiveControl.MaxLength) Then
SendKeys "{TAB}", True
End If
Endif
End Sub
```

—Lonnie Broadnax, Michael Ottomanelli, & Preston Werntz

VB3 VB4

SIMPLIFYING THE CONDITION-PART OF AN IF STATEMENT

When you write an If statement such as:

```
If Category = "CM" or Category = "M2" or Category = "P1" or
    Category = "ZZ" then
ProcessEmployee
Endif
```

it can be simplified by:

```
dim ValidValues as string
ValidValues = "CM M2 P1 ZZ"
' don't forget to insert any
' categories between P1 and ZZ
if (instr(1, ValidValues, Category)) > 0 then
ProcessEmployee
endif
```

Not only does this version not require you to go on scrolling horizontally while writing the code but is faster as well, based on simple tests I conducted using VB3 on a 486DX-66. Note that I have used a space to separate categories in ValidValues string. You may use any separator, such as semicolon, comma, etc. If you do not use a separator, the ValidValues string will become "CMM2P1ZZ" and you might get incorrect results and make the ValidValues string less readable.

—Jaspreet Singh

VB3 VB4

ELIMINATING THE IF STATEMENT WHEREVER POSSIBLE

If you have to assign True or False to a variable after testing a certain condition, then the If statement may be done away with altogether.

For example, if you have something like this:

```
If (age < 13 and sex = "M") or (age > _
    14 and sex = "F") then FetchRecord _
    = true
```

it can be replaced by:

```
FetchRecord = (age < 13 and sex = "M") _
    or (age > 14 and sex = "F")
```

This performs the same function as the If statement above. The FetchRecord variable is correctly assigned True or False. The advantage is that it allows the splitting condition-part of the If statement (see example below). Further, if you like replacing your If statements by IIf, then you know that IIf requires you to add msainfx.dll to your projects in VB3. The above version doesn't.

If the condition-part of if statement is very lengthy, then the above technique allows you to split it. Thus, if you have something like this in VB3:

```
If (Age > 25 and Category = "M1") or _
    (Age > 35 and Category <> "C1") or _
    (Age > 45 and Category = "M7") or _
    (Age > 45 and Category = "P1") then
ProcessRecord
Endif
```

you can do this:

```
Dim Result as integer
Result = (Age > 25 and Category = "M") _
    or (Age > 35 and Category <> "C1")
Result = Result or (Age > 45 and _
    Category = "M7") or (Age > 45 and _
    Category = "P1")
if Result then ProcessRecord
```

The advantage is that it allows limited splitting of lines in VB3, thus making the code more readable.

By the way, the technique:

```
x = (y op z)
```

to assign True or False to x works in C, Pascal and FoxPro as well. Remember to use = for assignment in Pascal and == for testing in C.

—Jaspreet Singh

VB3 VB4

SIZABLE FORMS WITHOUT A TITLE BAR

If you set a form properties caption to "" and control box to False, then a form with borderstyle 3 (fixed dialog) will display without its title. Unlike borderstyle zero (none), it keeps the form's 3-D properties. If you do this with a borderstyle 5 (sizable toolwindow) form, then you have a completely resizable (down to 1x1 twip) form.

This can also be done from your code in run time. Adding or removing the caption will add or remove the title bar. Be aware that this triggers a form resize event, and that the task bar caption will also be blank. Finally, remember to put a close button on your form!

—Clint Walker

VB3 VB4

BLANKS IN THE MASKED EDIT CONTROL

The Microsoft Masked Edit control only allows input that matches the mask. Similarly, you can only set the Text property to a string that matches the mask. If you have a mask for a phone number that only allows numbers (#), you cannot set the text to blanks. An easy way to set the text to blank is with this code:

```
vTemp = mskPhone.Mask
mskPhone.Mask = ""
mskPhone.Text = ""
mskPhone.Mask = vTemp
```

Removing the mask allows you to set the text to any appropriate string. Restore the mask afterwards. I use this code in the validation event of a data control when adding a new record.

—Scott Wallace

VB3 VB4

EDITING THE REGULAR GRID

The regular grid that comes with Visual Basic cannot be edited directly. However, this problem can be easily circumvented with one line of code:

```
Private Sub Grid1_KeyPress(KeyAscii As _
    Integer)
Grid1.Text = Grid1.Text & Chr(KeyAscii)
End Sub
```

All the user has to do is select the cell to edit.

—Lev Muginshteyn

VB3 VB4

ENFORCE UPPERCASE CHARACTERS

Do you want to facilitate the user to enter only uppercase characters during editing in text boxes and other edit controls? Irre-

spective of the position of CAPS LOCK key, this code in the KeyPress event of the Form will force the user always to enter uppercase characters.

```
Private Sub Form_KeyPress(KeyAscii _
    As Integer)
KeyAscii = Asc(UCase(Chr(KeyAscii)))
End Sub
```

For this code to work, you need to set the KeyPreview property of the Form to True.

—Balamurali Balaji

VB3 VB4

ADD 'TYPE AHEAD' FUNCTIONALITY TO COMBO BOXES

You can add 'Type Ahead' functionality to the standard VB ComboBox control with this subroutine:

```
Sub ComboTypeAhead(Combo As ComboBox)
Dim i As Integer, _
    KeyString As String, _
    KeyLen As Integer

KeyString =3D Combo.Text
KeyLen =3D Len(KeyString)
gblTypeAhead =3D False
If KeyLen > 0 Then
For i =3D 0 To Combo.ListCount - 1
If Left(Combo.List(i), KeyLen) =3D _
    KeyString Then
Combo.SetFocus
Combo.ListIndex =3D i
Combo.SelStart =3D KeyLen
Combo.SelLength =3D Len(Combo.Text) _
    - KeyLen
Exit For
End If
Next i
End If
End Sub
```

Call this routine from the Change event of the ComboBox with the name of the ComboBox as the only parameter. Additionally, you may want to add some code to the KeyPress event to handle the Delete and Backspace keys (that is where the gblTypeAhead variable is used), but other than that, this routine will handle all of the type-ahead functionality.

—Jon Rauschenberger

VB4

UNDOCUMENTED FEATURE LOGS JET QUERIES

Jet 3.0 includes an undocumented feature that lets you log Jet's optimization plan of queries. To enable this feature, you must create this registry key using Windows' RegEdit program:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\ _
    Jet\3.0\Engines\Debug
```

Create a string registry key under this node called "JetShowPlan" and set it equal to "On." Now all queries executed from VB4-32 and Access 95 will be logged to a file named SHOWPLAN.OUT in the current directory. To turn logging off, set the key equal to "Off." Note that this file may get large, so you may wish to delete it every so often. This feature is, of course, completely unsupported and may cause unwanted side effects, so use it with caution.

—Paul Litwin

VB3 VB4

SECURING A JET DATABASE THE RIGHT WAY

To secure a Jet database, you must purchase Access 2 for VB3/VB3-16 or Access 95 for VB4-32 and follow these steps:

1. Use the Access Workgroup Administrator to create a new workgroup with a non-null Workgroup ID.
2. Start Access and set a password for the default Admin account.
3. Create a new user account, adding it to the Admins group so it will have administer privileges. Remove the Admin account from the Admins group.
4. Restart Access, logging on as the new user, and set a password.
5. Run the Access Security Wizard. (For Access 2, download a copy from http://www.microsoft.com/accdev.)
6. Create the user and group accounts for the workgroup.
7. Set permissions on objects for the group accounts. Don't assign any permissions to the built-in Admin user or Users group accounts.

Don't skip any of these steps!

—Paul Litwin

VB3 VB4

INTEGRALHEIGHT PROPERTY OF LIST AND COMBO BOXES

In VB3, if you wanted to display a list box adjacent to another control such as a PictureBox on a sizable form, you were confronted with the problem that a list box's height was constrained to an integral number of list items. By using a series of Windows and VB API calls (requiring a special DLL, such as Desaware's SpyWorks), you could destroy an existing list-box control, reset its style bits to include LBS_NOINTEGRALHEIGHT, and re-create the control, which could then be resized to any height.

The IntegralHeight property is now included in the standard and data-bound list controls, so you'll get the expected behavior with automatic resizing tools such as VideoSoft's Elastic. The DBList and DBCombo also include a VisibleCount property that returns the number of visible items in the list.

—William Storage

VB4

REGEDIT ADDITION "AUTOMATION"

This sample file from Microsoft allows a more automated Windows 95 Registry Addition. You can create a similar file in Notepad or your favorite text editor, replacing the appropriate entries in place of the MS Internet Explorer entries listed. Make sure REGEDIT4 is the first line in the file:

```
REGEDIT4

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsof_
    t\Internet Explorer\Main]
"Default_Page_URL"="http://www.msn.com"
"Default_Search_URL"="http://www.msn._
    com/access/allinone.htm"

[HKEY_CURRENT_USER\Software\Microsof_
    t\Internet Explorer\Main]
"Start Page"="http://www.msn.com"
"Search _
    Page"="http://www.msn._
    com/access/allinone.htm"
```

Now run the file you create or double-click on the icon. If the REGEDIT association is set properly, RegEdit will run and place the appropriate information in the registry.

—Douglas Haynes

VB3 VB4

SET FORMS TO NOTHING WITH CAUTION

It's a good idea to set your form variables to Nothing to recover all memory that was allocated for the form module. Executing Set Form1 = Nothing for a loaded form, however, will have no apparent effect, but will leave the form module in a confused state. You can demonstrate this by executing:

```
Form2.Show
Set Form2 = Nothing
Form2.Show
Msgbox Forms.Count & " loaded forms"
Unload Form2
Unload Form2
```

The second line of this code seems to do nothing, but the second use of Form2's Show method will in fact load and display a second instance of Form2. The forms collection will know about both instances, but only one of them will be unloaded with the Unload statement.

To avoid these problems, always be sure a form is unloaded before setting it to Nothing. While you cannot execute Set Me = Nothing, you can achieve the same effect in a form's Unload event:

```
Form_Unload (Cancel As Integer)
    Dim Form As Form
    For Each Form In Forms
        If Form Is Me Then
            Set Form = Nothing
            Exit For
        End If
    Next Form
End Sub
```

—William Storage

VB3 VB4

AVOID REDUNDANT EXECUTION OF CODE

Most collections are zero-based. The COUNT property returns the number of members of a collection, as opposed to the highest member number. So when we see code that loops through all the members of a collection, it usually looks something like this:

```
    Dim I%

    For I% = 0 To Controls.Count - 1
        Controls(I%).Enabled = True
    Next
```

If your application does not mind stepping through the collection in reverse order, this code can be more efficient by eliminating the need to read the collection's COUNT property and deducting 1 for each iteration of the loop:

```
    Dim I%

    I% = Controls.Count
    While I%
        I% = I% - 1
        Controls(I%).Enabled = True
    Wend
```

I recommend the next technique for similar situations to avoid redundant execution of code. For example, to count the number of spaces in a string, we could write:

```
    Dim I%, C%

    C% = 0
    For I% = 1 To Len(SomeString$)
        If Asc(Mid$(SomeString$, I%)) _
            = 32 Then C% = C% + 1
    Next
```

which calls the Len() function for every iteration of the loop. Or, you could do it this way:

```
    Dim I%, C%

    C% = 0
    I% = Len(SomeString$)
    While I%
        If Asc(Mid$(SomeString$, I%)) = _
```

```
            32 Then C% = C% + 1
        I% = I% - 1
    Wend
```

—Robert C. Ashcraft

VB3 VB4

CHANGE THE MOUSEPOINTER TO PREVENT USER INPUT

Because setting the MousePointer property for a form only changes the shape and does not prevent the application from accepting user input from the mouse or keyboard, I have developed this tip.

The tip works on an MDI form where there may be several open child windows, and you need to disable one or all of the windows during long processing periods (e.g., long database updates). If you have only one MDI child open or need to disable only one form, you can disable it by setting the Enabled property to False. If you have several child forms open, create a collection of forms and scroll through the collection, disabling each form again by setting the Enabled property to False. When the selected forms are disabled, set the MousePointer property for the MDI parent to an hourglass. This will prevent user input from the mouse or the keyboard until you reverse the process by enabling the forms and changing the mouse back to whatever shape it was before.

The real trick involves setting the MDI parent's mouse icon to the hourglass shape. If you just set the MousePointer for the child form(s) to an hourglass and then disable the child form(s), the pointer will turn back into the default shape.

—Al Gehrig, Jr.

VB3

STUCK WITH PASTE

When you use some custom controls like QuickPak Professional by Crescent it's impossible to use the shortcut "Ctrl+V" for "Paste." "Ctrl+V" allows you to paste two times the text of the clipboard. You can eliminate the shortcut from the menu list but if you want to show in the menu the string "Ctrl+V" you can use this code in the "main_load" procedure:

```
menu_cut.Caption = "&Cut" + _
     Chr$(9) + "Ctrl+X"
menu_copy.Caption = "C&opy" + _
     Chr$(9) + "Ctrl+C"
menu_paste.Caption = "&Paste" _
     + Chr$(9) + "Ctrl+V"
```

—Daniele Alberti

VB3

STANDARD MATH PROBLEM

A program originally written using the Borland C++ Compiler has two problems in Visual Basic 3.0, Standard Edition. This program uses trigonometry to calculate the distance between two points, when you know their latitude and longitude, such as N 90 W 123 45 56 S 90 W 0.

The C++ formula, using the Inverse Cosine Function acos() double distance; double radians = 3.141592654 / 180; double intermediate_result; distance=(69.094*acos(intermediate_result)) / radians.

There is a resolution difference between C++ and Visual Basic results.

Problem 1: Inconsistent results: Assume intermediate_result = 0.999995422198391and printing to 10 decimal places. Distance: Using C++ 11.9876152369 miles, using VB 11.9931567014 miles. Inverse Cosine input is -1 to +1, its result is 0 to 180 degrees. In C++, acos(angle).

Visual Basic 3.0, Standard Edition, does not directly support Inverse Cosine. But VB help shows "Derived Math Functions."

Microsoft's formula is: Arcos(x)=Atn(-x/Sqr(-x*x+1))+1.5708. The difference in resolution is that 1.5708 is too large. When I subtract 3.67320510333E-06 from 1.5708, my VB results match my C++ results. Now, I get the exact same answers with VB as with C++.

Problem 2: Hang on -1. When a value of -1 is presented to this "Derived Math Function," Windows Hangs in this Arcos function. Fix for Visual Basic: if intermediate_result = 1 then distance = 0, if intermediate_result = -1 then distance = 180 * 69.094

—David Ferber

VB4

SIMULTANEOUS DEBUG FOR SERVER AND CALLING APPLICATION

VB4 not only allows you to create OLE Automation servers, but to debug the server and the calling application simultaneously. If you want to create a remote OLE server, set the Instancing property of the class modules to Creatable SingleUse. This will make debugging more interesting.

Each time you call that class, the calling application will try to create another instance (process) of that server. The server is running in design mode, and VB will not start another copy of itself and load the server app again. The solution is to temporarily set the instancing of the class modules to Creatable MultiUse for testing purposes. Don't forget to set the instancing back to SingleUse before compiling the OLE server.

—L.J. Johnson

VB3 VB4

LEAP YEAR RULES

One of the trickier parts of handling date values is dealing with leap years. Everyone knows that every fourth year is a leap year in which February has 29 days instead of the typical 28. What is less known is that there are two more rules to check to determine whether a given year is a leap year or not:

- Years that are evenly divisible by 100 are *not* leap years;
- Years that are evenly divisible by 400 *are* leap years.

With that in mind, I wrote this function to test whether any given year is a leap year:

```
Function IsLeap (iYear As Integer) _
     As Integer

     'Set Default Value
     IsLeap = False

     'Check the 400 Year rule
     If (iYear Mod 400 = 0) Then
          IsLeap = True
          GoTo IsLeap_Exit
     End If

     'Check the 100 Year rule
     If (iYear Mod 100 = 0) Then
     IsLeap = False
     GoTo IsLeap_Exit
     End If

     'Check the 4 Year rule
     If (iYear Mod 4 = 0) Then
     IsLeap = True
     Else
     IsLeap = False
     End If

IsLeap_Exit:

End Function
```

(Note: This is the VB3 version. For VB4 substitute a Boolean return value for the function.)

—Joseph H. Ackerman

VB4

IDENTIFY NETWORKED CD DRIVES

The 32-bit API is much richer than the 16-bit API. However, the GetDriveType still reports networked CD drives as just a DRIVE_REMOTE (i.e., a networked drive). While true, it is not particularly helpful. Combine the GetDriveType call with the GetVolumeInformation call to determine that it is both a network drive and a CD drive.

The next-to-last parameter of this call returns a string that gives the type of file system on that volume: FAT, NTFS, HPFS, CDFS (CD File System):

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

     pstrRootPath = "E:\"
     pstrVolName = Space$(256)
     pstrSystemType = Space$(32)
     plngSysTypeSize = CLng(Len(pstrSystemType))
     plngVolNameSize = CLng(Len(pstrVolName))
     plngRtn = GetVolumeInformation_
          (pstrRootPath, pstrVolName, _
          plngVolNameSize, plngVolSerialNum, _
          plngMaxFilenameLen, plngSysFlags, _
          pstrSystemType, plngSysTypeSize)
```

—L.J. Johnson

VB3 VB4

DATABOUND GRID BUG SOLUTION

A severe bug in VB4 appears when using the Databound Grid with modal forms. Create 3 forms: form1, form2 and form3. Put a button command1 on each of the forms. In the click event of command1 in form1, show form2 modally. In the click event of command1 in form2, show form3 modally. Drop a DBGRID on form3. In the click event of command1 in form3, unload form3.

Run form1. Press each button as the forms show up. When pressing the third button, I get a stack overflow error in both 16-bit and 32-bit VB4. Also, on Windows 3.1, the system hangs up.

Solve the problem by avoiding modal forms when using bound controls. If you need modal behavior in a form, all you have to do is keep track of the form that opened it, and set its Enabled property to False. You can create a property procedure to keep a reference to the caller form. Then you could show the "modal" form like this:

```
With FormModal
     .Prop_Caller = Me
     .Show
End With
```

Now set Caller.enabled to False in the "modal" form Load event, and to True in the Unload event.

—Luis Miguel da Costa Pereira Ferreira

VB4

USING THE UNDOCUMENTED COUNT PROPERTY OF CONTROL ARRAYS

In VB4, each control array is its own collection and therefore has a Count property. This was not the case in version 3. It is now possible to loop through all of the elements of a control array without having to hard-code the maximum value of the array index.

This feature is not documented in the VB4 manuals or online help. The likely reason for this is that a control array collection does not support all of the standard methods and properties of a "real" collection. The Count property and the Item method are supported, but the Add and Remove methods are not.

This simple example uses the Count property to determine which element of an array of option buttons has been selected:

```
Private Sub FindSelectedOption ( )

    Dim ii As Integer

    For ii = 0 To Option1.Count - 1
        If Option1(ii).Value Then
        MsgBox "Option Button " & ii _
            & " is selected."
        End If
    Next ii

End Sub
```

This procedure will only work if the array indices are continuous, with no gaps in the sequence. If, for example, the control array consists of elements 0, 1, 3, 4, the above procedure will generate a runtime error 340 when it tries to evaluate Option1(2), and even if this first error is trapped, Option1(4) will never be reached in the loop, and therefore will not be evaluated.

—Craig Everett

VB3 VB4

HEXADECIMAL COLOR CONSTANTS

Properly designed Windows programs should change their own colors according to the system colors. One of the reasons why developers tend to forget that part of design is that it is not easy to find values for those colors. Microsoft forgot to mention them in the manuals and they cannot be found in help files either.

There are several ways to determine constants for the system colors. One is by the Object Browser. Unfortunately, it could be time consuming. From now on, you can use this list:

System Color Name	Color
Menu Text	&H80000007&
Scroll Bars	&H80000000&
Window Background	&H80000005&
Window Frame	&H80000006&
Window Text	&H80000008&
Active Border	&H8000000A&
Active Title Bar	&H80000002&
Active Title Bar Text	&H80000009&
Application Workspace	&H8000000C&
Button Face	&H8000000F&
Button Highlight	&H80000014&
Button Shadow	&H80000010&
Button Text	&H80000012&
Desktop	&H80000001&
Disabled Text	&H80000011&
Highlight	&H8000000D&
Highlighted Text	&H8000000E&
Inactive Border	&H8000000B&
Inactive Title Bar	&H80000003&
Inactive Title Bar Text	&H80000013&
Menu Bar	&H80000004&

—Frank Coliviras and Dejan Sunderic

VB4

DETERMINING IF AN OBJECT HAS BEEN SET

VB4 provides lots of new capabilities to use objects. Unfortunately, all of them require the object to be set beforehand, which isn't always feasible. VB provides no way to see if an object has been set. The only way of checking is to pass the object to a function and attempt to access it. If it hasn't been set, an error (number 91) will occur.

For example:

```
Public Function IsSomething(o As _
    Object) As Long
Dim j As Long

    Err.Clear
    On Error Resume Next

    If TypeOf o Is TextBox Then
        j = 1
        'just a mindless test
        'to see if we get an error
    End If

    If Err.Number = 91 Then
    'error 91 = object not set
        IsSomething = False
        ElseIf Err.Number = 0 Then
            IsSomething = True
        Else
            Err.Raise Err.Number
            'if some other error happened, raise it
    End If

    On Error GoTo 0
End Function
```

—Evan Dickinson

VB3

CREATING INNER JOINS IN AN ACCESS DATABASE

This seemingly undocumented SQL table qualification syntax might be useful to access an external table by using the 'IN' keyword. I had used this ability before, but kept having trouble with joins (probably due to my syntax) until I used the database path as a qualifier for the table. In this example I separated the four tables found in the Biblio database into their own databases and issued this query in an empty database:

```
SELECT Authors.*
FROM C:\VB3\Biblio1.Authors _
     INNER JOIN _
     (C:\VB3\Biblio2.MDB.Titles
INNER JOIN _
     (C:\VB3\Biblio3.MDB.Publishers
INNER JOIN _
     C:\VB3\Biblio4.MDB.[Publisher _
     Comments]
ON Publishers.PubID = [Publisher _
     Comments].PubID)
ON Titles.PubID = Publishers.PubID)
ON Authors.Au_ID = Titles.Au_ID;
```

While this may not work on ODBC, it works perfectly with Access databases and overcomes the single external database ability with the 'IN' clause.

—Mark P. Atwood

VB4

SEQUENTIAL NAVIGATION OF A TREEVIEW CONTROL

While the TreeView control with its associated Node objects is a far cry better than the VB3 Outline control, it lacks a method for sequentially traversing the tree, as if you were scrolling using the up and down arrow keys. The For Each construct allows you to process every node, but in order of Index, which can be meaningless if the tree is sorted by the display text. This code, to be called after the TreeView has been loaded and sorted, creates a 1-based array of keys in the order that the Nodes appear in a TreeView:

```
Global asKey() as String

Sub Main()
    '...Fill the TreeView1 control
    ReDim asKey(1 To _
         TreeView1.Nodes.Count) As String
    KeyArray TreeView1.Nodes(1).Root.FirstSibling, 1
End Sub

Private Sub KeyArray(n as Node, _
    Optional iStart)
    Static i As Integer

    If IsMissing(iStart) Then
         i = i + 1
    Else
         i = iStart
    End If

    asKey(i) = n.Key

    If n.Children > 0 Then KeyArray _
         n.Child

    Do While n <> n.LastSibling
         i = i + 1
         Set n = n.Next
         asKey(i) = n.Key
         If n.Children > 0 Then _
              KeyArray n.Child
    Loop
End Sub
```

The first sibling of the root of an arbitrary node is used to return the first entry in the TreeView. The use of the Optional parameter allows the routine to be called whenever the TreeView is reloaded or modified. Once the array is established, it allows for sequential navigation or searching, or whatever operation would benefit from knowing the order of all Nodes.

—Greg Frizzle

VB4

USE YOUR OWN POPUP MENUS IN VB4

In VB4 if you want to show a popup menu for a text box, a system-provided context menu appears first, and your popup menu does not appear until you close the system context menu.

Here is a workaround:

```
Private Sub Text1_MouseDown(Button _
    As Integer, Shift As Integer, X _
    As Single, Y As Single)

If Button = 2 Then
    Text1.Enabled = False
    PopupMenu mnuFile
    Text1.Enabled = True
End If

End Sub
```

—Mario Coelho

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

VB4

WRITING ADD-INS CAN BE TRICKY

Writing add-ins for Visual Basic 4.0 can be challenging, rewarding, and tricky. If you are not careful writing add-ins, you can cause VB to do strange things and even abort. Although I'm sure that other diagnostics can occur, this error has appeared more than once for me while I was debugging add-ins. The messages may vary, depending on the operating system, but the result is the same.

For example, under Windows 95, you might see "This program has caused an error and will be terminated…" or "If the problem persists, contact the vendor.…" Under Windows 3.1, it may result in a GPF.

These occur when the IDE is being unloaded and will be followed on a subsequent reload of VB with this: "xxxxxx add-in could not be loaded, do you want to remove it from the list of Add-ins?"

After this, you will have to re-execute the add-in to reregister it for attachment to VB. I have found two causes for these errors:

1. Referencing a property of the VBIDE Instance Object, such as ActiveProject.Filename in the ConnectAddin Event of the Connector Class.
2. Connecting more menu or sub-menu items than you disconnect.

Most programming is an exact science, and strict adherence to "mostly undocumented" rules is an absolute necessity when writing add-ins for VB4.

—Les Smith

VB4

AVOID UPDATE ERRORS IN ACCESS DATABASES

Avoid "Couldn't update; currently locked by user 'name' on machine 'machine' " error in VB4 when two or more applications access the same MS Access database table.

For the given table, a primary or any nonunique key is defined. The two applications may have opened the database table as any one of the record-set types such as dynaset or table.

If one of the applications is idle (just opened the record set and no edit/updation taking place) and the other application tries to update or add a record to the table, then the Error Code: 3260 ("Couldn't update; currently locked by user 'name' on machine 'machine' ") is displayed.

To avoid this, after the record set is opened (in case of a table record set, if index is being set, then after the Index property is set), include this statement:

```
DBEngine.Idle (dbFreeLocks)
```

VB4 releases the table locks and now the other application will be able to update the database table. For example:

```
Dim DB as Database, TB as Recordset
Set Db = WorkSpaces(0).OpenDatabase_
    ("Test.mdb")
Set TB = Db.OpenRecordset("Customer_
    Master",dbOpenTable)
Tb.Index = "PrimaryKey"

DBEngine.Idle (dbFreelocks)
```

—Rajesh Patil

VB4

RETRIEVING DATA FROM A DIALOG FORM WITHOUT USING GLOBALS

From the days of VB3, you no doubt recall the pain of retrieving answers from a dialog form that you displayed from another form. The use of Globals was about the only way provided. In VB4, forms are objects and you can call methods in them from another form just as if they were in a code module. This new feature allows you to call a function in a dialog form, which will display itself, and when the form is unloaded, return your answers. Add this code to your main code that needs user input:

```
Dim sAnswer As String

sAnswer = frmDialog.Display()

'when control returns sAnswer
'will have the user reply.
```

To your frmDialog (.frm) file, add this code:

```
Dim sRetValue As String

Public Function Display() As String
    Me.Show vbModal
    Display = sRetValue
End Function

Private Sub cmdOK_Click()
    ' This function must set up the
    ' return value because the
    ' Text box will be unloaded when
    '"Display" regains control
     sRetValue = Text1.Text
    Unload Me
End Sub
```

Obviously, you can retrieve data from more than one control. In that instance you would pass an object or array to the Display function for it to use to return multiple values. User-defined types (UDTs) don't appear to work.

—Les Smith

VB4

SPEED YOUR ANIMATIONS AND SAVE MEMORY

Tired of loading icons or bitmaps or creating many images with the several styles of pictures in form modules that blow up your application and slow it down in VB3?

If so, you can create several icons or bitmaps of similar style. Then create a pure text (ASCII) file like printer.txt or printer.rc this way:

```
NameID_keyword_[load-option]_
    [mem-option]_filename
```

First, define each of the terms in this code. Where nameID equals Integer or name, the ID has to be unique for every category specified by the keyword:

```
keyword = ICON,BITMAP
```

If the code "[load-option]" is equal to PRELOAD, then the resource loads immediately. If it is equal to LOADONCALL, the (Default) resource loads when called.

If the code "[mem-option]" is equal to FIXED, the resource remains at a fixed memory location. If it is equal to MOVABLE, then the (Default) Resource can be moved if necessary in order to compact memory. If it is equal to DISCARDABLE, then the resource can be discarded if no longer needed.

And "filename" specifies the name of the file that contains the resource. The name must be a valid MS-DOS file name, and it must be a full path if the file is not in the current working directory. The path can be either quoted or nonquoted string.

The text file looks like this:

```
1 ICON LOADONCALL DISCARDABLE _
    C:\ICONS\...\PRNTMAN0.ICO
9 ICON LOADONCALL DISCARDABLE _
    C:\ICONS\...\PRNTMAN8.ICO
```

Open the DOS-WINDOW in Win 3.1 or return to DOS and run the rc.exe that shipped with the German version of VB4 Pro (also on the American/English version CD-ROM).

It may look like this:

```
C:\VB\RESOURCE\RC16\rc -r printer.rc
```

or for the 32-bit RES file:

```
C:\VB\RESOURCE\RC32\rc -r printer.rc
```

Next, push enter and the resource file will soon be generated. Search for the file named "printer.res" or the name you have chosen.

Create a new project. Add a picture box and a timer to your form. Then add the RES file to your project.

The timer-event looks like this:

```
Private Sub Timer1_Timer()
Static ID
    ID = ID+1
    IF ID = 10 Then
        ID = 1
    End IF

    Picture.Picture = _
        LoadResPicture(ID,1)
End Sub
```

Don't forget to set the timer interval.

—Uwe Pryka

VB4

UNLOADING OUT-OF-CONTROL DLLS

When I work with VB under Windows 95, I'll often do something with a program that causes Windows 95 to be unstable. Normally, I would shut down Windows 95 and restart to clear out all loaded VBXs and DLLs, but I recently found an easier way.

Create a DOS batch file called RESTART.BAT with these contents on your hard drive:

```
EXIT
```

Within Windows 95, create a shortcut to this batch file. Under the properties, make sure that you select MS-DOS mode under 'Program / Advanced'. This method is much faster than rebooting the machine.

—Michael J. Dyer

VB3 VB4

MOVING ITEMS IN A LIST BOX

To change the item's location by dragging the mouse in a list box, follow this tip:

```
Sub List1_MouseDown (Button As _
    Integer, Shift As Integer, X As _
    Single, Y As Single)

    Old_Index = List1.ListIndex
    TmpText = List1.Text

End Sub
```

While the button mouse is on, this event saves the current index and text to variables:

```
Sub List1_MouseUp (Button As Integer,_
    Shift As Integer, X As Single, _
    Y As Single)

    New_Index = List1.ListIndex

    If Old_Index <> New_Index Then
        List1.RemoveItem Old_Index
        List1.AddItem TmpText, New_Index
    End If
```

```
End Sub
```

When the button mouse is off, a new variable sets the new mouse location. To test if it is the same, remove the old item and add the new item with the text saved in TmpText.

General Declarations:

```
Dim TmpText As String
Dim Old_Index As Integer
Dim New_Index As Integer
```

—Scotti, Marcio Cristaiano de Castro

VB3

ACTIVATING A PREVIOUS INSTANCE OF AN APPLICATION

To prevent users from launching multiple instances of your application, check the PrevInstance property of VB's App object; if the property is True, the program is already running. To activate the previous instance, call Windows' FindWindow, ShowWindow, and SetFocus APIs:

```
Global Const SW_RESTORE=9
Declare Function FindWindow Lib "User" _
     (ByVal lpClassName As Any, ByVal _
     lbWindowName As Any) As Integer
Declare Function ShowWindow Lib "User" _
     (ByVal hWnd As Integer, ByVal _
     nCmdShow As Integer) As Integer
Declare Function SetFocusAPI Lib _
     "User" Alias "SetFocus" (ByVal _
     hWnd As Integer) As Integer

Sub Main ()

     Dim hWnd As Integer
     Dim iResult As Integer

     ' App object's Title property may be
     ' set at runtime, as illustrated
     ' here, or at compile time in VB's
     ' Make .EXE' dialog.
     App.Title = "Test Application"

     If App.PrevInstance Then

          ' ThunderForm is the class name
          ' for VB forms (Thunder was the
          ' original code name for Visual
          ' Basic within 'Microsoft)

          ' Find the existing instance
          hWnd = FindWindow("Thunder_
               Form", App.Title)

          ' FindWindow function returns a
          ' non-zero value if it finds a
          matching window
          If hWnd <> 0 Then
               ' Restore window if minimized
               iResult = ShowWindow(hWnd,
                    SW_RESTORE)
' Set focus to the specified window
               iResult = SetFocusAPI(hWnd)
               ' Close this instance
               End
          End If
     End If

     ' If no previous instance, show
     ' main form and proceed normally.
     frmMain.Show

End Sub
```

—Senthil Shanmugham

VB4

INCORRECT API LISTINGS

APILOD16.EXE and APILOD32.EXE access the file WIN32API.TXT to allow the programmer to paste the TYPE declarations needed to call the Win32 API.

The file WIN32API.TXT incorrectly allots a LONG field to each bit field the API requires. A misalignment in parameters results, rendering those TYPEs somewhere between dangerous and useless. This could be very difficult to debug.

WIN32API.TXT (incorrectly) shows:

```
Type COMSTAT
     fCtsHold As Long       ' wrong!
     fDsrHold As Long       ' wrong!
     fRlsdHold As Long      ' wrong!
     fXoffHold As Long      ' wrong!
     fXoffSent As Long      ' wrong!
     fEof As Long           ' wrong!
     fTxim As Long          ' wrong!
     fReserved As Long      ' wrong!
     cbInQue As Long
     cbOutQue As Long
End Type
```

The WIN31API.TXT correctly says:

```
Type COMSTAT
     bunch_Of_Bits As Long
     cbInQue As Long
     cbOutQue As Long
End Type
```

—Andy Rosa

VB4

DATA ENTRY BECOMES USER FRIENDLY: MOVE TO THE NEXT CELL IN DBGRID STD CONTROL

When I replaced the Apex dbGrid, with the free upgrade, TDBGridS1.OCX, I got complaints about the behavior of the grids when I entered or edited data in TDBGrid Standard. The cell pointer stays put and highlights the text just entered. The user must press the enter key a second time or use the arrow key to move to the next cell.

Three lines of code automatically move the cursor to the next cell after entering or editing a cell. The grid is in the Bound mode.

1. Create a form level variable:

```
Dim KC as integer
```

2. In the TDBGridS1_Keydown Event:

```
Private Sub TDBGridS1_KeyDown(KeyCode _
    As Integer, Shift As Integer)
    KC = KeyCode ' Trap the keycode
End Sub
```

3. In the TDBGridS1_AfterColEdit Event:

```
Private Sub _
    TDBGridS1_AfterColEdit(ByVal _
        ColIndex As Integer)
    ' statments go here
    ' on the last line use this code:
    If KC = 13 Then SendKeys "{Enter}"

End Sub
```

—Philip Speck

VB4

HIGHLIGHTING A ROW IN A BOUND DBGRID

To highlight the row containing the active cell in a bound DBGrid, add the record set's current bookmark to the grid's SelBookmarks collection:

```
Private Sub DBGrid_RowColChange _
    (LastRow As Variant, ByVal LastCol As Integer)

    If datCtl.Recordset.RecordCount Then
        DBGrid.SelBookmarks.Add _
            datCtl.Recordset.Bookmark
    End If

End Sub
```

—Peter Chyan

VB3 VB4

IDENTIFYING A GENERIC CONTROL AT RUN TIME

When a procedure can operate on multiple types of controls, you can use VB's TypeOf function to determine a control type at run time:

```
Function myFunc(ctl as Control)

    ' This code works in both VB3 & VB4
    If TypeOf ctl Is TextBox Then
        ' Code for text boxes here…
    ElseIf TypeOf ctl Is CommandButton _
        Then
        ' Code for command buttons here…
    End if

End Function
```

VB4 adds the new TypeName function, which allows you to test the control's type once, then branch based on the result:

```
Function myFunc(ctl As Control)

    Dim sCtlType as String

    ' TypeName is new to VB4
    sCtlType = TypeName(ctl)
    Select Case sCtlType
        Case "TextBox"
            ' Code for text boxes here…
        Case "CommandButton"
            ' Code for command buttons
                'here…
    End Select

End Function
```

To learn the type (or class) name of a given control, highlight it at design time and look at VB's Properties window. The type name appears to the right of the control's name in the combo box at the top of the window.

—Senthil Shanmugham

VB3 VB4

BE NICE

If you can't think of anything nice to say about somebody, get to know them better.

—John Chmela