For the Week of May 3rd

Keep Your Background Processes Running

In Visual Basic, if you make a call to the MSGBOX function all other background processes that you may have running (counters, timer events, etc) are stopped until the user acknowledges the Msgbox dialog box. This can be potentially devastating if you write an application that runs unattended.

To overcome this problem, you must use the Windows API call for the MessageBox function. It looks and acts the same as the VB "msgbox" function, but does not stop the background processes from running.

In a module, paste the following API declaration:

```
Declare Function MessageBox Lib "user32" Alias "MessageBoxA" (ByVal hwnd As
Long, ByVal lpText As String, ByVal lpCaption As String, ByVal wType As Long) As
Long
```

Next, on the default form add a timer control, 2 command buttons, and a label. Then type the following code into the form, which demonstrates the VB msgbox and API MessageBox functions. That's all there is to it.

```
Private Sub Command1_Click()
MsgBox "The Timer STOPS!"
End Sub
Private Sub Command2_Click()
MessageBox Me.hwnd, "Notice the timer does not stop!", "API Call", _ vbOKOnly +
vbExclamation
End Sub
Private Sub Timer1_Timer()
Label1.Caption = Time
End Sub
```

For a Week of August 2nd

A new Format function

VB 5 has the Format command that almost works the same as Print. The difference is that Format shortens the output string length if all the format characters are not used. To work around this I wrote a Public Function called FormatNum.

Public Function FormatNum(MyNumber As Double, FormatStr As String) As String

```
   ' This Function returns number formatted as a string
   '    with the desired minimum number of characters
   ' MyNumber - Use CDbl(MyNumber) in the function
   '    call to prevent type mismatch error.
   '
   FormatNum = Format$(MyNumber, FormatStr)
   If Len(FormatNum) < Len(FormatStr) Then
      FormatNum = Space$(Len(FormatStr) - Len(FormatNum)) & FormatNum
   End If
End Function
```

Use this function like this:

```
Print #FileNumber, FormatNum(CDbl(MyVariable), " #### ")
```

Febuary 2, 1998

Showing long ListBox entries as a ToolTip

By Matt Vandenbush, matt_vandenbush@whbrady.com

Sometimes the data you want to display in a list is too long for the size of ListBox you can use. When this happens, you can use some simple code to display the ListBox entries as ToolTips when the mouse passes over the ListBox.

First, start a new VB project and add a ListBox to the default form. Then declare the SendMessage API call and the constant (LB_ITEMFROMPOINT) needed for the operation:

```
Option Explicit
```

```
'Declare the API function call.
Private Declare Function SendMessage _
Lib "user32" Alias "SendMessageA" _
(ByVal hwnd As Long, _
ByVal wMsg As Long, _
ByVal wParam As Long, _
lParam As Any) As Long
' Add API constant
Private Const LB_ITEMFROMPOINT = &H1A9
```

Next, add some code to the form load event to fill the ListBox with data:

```
Private Sub Form_Load()
'
' load some items in the list box
With List1
.AddItem "Michael Clifford Amundsen"
.AddItem "Walter P.K. Smithworthy, III"
.AddItem "Alicia May Sue McPherson-Pennington"
End With
'
End Sub
```

Finally, in the MouseMove event of the ListBox, put the following code:

```
Private Sub List1_MouseMove(Button As Integer, Shift As Integer, _
X As Single, Y As Single)
'
' present related tip message
'
Dim lXPoint As Long
Dim lYPoint As Long
Dim lIndex As Long
'
If Button = 0 Then ' if no button was pressed
lXPoint = CLng(X / Screen.TwipsPerPixelX)
lYPoint = CLng(Y / Screen.TwipsPerPixelY)
'
With List1
' get selected item from list
lIndex = SendMessage(.hwnd, _
LB_ITEMFROMPOINT, _
0, _
ByVal ((lYPoint * 65536) + lXPoint))
' show tip or clear last one
If (lIndex >= 0) And (lIndex <= .ListCount) Then
.ToolTipText = .List(lIndex)
Else
.ToolTipText = ""
End If
End With '(List1)
End If '(button=0)
'
End Sub
```

Febuary 9, 1998

Simple file checking from anywhere

By Matthew Kent, mace@pacificcoast.net

To keep my applications running smoothly, I often need to check that certain files exist. So, I've written a simple routine to make sure they do. Here it is:

```
Public Sub VerifyFile(FileName As String)
'
On Error Resume Next
'Open a specified existing file
Open FileName For Input As #1
```

```
'Error handler generates error message with file and exits the routine
If Err Then
MsgBox ("The file " & FileName & " cannot be found.")
Exit Sub
End If
Close #1
'
End Sub
```

Now add a button to your form and place the code below behind the "Click" event.

```
Private Sub cmdVerify_Click()
'
Call VerifyFile("MyFile.txt")
'
End Sub
```

November 16, 1998

Figuring out the current screen resolution

You can use the following small piece of code to detect the current screen resolution and then act on the information - for instance, by resizing form objects to suit the user's resolution.

```
Dim x,y As Integer
    x = Screen.Width / 15
    y = Screen.Height / 15
    If x = 640 And y = 480 Then MsgBox ("640 * 480")
    If x = 800 And y = 600 Then MsgBox ("800 * 600")
    If x = 1024 And y = 768 Then MsgBox ("1024 * 768")
```

June 29, 1998

Creating a incrementing number box

submitted by Bryan Shoemaker
www.shadow.net/~fubar

You can't increment a vertical scroll bar's value -- a fact that can become annoying. For example, start a new project and place a text box and a vertical scroll bar on the form. Place the vertical scroll bar to the right of the text box and assign their Height and Top properties the same values. Assign the vertical scroll bar a Min property value of 1 and a Max value of 10. Place the following code in the vertical scroll bar's Change event:

```
Text1.Text = VScroll1.Value
```

Now press [F5] to run the project. Notice that if you click on the bottom arrow of the vertical scroll bar, the value increases; if you click on the top arrow, the value decreases. From my perspective, it should be the other way around.

To correct this, change the values of the Max and Min properties to negative values. For example, end the program and return to the design environment. Change the vertical scroll bar's Max value to -1 and its Min value to -10. In its Change event, replace the line you entered earlier with the following:

```
Text1.Text = Abs(Vscroll1.Value)
```

Now press [F5] to run the project. When you click on the top arrow of the vertical scroll bar, the value now increases. Adjust the Height properties of the text box and the scroll bar so you can't see the position indicator, and your number box is ready to go.

March 23, 1998

Creating a new context menu in editable controls

By Antonio Almeida, future.systems@mail.telepac.pt

This routine will permit you to replace the original context menu with your private context menu in an editable control.

Add the following code to your form or to a BAS module:

```
Private Const WM_RBUTTONDOWN = &H204 Private Declare Function SendMessage Lib
"user32" Alias "SendMessageA" (ByVal hwnd As Long, ByVal wMsg As Long, ByVal
wParam As Long, lParam As Any) As Long
Public Sub OpenContextMenu(FormName As Form, MenuName As Menu)

'Tell system we did a right-click on the mdi
Call SendMessage(FormName.hwnd, WM_RBUTTONDOWN, 0, 0&)
'Show my context menu
FormName.PopupMenu MenuName
'
End Sub
```

Next, use the Visual Basic Menu Editor and the table below to create a simple menu.

Caption	**Name**	**Visible**
Context Menu	mnuContext	NO
...First Item	mnuContext1	
...Second Item	mnuContext2	

Note that the last two items in the menu are indented (...) one level and that only the first item in the list ("Context Menu") has the Visible property set to NO.

Now add a text box to your form and enter the code below in the MouseDown event of the text box.

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

If Button = vbRightButton Then
Call OpenContextMenu(Me, Me.mnuContext)
End If

End Sub
```

Note: If you just want to kill the system context menu, just comment out the line:

```
FormName.PopupMenu MenuName
```

in the OpenContextMenu routine.

March 30, 1998

Dragging items from a list to another one

By Bassam Alkharashi, bkhrashi@kacst.edu.sa

Here's a way that you can let users drag items from one list and drop them in another one. Create two lists (lstDraggedItems, lstDroppedItems) and a text box (txtItem) in a form (frmTip).

Put the following code in the load event of your form.

```
Private Sub Form_Load()
' Set the visible property of txtItem to false
txtItem.Visible = False
'Add items to list1 (lstDraggedItems)
lstDraggedItems.AddItem "Apple"
lstDraggedItems.AddItem "Orange"
lstDraggedItems.AddItem "Grape"
lstDraggedItems.AddItem "Banana"
lstDraggedItems.AddItem "Lemon"
'
End Sub
```

In the mouseDown event of the list lstDraggedItems put the following code:

```
Private Sub lstDraggedItems_MouseDown(Button As Integer, Shift As Integer, X As
Single, Y As Single)
'
txtItem.Text = lstDraggedItems.Text
txtItem.Top = Y + lstDraggedItems.Top
txtItem.Left = X + lstDraggedItems.Left
txtItem.Drag
'
End Sub
```

In the dragDrop event of the list lstDroppedItems put the following code:

```
Private Sub lstDroppedItems_DragDrop(Source As Control, X As Single, Y As
Single)
'
If lstDraggedItems.ItemData(lstDraggedItems.ListIndex) = 9 Then
Exit Sub
End If
' To make sure that this item will not be selected again
lstDraggedItems.ItemData(lstDraggedItems.ListIndex) = 9
lstDroppedItems.AddItem txtItem.Text
'
End Sub
```

Now you can drag items from lstDraggedItems and drop them in LstDroppedItems.
Note that you cannot drag from the second list to the first. Also, the dragged item remains in the first list. You'll have to address those limitations yourself.

Top of Page

April 06, 1998

Add Dithered Backgrounds to your VB Forms

By: Barron Anderson, Micron Electronics, Inc.

Ever wonder how the SETUP.EXE screen gets its cool shaded background coloring? This color shading is called dithering, and you can easily incorporate it into your forms. Add the following routine to a form:

```
Sub Dither(vForm As Form)
Dim intLoop As Integer
vForm.DrawStyle = vbInsideSolid
vForm.DrawMode = vbCopyPen
vForm.ScaleMode = vbPixels
vForm.DrawWidth = 2
vForm.ScaleHeight = 256
For intLoop = 0 To 255
vForm.Line (0, intLoop)-(Screen.Width, intLoop - 1), RGB(0, 0, 255 -intLoop), B
Next intLoop
End Sub
```

Now, add to the Form_Activate event the line

```
Dither ME
```

This version creates a fading blue background by adjusting the blue value in the RGB function. (RGB stands for Red-Green-Blue.) You can create a fading red background by changing the RGB call to

```
RGB(255 - intLoop, 0, 0).
```

May 25, 1998

Confirm Screen Resolution

Submitted by Nicholas L. Otley, nicholaso@kalamzoo.co.uk; www.kalamazoo.co.uk

Here's a great way to stop the user from running your application in the wrong screen resolution. First, create a function called CheckRez:

```
Public Function CheckRez(pixelWidth As Long, pixelHeight As Long) As Boolean
'
Dim lngTwipsX As Long
Dim lngTwipsY As Long
'
' convert pixels to twips
lngTwipsX = pixelWidth * 15
lngTwipsY = pixelHeight * 15
'
' check against current settings
If lngTwipsX <> Screen.Width Then
CheckRez = False
Else
If lngTwipsY <> Screen.Height Then
CheckRez = False
Else
```

```
CheckRez = True
End If
End If
'
End Function
```
Next, run the following code at the start of the program:
```
If CheckRez(640, 480) = False Then
MsgBox "Incorrect screen size!"
Else
MsgBox "Screen Resolution Matches!"
End If
```

June 22, 1998
Measuring a text extent
submitted by Nenad Cus Babic
Nenad@computer.org
It's very simple to determine the extent of a string in VB. You can do so with WinAPI functions, but there's an easier way: Use the AutoSize property of a Label component. First, insert a label on a form (labMeasure) and set its AutoSize property to True and Visible property to False. Then write this simple routine:
```
Private Function TextExtent(txt as String) as Integer
labMeasure.Caption = txt
TextExtent = labMeasure.Width
End Function
```
When you want to find out the extent of some text, simply call this function with the string as a parameter.
In my case it turned out that the measure was too short. I just added some blanks to the string. For example:
```
Private Function TextExtent(txt As String) As Integer
labMeasure.Caption = " " & txt
TextExtent = labMeasure.Width
End Function
```

August 24, 1998
Help with Shell
submitted by Brad Gile
bgile@amfam.com
Suppose you have a DOS program, Dosapp.exe. This program produces an output file that will subsequently be processed, and you want to do this N times. The code might look like this:
```
    for Trial = 1 to N
    x=shell("Dosapp.exe",vbHide)
    Process the output
    Next Trial
```
The problem is, the code after the Shell may be executed before the Shelled Dosapp.exe has created the file. There are complicated ways of solving this, including API calls, but here's a simple solution: the FileLen function. If the file to be processed is x$, you can just insert a few lines of code:
```
    for Trial = 1 to N
    Open x$ for output as 1
    Close 1
    ' This sets file length equal to zero
    x=shell("Dosapp.exe",vbHide)
    Do While FileLen(x$) = 0
    DoEvents
    ' Halt further execution until x$ is created and closed
    Loop

    ' Process the output to add to a new file
    Next Trial
```

You may want to do other things such as using Timer to prevent an infinite Do Loop, but this is the main idea. FileLen() works because even if x$ is open and has data in it, FileLen(x$) = 0. Thus you're assured that the process code won't execute until x$ is fully created.

GETting the contents of a file
submitted by Pritesh
TransCapacity LP; spritesh@hotmail.com
To read a complete file in VB, the normal procedure is to read the contents of the file line by line and accumulate it into a string. Instead, you can use the GET function to read the file with a single call. Doing so simplifies and speeds up the process of reading a file.
You can use the following function:

```
    Dim Handle As Integer
    Dim FileString As String
    Handle = FreeFile
    Open "C:\TEMP\TST.TXT" For Binary As #Handle
    FileString = Space(FileLen("C:\TEMP\TST.TXT"))
    Get #Handle, ,FileString
    Close #Handle
```

This code involves a single call to return the contents of the file.

Modernize Your Toolbar Look

Using only a few Windows API calls, you can change the standard VB5 toolbar into an Office 97 look-alike. I've implemented two display styles for the toolbar. The first allows you to change the toolbar to an Office 97-style toolbar (similar to the one used by VB5), and the second allows you to change the toolbar to the Internet Explorer 4.0-style toolbar. If you want to use the second style, you must supply each button with some text in order to achieve the effect. In both cases, the button edges are flat and only appear raised when the mouse passes over the button. To implement it, add this code to a BAS module:

```
Private Declare Function SendMessage Lib "user32" Alias _
        "SendMessageA" (ByVal hwnd As Long, ByVal wMsg As Long, _
        ByVal wParam As Integer, ByVal lParam As Any) As Long
Private Declare Function FindWindowEx Lib "user32" Alias _
        "FindWindowExA" (ByVal hWnd1 As Long, ByVal hWnd2 _
        As Long, ByVal lpsz1 As String, ByVal lpsz2 As _
        String) As Long

Private Const WM_USER = &H400
Private Const TB_SETSTYLE = WM_USER + 56
Private Const TB_GETSTYLE = WM_USER + 57
Private Const TBSTYLE_FLAT = &H800
Private Const TBSTYLE_LIST = &H1000

Public Sub Office97Toolbar(tlb As Toolbar, _
        tlbToolbarStyle As Long)
        Dim lngStyle As Long
        Dim lngResult As Long
        Dim lngHWND As Long

        ' Find child window and get style bits
        lngHWND = FindWindowEx(tlb.hwnd, 0&, _
                "ToolbarWindow32", vbNullString)
        lngStyle = SendMessage(lngHWND, _
                TB_GETSTYLE, 0&, 0&)

        ' Use a case statement to get the effect
        Select Case tlbToolbarStyle
        Case 1:
                ' Creates an Office 97 like toolbar
```

```
                lngStyle = lngStyle Or TBSTYLE_FLAT
        Case 2:
                ' Creates an Explorer 4.0 like toolbar,
                ' with text to the right
                ' of the picture. You must provide text
                ' in order to get the effect.
                lngStyle = lngStyle Or TBSTYLE_FLAT _
                        Or TBSTYLE_LIST
        Case Else
                lngStyle = lngStyle Or TBSTYLE_FLAT
        End Select

        ' Use the API call to change the toolbar
        lngResult = SendMessage(lngHWND, _
                TB_SETSTYLE, 0, lngStyle)

        ' Show the effects
        tlb.Refresh
End Sub
```

Call this routine while a form with a toolbar is loading:

```
Private Sub Form_Load()
        Call Office97Toolbar(Me.Toolbar1, 2)
        ' whatever...
End Sub
```

Michiel Leij
The Netherlands,

Loop on Non-Numeric Indices

You might occasionally need to execute a group of statements with different and unrelated values of a variable. For example, say you need to verify that a number isn't a multiple of 2, 3, 5, 7, or 11. In these circumstances, you can't use a regular For...Next loop, unless you store these values into a temporary array. Here's a more concise solution:

```
Dim n As Variant
For Each n In Array(2, 3, 5, 7, 11)
        If (TestNumber Mod n) = 0 Then
                Print "Not prime"
                Exit For
        End If
Next
```

You can use the same technique to iterate on non-numeric values:

```
' check if a string embeds a shortened weekday name
Dim d As Variant
For Each d In Array("Sun", "Mon", "Tue", "Wed", "Thu", _
        "Fri", "Sat")
        If Instr(1, TestString, d, vbTextCompare) Then
                Print "Weekday = " & d
                Exit For
        End If
Next
```

Francesco Balena
Bari, Italy

Keypress Won't Fire When Pasting Into Text Box

Don't put rules for validating text values or formats in the KeyPress event-use the Change event instead. If you "paste" into a text box, the KeyPress event isn't fired and all your

validation goes out the window. Also, if you don't carefully put code in the Change event that sets the value of a text box, you'll create an infinite loop:

```
Private Sub Text1_Change()
        'Append asterisk to text
        Text1.Text = Text1.Text & "*"
End Sub
```

Here's a better way:

```
Private Sub Text2_Change()
        Dim lCurr As Long
        'Append asterisk to text
        lCurr = Text2.SelStart
        If Right$(Text2.Text, 1) <> "*" Then
                Text2.Text = Text2.Text & "*"
                'Be kind and don't put the cursor at the front of the
                'text
                Text2.SelStart = lCurr
        End If
End Sub
```

Joe Karbowski
Traverse City, Michigan

In Search of Sample Code

I'm always looking for sample code, and the setup1.vbp file is an excellent source of reusable code. It comes with VB and is part of the VB setup kit. The contents vary, depending on what version of VB you have, but you'll find useful examples in each version. For example, the VB5 file sample code does these things:

- Gets the Windows directory.
- Gets the Windows System directory.
- Determines if a file or directory exists.
- Determines if you're running WinNT or Win95.
- Determines drive type.
- Checks disk space.
- Creates a new path.
- Reads from an INI file.
- Parses date and time.
- Retrieves the short path name of a file containing long file names.

Plus, a whole module works to log errors to an error file. This code is well-commented and can easily be cut and pasted into your project. **Carole McCluskey**
Seattle, Washington

Improve on the Bubble Sort

A bubble sort's execution time is a multiple of the square of the number of elements. Because of this, the bubble sort is said to be an n-squared algorithm. You can easily make improvements to a bubble sort to speed it up.

One way is to reverse the direction of passes reading the array, instead of always reading the array in the same direction. This makes out-of-place elements travel quickly to their correct position. This version of a bubble sort is called the shaker sort, because it imparts a shaking motion to the array:

```
Public Sub Shaker(Item() As Variant)
        Dim Exchange As Boolean
        Dim Temp As Variant
        Dim x As Integer

        Do
                Exchange = False
                For x = (UBound(Item)) To (LBound(Item) + 1) Step -1
                        If Item(x - 1) > Item(x) Then
```

```
                                Temp = Item(x - 1)
                                Item(x - 1) = Item(x)
                                Item(x) = Temp
                                Exchange = True
                        End If
                Next x

                For x = (LBound(Item) + 1) To (UBound(Item))
                        If Item(x - 1) > Item(x) Then
                                Temp = Item(x - 1)
                                Item(x - 1) = Item(x)
                                Item(x) = Temp
                                Exchange = True
                        End If
                Next x
        Loop While Exchange
End Sub
```

Although the shaker sort improves the bubble sort, it still executes as an n-squared algorithm. However, because most programmers can code a bubble sort with their eyes closed, this is a nice way to shave 25 to 33 percent off the required execution time without having to dig out the algorithm books. Still, you don't want to use either a bubble or shaker sort for extremely large data sets. **Tan Shing Ho**
Kuala Lumpur, West Malaysia

Implement a Binary Tree

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

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

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

```
        End If
        'Display this node
        MsgBox NodeData
        'Traverse right branch
        If Not RightBranch Is Nothing Then
                RightBranch.TraverseTree
        End If
End Sub
```

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

David Doknjas
Surrey, British Columbia, Canada

Hunt for Developers

Want to see a list of the developers who worked on VB5 and VB6? Try this: From VB's View menu, select Toolbars, then Customize.... In the resulting dialog, click on the Commands tab. In the Categories list, select Help. Select "About Microsoft Visual Basic" in the Commands list, and drag it to any menu or toolbar. Right-click on the item you just dragged and rename it to "Show VB Credits" (without the quotes). Then close the "Customize" dialog and click on the "Show VB Credits" item. **Phil Weber**
Tigard, Oregon

Grab System Fonts Easily

At times, you might want to retrieve the current system font settings, such as the font being used for window title bars, or the menu or message box font. You could delve into the Registry, but why go to the trouble if the SystemParametersInfo API does it for you? Here's how:

```
Private Declare Function SystemParametersInfo Lib "user32" _
        Alias "SystemParametersInfoA" (ByVal uAction As Long, _
        ByVal uParam As Long, lpvParam As Any, ByVal fuWinIni _
        As Long) As Long

        Private Type LOGFONT
                lfHeight As Long
                lfWidth As Long
                lfEscapement As Long
                lfOrientation As Long
                lfWeight As Long
                lfItalic As Byte
                lfUnderline As Byte
                lfStrikeOut As Byte
                lfCharSet As Byte
                lfOutPrecision As Byte
                lfClipPrecision As Byte
                lfQuality As Byte
                lfPitchAndFamily As Byte
                lfFaceName As String * 32
        End Type

        Private Type NONCLIENTMETRICS
                cbSize As Long
                iBorderWidth As Long
                iScrollWidth As Long
                iScrollHeight As Long
                iCaptionWidth As Long
                iCaptionHeight As Long
                lfCaptionFont As LOGFONT
                iSMCaptionWidth As Long
                iSMCaptionHeight As Long
```

```
        lfSMCaptionFont As LOGFONT
        iMenuWidth As Long
        iMenuHeight As Long
        lfMenuFont As LOGFONT
        lfStatusFont As LOGFONT
lfMessageFont As LOGFONT
End Type

Private Const SPI_GETNONCLIENTMETRICS = 41

Public Function GetCaptionFont() As String
        Dim NCM As NONCLIENTMETRICS
        NCM.cbSize = Len(NCM)
        Call SystemParametersInfo(SPI_GETNONCLIENTMETRICS, _
                0, NCM, 0)
        If InStr(NCM.lfCaptionFont.lfFaceName, Chr$(0)) _
                > 0 Then
                GetCaptionFont = _
                        Left$(NCM.lfCaptionFont.lfFaceName, _
                        InStr(NCM.lfCaptionFont.lfFaceName, Chr$(0)) _
                        - 1)
        Else
                GetCaptionFont = NCM.lfCaptionFont.lfFaceName
        End If
End Function
```

Keep in mind this function-GetCaptionFont-returns only the name of the font. However, all the other font information is there in the LOGFONT structures as well. **Ben Baird**
Twin Falls, Idaho

Generate Random Strings

This code helps test SQL functions or other string-manipulation routines so you can generate random strings. You can generate random-length strings with random characters and set ASCII bounds, both upper and lower:

```
Public Function RandomString(iLowerBoundAscii As _
        Integer, iUpperBoundAscii As Integer, _
        lLowerBoundLength As Long, _
        lUpperBoundLength As Long) As String

        Dim sHoldString As String
        Dim lLength As Long
        Dim lCount As Long

        'Verify boundaries
        If iLowerBoundAscii < 0 Then iLowerBoundAscii = 0
        If iLowerBoundAscii > 255 Then iLowerBoundAscii = 255
        If iUpperBoundAscii < 0 Then iUpperBoundAscii = 0
        If iUpperBoundAscii > 255 Then iUpperBoundAscii = 255
        If lLowerBoundLength < 0 Then lLowerBoundLength = 0

        'Set a random length
        lLength = Int((CDbl(lUpperBoundLength) - _
                CDbl(lLowerBoundLength) + _
                1) * Rnd + lLowerBoundLength)

        'Create the random string
        For lCount = 1 To lLength
                sHoldString = sHoldString & _
                        Chr(Int((iUpperBoundAscii - iLowerBoundAscii _
                        + 1) * Rnd + iLowerBoundAscii))
        Next
        RandomString = sHoldString
```

```
        End Function
```

Eric Lynn
Ballwin, Missouri

Friendly Enumerated Values

If you build an ActiveX control that exposes an enumerated property, you should define a Public Enum structure that gathers all the possible values for that property. Doing this helps the developer that uses your control because the enumerated values will be listed in a combo box in the Property window. However, at first glance, it seems impossible to achieve the same behavior as most of VB's intrinsic controls, which expose enumerated properties with short descriptions and embedded spaces. Even if they're not documented in the language manuals, you can create enumerated items that embed spaces by simply enclosing their names within square brackets: Public Enum DrawModeConstants Blackness = 1 [Not Merge Pen] [Mask Not Pen] [Not Copy Pen] ... End Enum Then add a DrawModeConstants property to the ActiveX control. All the enumerated values appear in the Property window of the VB IDE, without the square brackets and with all the spaces you included. Use this technique to embed other otherwise forbidden characters, such as math or punctuation symbols. **Francesco Balena**
Bari, Italy

Evaluate Polynomials Faster

The well-known Horner schema lets you calculate polynomial expressions efficiently. To calculate:

```
A*x^N + B*x^(N-1) + . + Y*x + Z ( ^ means power ),
```

simply write this expression as

```
(.((A*x + B)*x + C)*x + . +Y)*x + Z
```

Alex Bootman
Foster City, California

Enum API Constants Save Time Coding

You can simplify Win32 APIs by using enumerated types instead of constants. When you use enumerated types, VB provides you with a list of values when you define the API in your application:

```
Option Explicit

' define scrollbar constants as enumerations
Enum sb
        SB_BOTH = 3
        SB_CTL = 2
        SB_HORZ = 0
        SB_VERT = 1
End Enum

Enum esb
        ESB_DISABLE_BOTH = &H3
        ESB_DISABLE_DOWN = &H2
        ESB_DISABLE_LEFT = &H1
        ESB_ENABLE_BOTH = &H0
        ESB_DISABLE_RIGHT = &H2
        ESB_DISABLE_UP = &H1
End Enum
```

Note that you need to change the Declares to match the new Enums:

```
Private Declare Function EnableScrollBar Lib _
        "user32" (ByVal hWnd As Long, ByVal _
        wSBflags As sb, ByVal wArrows As esb) As _
        LongPrivate Declare Function _
        ShowScrollBar Lib "user32" (ByVal hWnd _
        As Long, ByVal wBar As sb, ByVal bShow _
        As Boolean) As Long
```

When coding up these API calls, VB displays enumerated lists for both the wSBflags and wArrows parameters to EnableScrollBar, and displays both the wBar and bShow parameters to ShowScrollBar:

```
Call EnableScrollBar(Me.hWnd, SB_BOTH, _
        ESB_ENABLE_BOTH)
Call ShowScrollBar(Me.hWnd, SB_BOTH, True)
```

Tom Domijan
Aurora, Illinois

Duplicate Lines of Code Without Syntax Errors

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

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

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

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

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

Draw Frames on Form Without Control

The DrawEdge API provides a convenient way to draw a number of interesting effects. You can change the EDGE_ constants to give different border effects; the BF_ constants determine which borders are drawn (for example, BF_BOTTOM):

```
Private Declare Function DrawEdge Lib "user32" (ByVal hDC _
        As Long, qrc As RECT, ByVal edge As Long, ByVal _
        grfFlags As Long) As Long
Private Declare Function GetClientRect Lib "user32" _
        (ByVal hWnd As Long, lpRect As RECT) As Long

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

Const BDR_INNER = &HC
Const BDR_OUTER = &H3
Const BDR_RAISED = &H5
Const BDR_RAISEDINNER = &H4
Const BDR_RAISEDOUTER = &H1
Const BDR_SUNKEN = &HA
Const BDR_SUNKENINNER = &H8
Const BDR_SUNKENOUTER = &H2
```

```
Const BF_RIGHT = &H4
Const BF_LEFT = &H1
Const BF_TOP = &H2
Const BF_BOTTOM = &H8

Const EDGE_BUMP = (BDR_RAISEDOUTER Or BDR_SUNKENINNER)
Const EDGE_ETCHED = (BDR_SUNKENOUTER Or BDR_RAISEDINNER)
Const EDGE_RAISED = (BDR_RAISEDOUTER Or BDR_RAISEDINNER)
Const EDGE_SUNKEN = (BDR_SUNKENOUTER Or BDR_SUNKENINNER)
Const BF_RECT = (BF_LEFT Or BF_RIGHT Or BF_TOP Or BF_BOTTOM)
```

In the Form_Paint event, put this code where you wish to draw the rectangle:

```
Private Sub Form_Paint()
        Static Tmp As RECT
        Static TmpL As Long
        TmpL = GetClientRect(hWnd, Tmp)
        TmpL = DrawEdge(hDC, Tmp, EDGE_SUNKEN, BF_RECT)
End Sub
```

If the rectangle doesn't draw, do a Debug.Print on the TmpL variable. It should read a nonzero value upon success. **Jeff Shimano**
Mississauga, Ontario, Canada

Don't Auto-Optimize for Fast Code

If you take a look at VB's native code optimization options for the first time, you might be tempted to click on "Optimize for Fast Code" right away. Strange as it may sound, though, this does not always guarantee the best performance. Applications optimized for performance generally don't run that much faster, but do have a larger memory footprint. This causes them to load slower, especially on memory-constrained machines, giving the user the impression that your app is actually slower than one optimized for compact code. For the same reason, consider leaving your applications compiled as p-code anyway. Especially for large, UI- and database-intensive applications, the performance gain of compiling to native code won't outweigh the increase in application size. To determine exactly which compilation option is right for you, use the VB Application Performance Explorer (APE) included on your VB CD. **Michiel de Bruijn**
Rotterdam, The Netherlands

Do You Know About Date Literals?

Using a Date literal is about 12 times faster-according to NuMega TrueTime-than using the CDate function and converting a string literal. Here's an example:

```
Dim TestDate as Date
        'The following 2 lines produce the same results
        TestDate = #7/1/98#
        TestDate = CDate("7/1/98")
```

Just as you enclose a string literal with quotes ("Hello"), you can enclose Date literals with pound signs (#07/07/1998#). So, these are all valid Date literals: #July 7, 1998#, #7-JUL-98#, and #07/07/1998# **James Bragg**
received by e-mail,

Customize VB Toolbars

Here are a few simple ways you can customize your VB5 IDE: Add tabs to the custom control toolbox by right-clicking on the General button and selecting the Add Tab command. You can also move tabs around and delete them, as well as move control icons from one tab to the other using the drag-and-drop method. Create toolbar buttons for any menu command by right-clicking on any toolbar and selecting the Customize command. Move to the Commands tab, select the menu command in the right-most list box, and drag it onto the toolbar where you want to move it. Good candidates for this procedure are the Project-References, Project-Properties, and Tools-Add Procedure commands.

Create a brand new toolbar in the Toolbars tab of the Customize dialog box. After you define a toolbar, add buttons using the procedure outlined above. When the Customize dialog box is active, right-click on any toolbar button to change its image, create a group divider, show/hide text, and more. **Francesco Balena**
Bari, Italy

Create an Array on the Fly with the Array Function

The GetRows method retrieves multiple rows of a Recordset (JET) or rdoResultset (RDO) into an array. I often use this feature to transfer data between an OLE Server and client applications. This method uses a Variant type variable as a parameter to store the returned data. It is internally a two-dimensional array and it is treated like one on the client side, but in declaration of the custom method on the OLE server, it looks so much tidier as variant. I've tried to pass some additional information such as field names, types, and so on. Usual means of transportation such as collections and regular arrays are either too slow or destroy the symmetry and good look in the declaration. Fortunately, the Array function returns a Variant containing an array:

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

Dejan Sunderic
Etobicoke, Ontario, Canada

Add Remarks to Your Procedures

You can make your code more readable by always adding a remark on top of all your procedures. Create an add-in that makes it fast and easy. First, run New Project under the File menu and select Addin from the project gallery that appears. In the Project Properties dialog, change the project name to RemBuilder. In the AddToIni procedure (contained in the AddIn.bas module), change the MyAddin.Connect string to RemBuilder.Connect.
Press F2 to show the Object Browser, select the RemBuilder project in the upper combo box, then right-click on the Connect class in the left-most pane and select the Properties menu command. In the dialog that appears, change the description into Automatic Remark Builder (or whatever you want).
In the IDTExtensibility_OnConnection procedure (in the Connect.cls module), search for the My Addin string and modify it to &Remark Builder. This is the caption of the menu item that will appear in the Add-Ins menu. In the Immediate window, type AddToIni and press Enter to register the add-in in the VBADDIN.ini file. In the MenuHandler_Click procedure in Connect.cls, delete the only executable line (Me.Show) and insert this code instead:

```
SendKeys "'" & String$(60, "-") & vbCrLf _
        & "' Name:" & vbCrLf _
        & "' Purpose:" & vbCrLf _
        & "' Parameters:" & vbCrLf _
        & "' Date: " & Format$(Now, "mmmm,dd yy") _
        & "' Time: " & Format$(Now, "hh:mm") & vbCrLf _
        & "'" & String$(60, "-") & vbCrLf
```

Compile this program into an EXE or a DLL ActiveX component, then install the add-in as usual from the Add-In Manager. Before you create a procedure, select the Remark Builder menu item from the Add-Ins menu to insert a remark template in your code window, and you'll never again have to struggle against an under-documented program listing. **Francesco Balena**
Bari, Italy

Reduce the Clutter in Your VB IDE

Here's another simple but useful add-in you can add to your arsenal. Follow the directions given in the previous tip "Add Remarks to Your Procedures," with only minor differences. Use the project name CloseWindows rather than RemBuilder. Also, change the description to "Close All IDE

Windows." Finally, type a suitable caption for the menu command, such as Close IDE &Windows. Insert this code in the MenuHandler_Click procedure:

```
Dim win As VBIDE.Window
For Each win In VBInstance.Windows
If win Is VBInstance.ActiveWindow Then
' it's the active window, do nothing
ElseIf win.Type = vbext_wt_CodeWindow Or _
        win.Type = vbext_wt_Designer Then
' code pane or designer window
win.Close
End If
Next
```

When you select the add-in from the Add-Ins menu, it closes all the forms and code windows currently open, except the one you're working with. **Francesco Balena**
Bari, Italy

Read and Write Arrays Quickly

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

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

Rod Stephens
Boulder, Colorado

Reduce Filtering Frustration

This code works wonders to reduce flicker and lessen your frustration. Place a timer on the form (tmr_Timer) and set the Interval to 1000. Set Enabled to False, then place this code in the txt_Filter_Change event:

```
Private Sub txtFilter_Change()
        Timer1.Enabled = False
        Timer1.Enabled = True
End Sub
```

In the Timer event, call this routine that refreshes your recordset:

```
Private Sub Timer1_Timer()
        Timer1.Enabled = False
        Call MyUpdateRecordsetRoutine
End Sub
```

The recordset will only be updated if you haven't pressed a key for a full second. Each time you press a key, the timer is reset and the one-second countdown starts all over again. **Tom Welch**
received by e-mail,

ReDim the Right Array!

Many VB programmers use the Option Explicit statement to make sure each variable has been explicitly declared before using it. This means you'll always notice a misspelled variable, which if not caught might cause your application to behave erratically. However, when you use the ReDim statement (documented, albeit ambiguously), Option Explicit can't save you. Consider this procedure:

```
Sub DisplayDaysInThisYear
        Dim iDaysInYear(365)
        ' Initially dimension array

        If ThisIsLeapYear() Then
        ' Is this year a leap year?
```

```
                ReDim iDaysInYr(366)
                ' Extra day this year!
        End If

        MsgBox "This year has " & _
                UBound(iDaysInYear) & " days in it!"
End Sub
```

This ReDim statement creates a new variable called iDaysInYr, even though you really wanted to reallocate the storage space of the iDaysInYear() array. So the message box displays the incorrect number of days in the year. You can't prevent this from happening, other than being careful when coding the ReDim statement. However, if you use ReDim Preserve, Option Explicit makes sure the variable was previously declared. **Frank Masters Grove City, Ohio**

Replacement for Now() and Timer()

The simple BetterNow() function, shown here, replaces the built-in Now() function. It's faster (10 microseconds vs. 180 microseconds on a Pentium 166MMX) and more accurate, potentially supplying one-millisecond resolution, instead of 1000 milliseconds.
Because it's also faster and more accurate than Timer(), which clocks at 100 microseconds and provides 55 milliseconds resolution, it should also replace Timer, especially when Timer() is used to measure elapsed times. Besides, Timer() rolls over at midnight, and BetterNow() doesn't:

```
#If Win16 Then
        Private Declare Function timeGetTime Lib _
                "MMSYSTEM.DLL" () As Long
#Else
        Private Declare Function timeGetTime Lib "winmm.dll" _
                () As Long
#End If

Function BetterNow() As Date
        Static offset As Date
        Static uptimeMsOld As Long
        Dim uptimeMsNew As Long
        Const oneSecond = 1 / (24# * 60 * 60)
        Const oneMs = 1 / (24# * 60 * 60 * 1000)
        uptimeMsNew = timeGetTime()
        ' check to see if it is first time function called or
        ' if timeGetTime rolled over (happens every 47 days)
        If offset = 0 Or uptimeMsNew < uptimeMsOld Then
                offset = Date - uptimeMsNew * oneMs + CDbl(Timer) * _
                        oneSecond
                uptimeMsOld = uptimeMsNew
        End If
        BetterNow = uptimeMsNew * oneMs + offset
End Function
```

Andy Rosa
received by e-mail,

Resize the Drop-Down List Area of Combo Boxes

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

```
Option Explicit

Type POINTAPI
        x As Long
```

```
        y As Long
End Type

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

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

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

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

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

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

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

        'Resize the combo box
        Call MoveWindow(rCbo.hwnd, pt.x, _
                pt.y, iItemWidth, iItemHeight, 1)
End Sub
```

Keith Meulemans
Green Bay, Wisconsin

Right-Justify or Left-Justify Text

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

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

Rod Stephens
Boulder, Colorado

Retrieving a Control From the Controls Collection With an hWnd

The GetDlgCtrlID API, when passed a valid hWnd, returns a value that directly corresponds to the Index property of the Controls collection:

```
Private Declare Function GetDlgCtrlID Lib "user32" _
        (ByVal hWnd As Long) As Long

Private Sub Form_Load()
        Dim i As Long

        On Error Resume Next
        For i = 0 To Controls.Count - 1
                Debug.Print Controls(i).Name,
                Debug.Print Controls(GetDlgCtrlID(Controls(i).hWnd) _
                        - 1).Name
        Next i
End Sub
```

This loop, located in the Form_Load event of a form with a number of controls on it, loops through all the controls and prints the name of each windowed control twice, demonstrating that it has correctly located the control without looping through the control collection.

Jeremy Adams
Tiverton, Devon, United Kingdom

Roll-Your-Own Decimal Entry Filter

Here's an easy method for making sure your users enter only numeric data, and only one decimal point. First, place two Public procedures in a standard module. You can use Private procedures in a form if you're only using it there, but you'll lose easy portability for future projects.

The first procedure makes sure the decimal point is only entered once. The second procedure filters out all non-numeric characters except the decimal point:

```
Public Sub DecCheck(Target As String, ByRef KeyStroke As _
        Integer)
        If InStr(Target, ".") And KeyStroke = 46 Then
                KeyStroke = 0
        End If
End Sub

Public Sub NumCheck(ByRef KeyStroke As Integer)
        If (KeyStroke < 48 Or KeyStroke > 57) And (KeyStroke _
                <> 46 And KeyStroke <> 8) Then
                KeyStroke = 0
        End If
End Sub
```

Then invoke the code from your TextBox's KeyPress event:

```
Private Sub txtUnitPrice_KeyPress(KeyAscii As Integer)
        DecCheck txtUnitPrice, KeyAscii
        NumCheck KeyAscii
End Sub
```

One caveat: This code doesn't prevent text characters from being pasted in via the clipboard.

Ron Schwarz
Mt. Pleasant, Michigan

Rotate an Object About a Point

You can rotate any object about a center using polar coordinates. Simply define your center Xo and Yo, which in this case is the center of a form. The amount of rotation is determined by direction, one degree:

```
Private Direction As Long
Private Xo As Long, Yo As Long

Private Sub Form_Click()
        If Direction = 1 Then
                Direction = 359 'counterclockwise
        Else
                Direction = 1           'clockwise
        End If
End Sub

Private Sub Form_Load()
                Direction = 1           'clockwise
End Sub

Private Sub Form_Resize()
        Xo = Me.ScaleWidth \ 2
        Yo = Me.ScaleHeight \ 2
End Sub

Private Sub Timer1_Timer()
        Dim i  As Byte
        Dim r As Single
        Dim Pi As Single
        Dim theta As Single
        Dim plotx, ploty, dx, dy As Integer

        Xo = Form1.Width / 2
        'get center, image is to rotate about
        Yo = Form1.Height / 2
        Pi = 4 * Atn(1)
        dx = Image1.Left - Xo
        'get horizontal distance from center
        dy = Image1.Top - Yo
        ' ""  vertical  ""
        theta = Atn(dy / dx)
        'get angle about center
        r = dx / Cos(theta)
        'get distance from center
        plotx = r * Cos(theta + Direction * Pi / 180) + Xo
        'get new x rotate about center
        ploty = r * Sin(theta + Direction * Pi / 180) + Yo
        '       ""      y       ""
        Image1.Left = plotx
        Image1.Top = ploty
End Sub
```

David A. Sorich
Countryside, Illinois

Shortcuts for the VB Environment

1) In VB5, pressing Ctrl-F3 when the cursor is over a word automatically searches to the next occurrence of that word, bypassing the search dialog. You need to be past the first character of the word for it to work properly. 2) VB4/5 Ctrl-Tab cycles through all your open windows in the IDE often quicker than going to the Window menu. **Tim Jones**
Castlemaine, Victoria, Australia

Show 3-D Text Messages

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

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

Atmabodh Hande
Shamla Hills, Bhopal, India

Showing "&" Character in Labels

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

Speed up your Code Using Choose

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

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

Francesco Balena
Bari, Italy

Taking a Form in Front of Another Form

When building a floating toolbar, you might need to keep it in front of the main form of your application. This took time to do in VB3 and VB4, because you had to resort to API functions. In VB5, you can take advantage of a new, optional argument of the Show method:

```
' within the main form
frmFloating.Show 0, Me
```

The second argument sets the owner form for the window being displayed. The "owned" form will always be in front of its owner, even when it doesn't have the input focus. Moreover, when the owner form is closed, all its owned forms are automatically closed also. **Francesco Balena**
Bari, Italy

Test for "File Exist" the Right Way

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

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

Pedro Prospero Luis
Odivelas, Portugal

Tie a Message Box to Debug.Assert for Advanced Debugging

Placing a message box in an error trap can provide useful debugging information, but it doesn't allow you to return to the subroutine or function to poke around and further debug the code. This version of a message box expedites design-time debugging by breaking execution if the developer presses OK:

```
Private Function MyDebugMsg(ByVal aMessage _
        As String) As Boolean
        ' This function is used for expediting
        ' development
        If MsgBox(aMessage, vbOKCancel, _
                "OK puts you into the Error Trap") = vbOK Then
                MyDebugMsg = False
        Else
                MyDebugMsg = True
        End If
End Function

' Sample sub
Public Sub SetColor()
On Error GoTo SetColorError

' body of the subroutine would go here,
```

```
' force an error to demonstrate
Error 5

SetColorErrorExit:
        Exit Sub

SetColorError:
        ' In an error trap place this line in addition to any
        ' other error handling code
        Debug.Assert MyDebugMsg(Err.Description & " in SetColor")

        'other error handling code
        Resume SetColorErrorExit
End Sub
```

Stan Mlynek
Burlington, Ontario, Canada

Translate Color Values

With the RGB function, VB provides a neat and valuable tool for converting separate Red, Green, and Blue values into a single Long color value.. However, VB won't let you translate back from its this color value to back to its constituent RGB values. But, you can pick the individual colors out of a hexadecimal representation of the Long value produced by RGB. The colors fall in "BBGGRR" order. Put this code in a module:

```
Type RGB_Type
        R As Long
        G As Long
        B As Long
End Type

Function ToRGB(ByVal Color As Long) As RGB_Type
        Dim ColorStr As String
        ColorStr = Right$("000000" & Hex$(Color), 6)
        With ToRGB
        .R = Val("&h" & Right$(ColorStr, 2))
        .G = Val("&h" & Mid$(ColorStr, 3, 2))
        .B = Val("&h" & Left$(ColorStr, 2))
        End With
End Function
```

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

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

Click on different places on the picture. VB3 users must return the values differently, because VB didn't support the return of a user-defined type until VB4. **Brian Donovan**
Bakersfield, California

Trapping a Double Click for a Toolbar Button

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

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

```
        In the Toolbars ButtonClick event, add this code:

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

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

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

Sushrut Nawathe
Pune, India

Type-o-matic Text Box

This code creates a smart input box. Every time you type something into this text box, the first letters of your string are compared against the members of a hidden list box. The code guesses how your string should be completed and finishes it for you, similar to how the latest versions of Microsoft Excel and Internet Explorer behave.

To use this technique, add a list box to your form and set its Visible property to False. This example fills the list at Form_Load with some likely selections. In a real app, you'd add a new element to the list after each user entry is completed. Add this code to the form containing the text and list boxes:

```
Option Explicit

#If Win32 Then
        Private Const LB_FINDSTRING = &H18F
        Private Declare Function SendMessage Lib _
```

```
                "User32" Alias "SendMessageA" (ByVal _
                hWnd As Long, ByVal wMsg As Long, _
                ByVal wParam As Long, lParam As Any) _
                As Long
#Else
        Private Const WM_USER = &H400
        Private Const LB_FINDSTRING = (WM_USER + 16)
        Private Declare Function SendMessage Lib _
                "User" (ByVal hWnd As Integer, ByVal _
                wMsg As Integer, ByVal wParam As _
                Integer, lParam As Any) As Long
#End If

Private Sub Form_Load()
        List1.AddItem "Orange"
        List1.AddItem "Banana"
        List1.AddItem "Apple"
        List1.AddItem "Pear"
End Sub

Private Sub Text1_Change()
        Dim pos As Long
        List1.ListIndex = SendMessage( _
                List1.hWnd, LB_FINDSTRING, -1, ByVal _
                CStr(Text1.Text))
        If List1.ListIndex = -1 Then
                pos = Text1.SelStart
        Else
                pos = Text1.SelStart
                Text1.Text = List1
                Text1.SelStart = pos
                Text1.SelLength = Len(Text1.Text) - pos
        End If
End Sub

Private Sub Text1_KeyDown(KeyCode As _
        Integer, Shift As Integer)
        On Error Resume Next
        If KeyCode = 8 Then 'Backspace
                If Text1.SelLength <> 0 Then
                        Text1.Text = Mid$(Text1, 1, _
                                Text1.SelStart - 1)
                        KeyCode = 0
                End If
        ElseIf KeyCode = 46 Then 'Del
                If Text1.SelLength <> 0 And _
                        Text1.SelStart <> 0 Then
                        Text1.Text = ""
                        KeyCode = 0
                End If
        End If
End Sub
```

Paolo Marozzi
Ascoli Piceno, Italy

Use Backquotes Instead of Apostrophes

Often when using Transact-SQL, I want to capture comments from a user in a text box and send them to the database. However, if the user types an apostrophe in the text box, a run-time error is generated when the update is processed, because SQL Server thinks the apostrophe is being used to mark the end of a string. To get around this problem, intercept the user's keystrokes

in the KeyPress event and exchange the apostrophe with an "upside-down" quote mark (ASCII(145)) like this:

```
Private Sub Text1_Keypress_
        (KeyAscii as Integer)
        If KeyAscii = 39 Then
                KeyAscii = 145
        End If
End Sub
```

Alternatively, you might decide to substitute all occurrences of single quotes into backquotes immediately before sending them to SQL Server. **Mike McMillan**
North Little Rock, Arkansas

Use MouseMove for Easy StatusBar Updates

You can easily make your program show descriptive text on a StatusBar control in response to mouse movement. Assign the text to the appropriate panel in the MouseMove events of the appropriate controls, then use the Form_MouseMove event to clear text from the panel:

```
Private Sub txtAddress_MouseMove(Button As Integer, Shift _
        As Integer, X As Single, Y As Single)
        StatusBar1.Panels(1).Text = "Enter Address here."
End Sub

Private Sub txtName_MouseMove(Button As Integer, Shift _
        As Integer, X As Single, Y As Single)
        StatusBar1.Panels(1).Text = "Enter Name here."
End Sub

Private Sub Form_MouseMove(Button As Integer, Shift _
        As Integer, X As Single, Y As Single)
        StatusBar1.Panels(1).Text = ""
End Sub
```

Ron Schwarz
Mt. Pleasant, Michigan

Use Name Parameters With Oracle Stored Procedures

When executing an Oracle stored procedure, use the named parameter convention. In place of this code:

```
OraDatabase.ExecuteSQL _
        ("Begin Employee.GetEmpName (:EMPNO, :ENAME); end;")

        Use this code:

OraDatabase.ExecuteSQL ("Begin Employee.GetEmpName _
        (empno=>:EMPNO, ename=>:ENAME); end;")
```

The second example still works even if you change the positions of the stored-procedure arguments. Also, with this convention, you can write a generic routine to assemble the SQL statement without worrying about positioning the stored-procedure arguments.
Arnel J. Domingo
Hong Kong, China

Using the Format Function With Strings

You'll use the Format function most often with numbers, but it can be useful when applied to strings as well. For example, you can format a credit card number-which is held in a string variable, even if it contains only digits-and subdivide the number into four groups of four characters each, using a complex string expression:

```
' X holds the sequence of 16 digits
CreditCardNum = Left$(x, 4) & " " & Mid$(x, 5, 4) & " " & _
        Mid$(x, 9, 4) & " " & Right$(x, 4)
```

The Format function lets you accomplish the same result in a more readable and efficient way:

```
CreditCardNum = Format$(x, "!@@@@ @@@@ @@@@ @@@@")
```

Francesco Balena
Bari, Italy

Using Label Control as Splitter

Here's a demo for using a Label control as a splitter between two controls, as well as sample code for employing the splitter in an Explorer-like application:

```
Option Explicit

Private mbResizing As Boolean
        'flag to indicate whether mouse left
        'button is pressed down

Private Sub Form_Load()
        TreeView1.Move 0, 0, Me.ScaleWidth / 3, _
                Me.ScaleHeight
        ListView1.Move (Me.ScaleWidth / 3) + 50, 0, _
                (Me.ScaleWidth * 2 / 3) - 50, _
                Me.ScaleHeight
        Label1.Move Me.ScaleWidth / 3, 0, 100, _
                Me.ScaleHeight
        Label1.MousePointer = vbSizeWE
End Sub

Private Sub Label1_MouseDown(Button As Integer, Shift As _
        Integer, X As Single, Y As Single)
        If Button = vbLeftButton Then mbResizing = _
                True
End Sub

Private Sub Label1_MouseMove(Button As _
        Integer, Shift As Integer, X As _
        Single, Y As Single)
        'resizing controls while the left mousebutton is
        'pressed down
        If mbResizing Then
                Dim nX As Single
                nX = Label1.Left + X
                If nX < 500 Then Exit Sub
                If nX > Me.ScaleWidth - 500 Then Exit Sub
                TreeView1.Width = nX
                ListView1.Left = nX + 50
                ListView1.Width = Me.ScaleWidth - nX - _
                        50
                Label1.Left = nX
        End If
End Sub

Private Sub Label1_MouseUp(Button As Integer, _
        Shift As Integer, X As Single, Y As Single)
        mbResizing = False
End Sub
```

Rajesh R. Vakharia
Mumbai, India

Use TypeName Instead of TypeOf...Is

To write reusable routines that work with multiple types of controls, test the control type using the TypeName function in place of the TypeOf...Is statement. For example, take a look at this routine-you can reuse it in another project only if you also add the RichTextBox control to the Components list:

```
' save the selected text to an open file
' works with TextBox and RichTextBox controls
Sub SaveSelectedText(ctrl As Control, filenum As Integer)
        If TypeOf ctrl Is TextBox Then
                Print #filenum, ctrl.SelText
        ElseIf TypeOf ctrl Is RichTextBox Then
                Print #filenum, RichTextBox1.SelRTF
        End If
End Sub
```

To avoid this problem and gain additional benefits such as the ability to use a Select Case block, use the TypeName function instead:

```
Sub SaveSelectedText(ctrl As Control, filenum As Integer)
        Select Case TypeName(ctrl)
                Case "TextBox"
                        Print #filenum, ctrl.SelText
                Case "RichTextBox"
                        Print #filenum, RichTextBox1.SelRTF
        End Select
End Sub
```

Francesco Balena
Bari, Italy

Use This Higher-Resolution Stopwatch

Use this code to create a class called HiResTimer:

```
'The number is codified as HighPart*2^32+LowPart
Private Type LARGE_INTEGER
        LowPart As Long
        HighPart As Long
End Type

Private Declare Function QueryPerformanceCounter Lib _
        "kernel32" (lpPerformanceCount As LARGE_INTEGER) _
        As Long
Private Declare Function QueryPerformanceFrequency Lib _
        "kernel32" (lpFrequency As LARGE_INTEGER) As Long

Private m_TicksPerSecond As Double
Private m_LI0 As LARGE_INTEGER
Private m_LI1 As LARGE_INTEGER

Friend Sub Class_Initialize()
        Dim LI As LARGE_INTEGER

        If QueryPerformanceFrequency(LI) <> 0 Then
                m_TicksPerSecond = LI2Double(LI)
        Else
                m_TicksPerSecond = -1
        End If
End Sub

Friend Property Get Resolution() As Double
        Resolution = 1# / m_TicksPerSecond
End Property

Friend Sub EnterBlock()
```

```
        QueryPerformanceCounter m_LI0
End Sub

Friend Sub ExitBlock()
        QueryPerformanceCounter m_LI1
End Sub

Friend Property Get ElapsedTime() As Double
        Dim EnterTime As Double, ExitTime As Double

        EnterTime = LI2Double(m_LI0) / m_TicksPerSecond
        ExitTime = LI2Double(m_LI1) / m_TicksPerSecond
        ElapsedTime = ExitTime - EnterTime
End Property

Friend Function LI2Double(LI As LARGE_INTEGER) As Double
        Dim Low As Double
        Const TWO_32 = 4# * 1024# * 1024# * 1024#

        Low = LI.LowPart
        If Low < 0 Then Low = Low + TWO_32
                'Now Low is in the range 0...2^32-1

                LI2Double = LI.HighPart * TWO_32 + Low
End Function
```

Here's an example of the HiResTimer in use:

```
Dim hrt As HiResTimer, d As Double
Set hrt = New HiResTimer
Debug.Assert hrt.Resolution > 0
MsgBox "Resolution [usecs]:" & hrt.Resolution * 1000000#

hrt.EnterBlock
hrt.ExitBlock
MsgBox "Call overhead [usecs]:" & hrt.ElapsedTime * _
        1000000#

hrt.EnterBlock
d = 355# / 113#
hrt.ExitBlock

MsgBox "Elapsed Time [usecs]:" & hrt.ElapsedTime * _
        1000000#
```

Believe it or not, you can time even native-compiled code division. For more information, look at the MSDN Library description of the kernel APIs used here. On x86 architectures, resolution is better that 1 microsecond. Be careful, however, of trusting single instance timings, as you'll find the "resolution" of this performance counter varies over time. In fact, the overhead of simply calling QueryPerformanceCounter in VB is quite a measurable time period itself.

Although you can time single operations, you're still better off averaging the time required for hundreds or thousands of similar operations.

Alessandro Coppo
Rapallo, Italy

Use Toolbar-Style Title Bars

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

```
Declare Function GetWindowLong Lib "user32" _
        Alias "GetWindowLongA" ( _
        ByVal hwnd As Long, _
        ByVal nIndex As Long) As Long
Declare Function SetWindowLong Lib "user32" _
```

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

Rod Stephens
Boulder, Colorado

Use Unadvertised Controls

When you open VB5's Components list, you'll see many controls and libraries not available for your development. Some are controls you downloaded from Web pages; others come from who knows where.

If you've ever tried adding an unknown control to your IDE, you probably saw an icon added to your control's palette. However, since you couldn't use the control, you probably just ignored them all and selected the controls that you're positive came with your copy of VB.

Wait! Open that Component list again and select these items:

```
Wang Image Admin Control
Wang Image Scan Control
Wang Image Edit Control
Wang Image Thumbnail Control.
```

Under Windows 98, the name "Kodak" is used, rather than "Wang." Add these items to your palette, then add them to a form. Select the control and press F1. Up pops the developer's help on using the controls in your projects.

These may not be the final word on imaging controls, but with all their properties and methods for image manipulation, conversions, displays, and more, they're leaps and bounds beyond picture and image controls, and they're free-with Windows 95/OSR2, Windows 98, and NT4. The one restriction you need to be aware of is that these controls are not redistributable, and Windows 95 users must download them (from http://www.eastmansoftware.com) and perform the separate install themselves.

Robert Smith
San Francisco, California

Use VB System Color Constants in API Calls

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

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

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

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

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

Steve Cisco
Perrysburg, Ohio

Watch How You Use Your Booleans

With the introduction of the Boolean data type in VB4, you might be tempted to convert it to a numeric value using the Val function for storage in a database table. Watch out! Val won't convert a Boolean into -1 (or 1) as you might expect. Use the Abs or CInt functions, depending on the format you need:

```
Val(True) gives 0
CInt(True) gives -1
Abs(True) gives 1
```

Joe Karbowski
Traverse City, Michigan

Watch Out for "()" When Calling Subroutines

To call a subroutine, you can use the Call statement or simply the name of the subroutine:

```
        Call MyRoutine(firstParameter)
        'Or
        MyRoutine firstParameter
```

Notice you don't include the parentheses in the second case. If you do, VB assumes you mean them as an operator. VB then determines the value of the parameter and passes the value to the routine, instead of passing the reference as expected. This is apparent in this example:

```
        Call MyRoutine(Text1)
```

This passes the text-box control to MyRoutine. If you did it without the Call statement, VB evaluates Text1, which returns the default property value of the text box:

```
        MyRoutine(Text1)
```

This default property is the text-box text. So, if the routine expects a control, you pass the text string from the control instead and will receive a type-mismatch error. To prevent this, always use the Call statement or don't put parentheses in when calling a subroutine.

Deborah Kurata
Pleasanton, California

Working With Collections

When working with collections, use an error handler to easily determine if a given key exists in the collection. If you try to access an item from a collection where the key doesn't exist, you'll get an error. Likewise, if you try to add an item that exists, you'll also get an error. This example shows an error handler for adding an item to a collection. To trap for errors where an item exists, trap error code 457:

```
Private Function BuildCustCol(CustList As ListBox) As _
```

```
        Collection
        On Error GoTo ProcError
        Dim colCust As Collection
        Dim lngCustCnt As Long
        Dim J As Long

        Set colCust = New Collection
        For J = 0 To CustList.ListCount - 1
                lngCustCnt = colCust(CStr(CustList.List(J))) + 1
                colCust.Remove (CStr(CustList.List(J)))
                colCust.Add Item:=lngCustCnt, _
                        Key:=CStr(CustList.List(J))
        Next J
        Set BuildCustCol = colCust
        Set colCust = Nothing
        Exit Function

ProcError:
        Select Case Err
                Case 5 'collection item doesn't exist, so add it
                        colCust.Add Item:=0, _
                                Key:=CStr(CustList.List(J))
                        Resume
                Case Else
                'untrapped error
        End Select

End Function
```

Joe Karbowski
Traverse City, Michigan

Where Did It Go?

Have you ever wondered why your ActiveX DLL with a form doesn't show up in the taskbar? Because you're showing the form modally (.Show vbModal). VB4 only allows DLLs with a user interface to be shown modally. VB5, however, has no such limitation. If you want your VB5 DLL to show up in the taskbar, you need to change your code to support showing it nonmodally.

Joe Karbowski
Traverse City, Michigan

Beginner # Watch the Parens If you want to pass a parameter to a subroutine, use this code:

```
        Call doFormat(txtPerson)
```

You can also call the subroutine without the Call statement. However, if you don't include the Call statement, you can't include parentheses:

```
        doFormat (txtPerson)
```

In VB, expressions in parentheses are evaluated before they're processed. So by putting parentheses around the control name, you're telling it to evaluate it. Because a control can't be evaluated, it gives you the value of the default property. This code actually passes the Text string value-because Text is the default property-to the subroutine instead of passing the control. Because the routine expects a textbox and not a string, it generates the type mismatch. **Deborah Kurata**
Pleasanton, California

Yet Another CenterForm Routine In the April 1997 issue of VBPJ, you published a tip called "Consider the Taskbar When Centering Forms." You can center forms more easily with the SystemParametersInfo API call:

```
Private Declare Function _
        SystemParametersInfo Lib "user32" Alias _
        "SystemParametersInfoA" (ByVal uAction _
        As Long, ByVal uParam As Long, R As Any, _
        ByVal fuWinIni As Long) As Long
Private Type RECT
        Left As Long
        Top As Long
        Right As Long
        Bottom As Long
End Type
Private Const SPI_GETWORKAREA = 48
Public Sub CenterForm(frm As Form)
        Dim R As RECT, lRes As Long,
        Dim lW As Long, lH As Long
        lRes = SystemParametersInfo( _
                SPI_GETWORKAREA, 0, R, 0)
        If lRes Then
                With R
                        .Left = Screen.TwipsPerPixelX * .Left
                        .Top = Screen.TwipsPerPixelY * .Top
                        .Right = Screen.TwipsPerPixelX * .Right
                        .Bottom = Screen.TwipsPerPixelY * .Bottom
                        lW = .Right - .Left
                        lH = .Bottom - .Top
                        frm.Move .Left + (lW - frm.Width) \ 2, _
                                .Top + (lH - frm.Height) \ 2
                End With
        End If
End Sub
```

Nicholas Sorokin
Sarasota, Florida

Write Less CPU-Bound Animations

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

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

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

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

Derek Robinson
Pretoria, South Africa

Conditionally Compile Your Code

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

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

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

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

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

End Function
```

Carl Denton
Marietta, Georgia

Switch

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

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

Switch is a rarely used function, yet in many cases it proves rather useful as a substitute for a lengthy If...ElseIf block:

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

Note the last test is True, because the three conditions are mutually exclusive and exhaustive. **Francesco Balena**
Bari, Italy

Comment Multiple Lines

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

```
Public Sub TestingSub()
        Print "This subroutine is used to"
        Print "demonstrate block"
        Print "commenting in VB."
#If COMMENT then
```

```
        Print "This line will not be printed."
        Print "Since this is commented out."
        Print "VB ignores these lines during compilation."
#End If
End Sub
```

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

Frewin Chan
Scarborough, Ontario, Canada

Comment and Uncomment Blocks of Code

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

Combine Default with Other Attributes

When building an ActiveX control, you can set a default property or method using the Procedure Attributes dialog box, after clicking on the Advanced button. However, if your default property also happens to require another special attribute-as is the case with Caption and Text properties-you're in trouble because the Procedure ID combo box only permits one selection. Suppose your ActiveX control exposes a Caption property you want to behave as a regular property-for example, all keys typed in the Property window are immediately reflected in the control itself. In order to achieve this behavior, assign the Caption attribute to this property in the Procedure ID combo box (see tip "Properties That Behave Like Text and Caption"). If you also want to make it the default property, you must resort to a trick: declare another, hidden property that delegates to your Caption property, and set this new property as the default member of the ActiveX control. The name of this property is not important because the user never sees it:

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

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

Francesco Balena
Bari, Italy

Convert from Fractional to Decimal Numbers

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

```
Private Function ReturnDecimalValue(Size As _
        String) As String
```

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

Randall Arnold
Coppell, Texas

Collect User Requirements With Scenarios

When talking to the user or subject-matter expert about an application's requirements, write the requirements in the form of scenarios. A scenario both defines the requirement and provides a list of steps detailing how the resulting feature will be used. For example, instead of writing a requirement to "process payroll," your scenario might be to select an employee from a list of existing employees, to enter the time allocated to the project for each employee, and so on. This clarifies requirements and helps you better visualize how users will use the feature. Once you understand the reasoning behind the request, you might even find a better way to meet the requirement. You can then use these scenarios as the test plan for the feature.
Deborah Kurata
Pleasanton, California

Code-Commenting Shortcuts

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

Close VB Before Compiling

When you're finished tinkering with your apps, close and restart VB before making the final EXE. This simple action can reduce the size of your EXE by 10 to 30 percent (many professional programmers also recommend restarting Windows before building an EXE). If you don't close and restart VB, your EXE may contain some garbage: VB doesn't fully clean up all the data structures or variables you used during development. Restarting VB also safeguards against some mysterious GPFs. If you have an app that runs fine in the development environment but GPFs when it's run as an EXE, try closing and restarting. Another option is to compile from the "command line." To do so from either Program Manager or File Manager, select Run from the File menu, and enter:

```
C:\VB\VB.EXE /MAKE D:\APPPATH\MYPROJ.MAK
```

Patrick O'Brien & Karl Peterson

Close Forms Uniformly

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

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

Rod Stephens
Boulder, Colorado

Close all MDI Children Simply

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

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

Clean Up Project File Paths Before Sharing Source Code

As you work with a VB project, the project file--VBP--can become littered with relative path references such as "..\..\..\..\myfolder\myform.frm". The project loads, but only on your machine. If you send the project to someone else, or move it to another path on your own machine, you need to edit the project file to remove the ambiguous entries. You can avoid this by ensuring that all the needed files are indeed in the same directory as the project file. It's not uncommon to load a file from a different directory, in which case VB does not automatically move it into your project directory. Load the project file into Notepad and edit out all path references, leaving only the actual file names. When VB goes to load the project, it looks for them in the current directory. **Ron Schwarz**
Mt. Pleasant, Michigan

Cheap Focus Tracking

The Lost_Focus and Got_Focus events are the most-used events for implementing validation and text highlighting. Wouldn't it be nice to respond instantly to these events and to do it from a single routine for all controls without the aid of a subclassing control? Here's the answer. Place a timer control on your form, set its

Interval property to 100 and set Enabled = True. Name the control tmrFocusTracking. Code its Timer event alike this:

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

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

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

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

To implement validation, replace the Print statement with a call to a validation routine. Use strActive in a Select Case structure. At the moment where the Print statement would occur, strActive is equal to the control that just Got Focus, and strControlName holds the name of the control that just Lost Focus. Don't place this routine in anything but a timer; otherwise, your program hangs once the routine is called. Even the timer here never makes it to a second interval. For a given control, don't write validation code both in the Got_Focus/Lost_Focus events, and in code called by this routine. Doing so might cause unpredictable results. **John S. Frias**
Santa Maria, California

Change Display Settings on the Fly

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

```
'- Declares
Private Declare Function lstrcpy _
        Lib "kernel32" Alias "lstrcpyA" _
        (lpString1 As Any, lpString2 As Any) _
        As Long
Const CCHDEVICENAME = 32
Const CCHFORMNAME = 32
Private Type DEVMODE
        dmDeviceName As String * CCHDEVICENAME
        dmSpecVersion As Integer
        dmDriverVersion As Integer
        dmSize As Integer
        dmDriverExtra As Integer
        dmFields As Long
        dmOrientation As Integer
        dmPaperSize As Integer
        dmPaperLength As Integer
        dmPaperWidth As Integer
        dmScale As Integer
        dmCopies As Integer
        dmDefaultSource As Integer
        dmPrintQuality As Integer
        dmColor As Integer
```

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

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

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

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

If the function is successful, it returns zero.

Huang Xiongbai
Shanghai, China

Center Forms with Taskbar Visible

Just about every VB developer uses the Move (Screen.Width - Width) \ 2, (Screen.Height - Height) \ 2 method to center the forms on screen. However, when the user has the Windows 95 or NT 4.0 taskbar visible, your form centers on screen but doesn't take into account the position of the taskbar

itself. The CenterForm32 routine centers a form in available screen area, taking into account the taskbar. Add this code to the Declarations section of a module, and put the code CenterForm32 Me on the Form_Load event of the forms you want to center:

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

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

Miguel Santos
Aveiro, Portugal

Center Forms on Screen

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

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

Francesco Balena
Bari, Italy

```
'
Private Declare Sub keybd_event Lib "user32" (ByVal bVk As _
        Byte, ByVal bScan As Byte, ByVal dwFlags As Long, _
        ByVal dwExtraInfo As Long)

Public Enum SystemKeyShortcuts
        ExplorerNew = &H45      ' Asc("E")
        FindFiles = &H46                ' Asc("F")
        MinimizeAll = &H4D      ' Asc("M")
        RunDialog = &H52                ' Asc("R")
        StartMenu = &H5B                ' Asc("[")
        StandbyMode = &H5E      ' Asc("^") -- Win98 only!
```

```
End Enum

Public Sub SystemAction(VkAction As SystemKeyShortcuts)
        Const VK_LWIN = &H5B
        Const KEYEVENTF_KEYUP = &H2
        Call keybd_event(VK_LWIN, 0, 0, 0)
        Call keybd_event(VkAction, 0, 0, 0)
        Call keybd_event(VK_LWIN, 0, KEYEVENTF_KEYUP, 0)
End Sub
```

Randy Birch
East York, Ontario, Canada

Browse VB Command as You Type

When you refer to an object in VB5, you get a drop-down list of that object's properties and methods. But, did you know that the statements and functions of the VB language itself are just a big list of properties and methods? You can view this list at any time in a VB code window by typing the name of the library in which this code resides:

```
        VBA.
```

Once you type the dot after VBA, the bulk of the VB language drops down. You can then select the language element you want from the list. This is a great help when you're trying to remember the name of a VB language element that you don't often use. **Jeffrey McManus**
'San Francisco, California

Dim x As Integer Dim y As Integer Dim z As Integer x = 10 y = 20 z = 0 ""Assume function max returns the maximum ""of the two if (z = max(x, y)) > 0 then Msgbox CStr(z) Else Msgbox "How Come?" End if
'Baskar S. Ganapathy
'Walnut Creek, California

'stop multiple event cascades

```
Private Sub Form_Resize()
        Static Executing As Boolean
        If Executing Then
                Exit Sub
        End If

        Executing = True
        If Width > 6000 Then
                Width = 6000
                GoDoSomeStuff
        End If
        Executing = False
End Sub
```

'Ron Schwarz
'Mt. Pleasant, Michigan

'**Bruce Goldstein**
'Highlands Ranch, Colorado

#If Win16 Then Declare Function LockWindowUpdate Lib _ "User" (ByVal hWndLock As Integer) As Integer #Else Declare Function LockWindowUpdate Lib _ "user32" (ByVal hWndLock As Long) As Long #End If

```
'vb tip lockwindowupdate
Dim lErr as Long
Dim x as Integer
```

```
'No list box flicker, it will appear blank for
'just a moment...
Screen.MousePointer = vbHourglass
lErr = LockWindowUpdate(Me.hWnd)

For x = 1 to 5000
        lstMyListbox.AddItem CStr(x)
Next

        Now all the information is there:

lErr = LockWindowUpdate(0)
Screen.MousePointer = vbDefault
```