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

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

If you'd like to submit a tip to *Visual Basic Programmer's Journal*, please send it to User Tips, Fawcette Technical Publications, 209 Hamilton Avenue, Palo Alto, California, USA, 94301-2500. You can also fax it to 650-853-0230 or send it electronically to vbpjedit@fawcette.com or 74774.305@compuserve.com. Please include a clear explanation of what the technique does and why it's useful, and indicate if it's for VBA, VBS, VB3, VB4 16- or 32-bit, or VB5. Please limit code length to 20 lines. Don't forget to include your e-mail and mailing address. If we publish your tip, we'll pay you $25 or extend your *VBPJ* subscription by one year.

VB3, VB4 16/32, VB5

Level: Beginning

DELETING AN ARRAY ELEMENT

Conventional wisdom suggests that to delete an array element, you must move up all the subsequent elements to close the "gap" left by the deleted item. However, if the sequence of the elements isn't significant (as in an unsorted array), this algorithm quickly deletes an item:

```
' Element to delete
iDelete = 5
' Number of elements before deletion
nElements = UBound(Array)
' Replace iDelete with last item in array
Array(iDelete) = Array(nElements)
' Use ReDim Preserve to shrink array by one
ReDim Preserve Array(LBound(Array) _
    To nElements - 1)
```

—Basil Hubbard, Hamilton, Ontario, Canada

VB4 32, VB5, VBA

Level: Intermediate

INVOKE "OPEN WITH ..." DIALOG BOX

When launching a data file with the ShellExecute() function, Windows tries to find the associated application and open the data file with this application. But what happens if no association exists? ShellExecute() simply returns error code 31 (no association) and nothing happens. Wouldn't it be nice if your program invoked the "Open with ..." dialog box so you can choose which application you want to associate with your data file? Here's a solution—call the ShellDoc routine and pass a fully qualified path/file name of the data file you wish to open:

```
Option Explicit

Declare Function GetDesktopWindow Lib "user32" () As Long
Declare Function ShellExecute Lib _
    "shell32.dll" Alias "ShellExecuteA" _
    (ByVal hWnd As Long, ByVal lpOperation _
    As String, ByVal lpFile As String, _
    ByVal lpParameters As String, _
    ByVal lpDirectory As String, _
    ByVal nShowCmd As Long) As Long
Declare Function GetSystemDirectory Lib _
    "kernel32" Alias "GetSystemDirectoryA" _
    (ByVal lpBuffer As String, ByVal nSize _
    As Long) As Long
Private Const SE_ERR_NOASSOC = 31
Public Sub ShellDoc(strFile As String)
    Dim lngRet As Long
    Dim strDir As String
    lngRet = ShellExecute(GetDesktopWindow, _
        "open", strFile, _
        vbNullString, vbNullString, vbNormalFocus)
    If lngRet = SE_ERR_NOASSOC Then
        ' no association exists
        strDir = Space(260)
        lngRet = GetSystemDirectory(strDir, _
            Len(strDir))
        strDir = Left(strDir, lngRet)
        ' show the Open with dialog box
        Call ShellExecute(GetDesktopWindow, _
            vbNullString, "RUNDLL32.EXE", _
            "shell32.dll,OpenAs_RunDLL " & _
            strFile, strDir, vbNormalFocus)
    End If
End Sub
```

—Thomas Weidmann, received by e-mail

VB4 32, VB5

Level: Beginning

SSTAB VS. OPTION BUTTONS

Although VB's SSTab control behaves as if each tab page is a container, it actually uses a single container for all tab pages. This can cause unexpected behavior if you have groups of option buttons on different pages. Clicking on an option button on one page clears all the uncontained option buttons on the other, seemingly unrelated, pages. Solve this problem by adding your own containers (frames or picture boxes) for each group of options you want to be mutually exclusive.

—Steve Cisco and Roland Southard, Franklin, Tennessee

VB4 32, VB5
Level: Beginning

CHANGE THE APPEARANCE PROPERTY OF A TEXT BOX AT RUN TIME

Sorry, you can't change the Appearance property of a text box at run time—but you can make it look like you have! If set to none, a 3-D picture box has a flat BorderStyle property. Put your text box (with a flat appearance) inside a picture box (with a 3-D appearance) and change the picture box's border style. Use this complete code—be sure you place Text1 inside Picture1:

```
Private m_Text1_Appearance As Long

Private Sub Form_Load()
    With Text1
        Picture1.Width = .Width
        Picture1.Height = .Height
        .Move 0, 0
    End With
    Text1_Appearance = 1  '3D
End Sub

Public Property Let _
    Text1_Appearance(nAppearance As Long)
    With Picture1
        Select Case nAppearance
            Case 0  'Flat
                .BorderStyle = nAppearance
            Case 1  '3D
                .BorderStyle = nAppearance
        End Select
    m_Text1_Appearance = .BorderStyle
    End With
End Property

Public Property Get Text1_Appearance() As Long
    Text1_Appearance = m_Text1_Appearance
End Property
```

—Jim Deutch, Cazenovia, New York

VB4 32, VB5
Level: Beginning

DEALING WITH NULL VALUES RETURNED FROM RDO RESULTSETS

If you're assigning the values of columns you return from RDO queries into string variables, you'll get an "Invalid use of Null" error if one of the columns has a Null value. For most purposes, I'd rather have the value as an empty string anyway. Rather than code for that each time I access a column, I've written a function called Clean that turns Null values into empty strings. I call it like this:

```
strMyString=Clean(rdoResultset("MyVarCharColumn"))
```

I also convert Empty values as well, for use with Variants:

```
Public Function Clean(ByVal varData As Variant) As String

If IsNull(varData) Then
    Clean = ""
ElseIf IsEmpty(varData) Then
    Clean = ""
Else
    Clean = CStr(varData)
End If

End Function
```

—James T. Stanley, Muncie, Indiana

VB3, VB4 16/32, VB5
Level: Beginning

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

VB4 16/32, VB5
Level: Intermediate

FLOATING AN EDIT BOX

To minimize the number of controls on my forms, I use a text box as a floating input control that I either overlay onto a grid or swap with a label. Here is my swap subroutine:

```
Public Sub SwapControls(cHide As Control, _
    cShow As Control, Optional Value)
    With cHide
        .Visible = False
        cShow.Move .Left, .Top, .Width, .Height
    End With
    If IsMissing(Value) Then
        If TypeOf cShow Is TextBox Or _
            TypeOf cShow Is Label Then
                cShow = cHide
        End If
    Else
        cShow = Value
    End If

    With cShow
        .Visible = True
        .ZOrder
        If TypeOf cShow Is TextBox Then
            .SelStart = 0
```

```
            .SelLength = Len(cShow)
            If .Visible Then
                .SetFocus
            End If
        End If
    End With
End Sub
```

When I enter the statement "SwapControls lblData, txtData," lblData disappears and txtData appears in its place with the value of lblData selected and the focus set to it. After you make your entry, execute the statement "SwapControls txtData, lblData."

—Calogero S. Cumbo, Waterloo, Ontario, Canada

VB4 32, VB5

Level: Intermediate

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

VB5

Level: Intermediate

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

VB4 32, VB5

Level: Intermediate

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

VB3, VB4 16/32, VB5

Level: Intermediate

FORCE AN MDI WINDOW REFRESH

I sometimes want an MDI parent window to be repainted. For example, if a modal dialog is displayed over the MDI form and you click on OK, the dialog is hidden and an operation occurs, which takes a few seconds to complete. In the meantime, remnants of the dialog are still visible because Windows doesn't have time to complete the paint operation, and the screen looks messy. MDI forms don't have a Refresh method, and I don't want to throw a DoEvents into my code because it's dangerous. This code gives my MDI form a Refresh method:

```
Public Sub Refresh()
    Call RedrawWindow(Me.hWnd, 0&, 0&, _
        RDW_ALLCHILDREN Or RDW_UPDATENOW)
End Sub
```

You need to declare these API constants:

```
Public Const RDW_ALLCHILDREN = &H80
Public Const RDW_UPDATENOW = &H100

' Note: The data type of the lprcUpdate
' parameter has been changed
' from RECT to Any so 0& (NULL) can be passed.
#If Win32 Then
    Declare Function RedrawWindow Lib _
        "user32" (ByVal hwnd As Long, _
        lprcUpdate As Any, ByVal hrgnUpdate _
        As Long, ByVal fuRedraw As Long) As Long
#ElseIf Win16 Then
    Declare Function RedrawWindow Lib "User" _
        (ByVal hWnd As Integer, lprcUpdate As Any, _
        ByVal hrgnUpdate As Integer, ByVal fuRedraw As _
        Integer) As Integer
#End If
```

—Thomas Weiss, Buffalo Grove, Illinois

VB5

Level: Beginning

TAKE ADVANTAGE OF RELATED DOCUMENTS AREA IN PROJECT WINDOW

If you use a resource file in your application, you can see the RES file appear in the project window under "Related Documents." This is the only type of file that VB automatically adds to this node of the project tree. You can add any type of file you like to this area manually, though. From the Project menu, select Add File, or right-click on the project window and select Add File from the context menu. In the dialog box, select All Files for the file type and check the Add As Related Document option.

Adding additional related files here helps organize your project and gives you quick access to useful items, including design documents, databases, resource scripts, help-project files, and so on. Once a file has been added, double-click on it in the project window to open it with the appropriate application.

—Joe Garrick, Coon Rapids, Minnesota

VB4 32, VB5

Level: Advanced

COPY DRAWN PICTURE TO CLIPBOARD

The VB Picture control can hold several different formats of pictures: BMP, DIB, ICO, CUR, WMF, and others under VB5. Additionally, you can use graphics methods to "draw" on the control. The only native method that converts the image on the picture control, including the drawn graphics, to a bitmap and transfers the bitmap to the system clipboard requires you to use AutoRedraw.

However, this technique causes problems. This code shows the declarations and functions required to transfer the image on a VB picture control to the system clipboard as a bitmap. Add this code to a BAS module, call PicToClip, and pass the picture box as the only parameter:

```
'   #
'   # API Declarations
'   #
'   Bitmap
Private Declare Function BitBlt Lib "gdi32" _
    (ByVal hDestDC As Long, ByVal x As Long, _
    ByVal y As Long, ByVal nWidth As Long, _
    ByVal nHeight As Long, ByVal hSrcDC As _
    Long, ByVal xSrc As Long, ByVal ySrc As _
    Long, ByVal dwRop As Long) As Long
Private Declare Function _
    CreateCompatibleBitmap Lib "gdi32" (ByVal hDC As Long, _
```

```
    ByVal nWidth As Long, ByVal nHeight As Long) As Long
Private Declare Function CreateCompatibleDC _
    Lib "gdi32" (ByVal hDC As Long) As Long
Private Declare Function DeleteDC Lib _
    "gdi32" (ByVal hDC As Long) As Long
Private Declare Function GetDC Lib "user32" _
    (ByVal hWnd As Long) As Long
Private Declare Function ReleaseDC Lib "user32" _
    (ByVal hWnd As Long, ByVal hDC As Long) As Long
Private Declare Function SelectObject Lib "gdi32" _
    (ByVal hDC As Long, ByVal hObject As Long) As Long
' Clipboard
Private Declare Function OpenClipboard Lib _
    "user32" (ByVal hWnd As Long) As Long
Private Declare Function CloseClipboard Lib _
    "user32" () As Long
Private Declare Function EmptyClipboard Lib _
    "user32" () As Long
Private Declare Function SetClipboardData Lib "user32" _
    (ByVal wFormat As Long, ByVal hMem As Long) As Long
'   #
'   # API Constants
'   #
'   Clipboard formats
Private Const CF_BITMAP = 2
' ROP
Private Const SRCCOPY = &HCC0020

Public Sub PicToClip(pic As PictureBox)
    Dim hSourceDC As Long
    Dim hMemoryDC As Long
    Dim lWidth As Long
    Dim lHeight As Long
    Dim hBitmap As Long
    Dim hOldBitmap As Long
    '   #
    '   # NOTE: Error trapping has been removed
    '   for the sake of clarity
    '   #
    With pic
        ' Determine bitmap size
        lWidth = .Parent.ScaleX(.ScaleWidth, _
            .ScaleMode, vbPixels)
        lHeight = .Parent.ScaleY(.ScaleHeight, _
            .ScaleMode, vbPixels)

        ' Get hBitmap loaded with image on
        ' Picture control
        hSourceDC = GetDC(.hWnd)
        hMemoryDC = CreateCompatibleDC(.hDC)
        hBitmap = CreateCompatibleBitmap( _
            .hDC, lWidth, lHeight)
        hOldBitmap = SelectObject(hMemoryDC, _
            hBitmap)
        Call BitBlt(hMemoryDC, 0, 0, lWidth, _
            lHeight, pic.hDC, 0, 0, SRCCOPY)
        hBitmap = SelectObject(hMemoryDC, _
            hOldBitmap)

        ' Copy to clip board
        Call OpenClipboard(.Parent.hWnd)
        Call EmptyClipboard
        Call SetClipboardData(CF_BITMAP, _
            hBitmap)
        Call CloseClipboard

        ' Clean up GDI
        Call ReleaseDC(.hWnd, hSourceDC)
        Call SelectObject(hMemoryDC, hBitmap)
        Call DeleteDC(hMemoryDC)
    End With
End Sub
```

—Tom McCormick, Bedford, Massachusetts

VB4 16/32, VB5

Level: Beginning

ERASE A VARIANT ARRAY

You'll find the IsArray() function helpful when you use Variant arrays that you can set or unset through your code and need to test often. However, once you declare the array, IsArray() returns True, even if the array has been erased using the Erase keyword. To solve this, reset a Variant array by assigning zero or null, so the IsArray() function returns the proper value:

```
Dim myVar As Variant
Debug.Print IsArray(myVar) 'Returns False
ReDim myVar(0 To 5)
Debug.Print IsArray(myVar) 'Returns True
Erase myVar
Debug.Print IsArray(myVar) 'Returns True
myVar = 0
Debug.Print IsArray(myVar) 'Returns False
```

To avoid this kind of tricky problem, use an Erase subroutine like this one:

```
Public Sub vErase(ByRef pArray As Variant)
    Erase pArray
    pArray = 0
End Sub
```

—Nicolas Di Persio, Montreal, Quebec, Canada

VB5

Level: Intermediate

FORCE A SINGLE SELECT FOR A GRID

Setting the SelectionMode property of the MSFlexGrid to flexSelectionByRow forces all columns in a row to be selected rather than a single cell. It also allows selection of multiple rows simultaneously. To force a single row selection for a grid, I have a function called UpdateGrid that ensures only one row is selected, regardless of a drag on the rows or if the Shift and the Up and Down arrow keys are used. This is useful if you want to present a list of items in a grid format and only want one highlighted:

```
Sub UpdateGrid(grdInput As MSFlexGrid)
    If grdInput.Rows = (grdInput.FixedRows + 1) Then
        ' only one row in the grid and it
        ' it a fixed one: don't do anything
        Exit Sub
    Else
        ' more than one row in the grid
        If grdInput.RowSel <> grdInput.Row Then
            ' user selected a different row in the grid
            ' than the current row:
            ' set it to the highlighted row
            grdInput.RowSel = grdInput.Row
        End If
    End If
End Sub
```

In the SelChange event for a grid, put in this code:

```
Private Sub myGrid_SelChange
    UpdateGrid myGrid
End Sub
```

—Mike Peters, received by e-mail

VB4 16/32, VB5

Level: Intermediate

RAGGED ARRAYS

Who said arrays in VB can't change all dimensions while preserving data? I call this the "variable dimensions array," and I use it when applications need data arrays with more flexible sizes in all dimensions. This variable prevents your apps from having empty elements in arrays (even if the Variant data type takes a lot of memory). For example, take a look at this two-dimensional array. Instead of declaring the variables with the two dimensions from the beginning, simply declare a Variant:

```
Dim  myVar as Variant
' Then 'redim' the first dimension only (2
' elements):
redim myVar(0 to 1)
' You can now use the Array() function for
' each element of the array:
myVar(0) = Array(0, 10, 50)
myVar(1) = Array("test1", "test2", "test3", "test4")
```

Use this code to get the data:

```
myVar(0)(1) = 10
myVar(1)(2) = "test3"
```

You can use as many parentheses as you want, and you can still use the Redim Preserve statement with each element and all dimensions. Simple! Note that you can also use a subroutine to resize one element of the array if you don't want to use the Array() function:

```
Public Sub sbDeclare(ByRef pItem As Variant, _
    pLower As Integer, pUpper As Integer)
    ReDim Preserve pItem(pLower To pUpper)
End Sub

Call sbDeclare(myVar(0), 0, 1)
```

—Nicolas Di Persio, Montreal, Quebec, Canada

VB4 32, VB5

Level: Intermediate

ADJUST COMBO BOX DROP-DOWN WIDTH

Due to limited space on a form, you sometimes must keep the width of combo boxes small. Because a combo box lacks a horizontal scrollbar, some text might remain hidden. Use these functions to retrieve the current size of a drop-down and to resize the drop-down portion of the combo box as needed at run time. Add this code to a BAS module and call it from wherever convenient—perhaps during your Form_Load procedure:

```
Private Declare Function SendMessage Lib _
    "USER32" Alias "SendMessageA" _
    (ByVal hwnd As Long, ByVal Msg As Long, _
    ByVal wParam As Long, ByVal lParam As _
    Long) As Long
Private Const CB_GETDROPPEDWIDTH = &H15F
Private Const CB_SETDROPPEDWIDTH = &H160
Private Const CB_ERR = -1

Public Function GetDropdownWidth(cboHwnd As Long) As Long
    Dim lRetVal As Long
    '*** To get the combo box drop-down width.
    '*** You may use this function if you want
    '*** to change the width in proportion
    '*** i.e. double, half, 3/4 of existing width.
    lRetVal = SendMessage(cboHwnd, CB_GETDROPPEDWIDTH, 0, 0)
    If lRetVal <> CB_ERR Then
        GetDropdownWidth = lRetVal
        'Width in pixels
    Else
        GetDropdownWidth = 0
    End If
End Function

Public Function SetDropdownWidth(cboHwnd As _
    Long, NewWidthPixel As Long) As Boolean
    Dim lRetVal As Long
    ' *** To set combo box drop-down width ***
    lRetVal = SendMessage(cboHwnd, _
        CB_SETDROPPEDWIDTH, NewWidthPixel, 0)
    If lRetVal <> CB_ERR Then
        SetDropdownWidth = True
    Else
        SetDropdownWidth = False
    End If
End Function
```

—Rajeev Madnawat, Sunnyvale, California

VB3, VB4 16/32

Level: Intermediate

MESSAGEBOX ADVANTAGE

You've probably noticed that the display time stops when an application pops up VB's built-in MsgBox. Although the system timer continues to tick, the timer control isn't updated every second, nor do other events (such as painting) process. To update the timer, replace VB's built-in MsgBox with the MessageBox API function. MessageBox-generated dialogs don't stop the timer from updating, and they allow other normal processing, such as form painting:

```
' General Declarations in BAS module
Public 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

' Call from within any form like this:
Call MessageBox(Me.hWnd, _
    "This is a test in API Message Box", _
    "API Message Box", vbInformation)
```

To use this technique in VB3, declare all parameters in the API call as integer. While calling, pass MB_ICONINFORMATION as the last parameter, instead of vbInformation. You can find the constant value for MB_ICONINFORMATION in the CONSTANT.txt file. Note that many of the intrinsic VB constants used with MsgBox also work with the MessageBox API. Now for the best news about this workaround—it's *totally* unnecessary under VB5! Timer (and other) events are never blocked by a MsgBox call

when run from an EXE. It's important to understand that they'll still be blocked in the IDE, but take a look next time you compile and you'll see your clock just keeps on ticking.

—Vasudevan Sampath, San Jose, California

VB5

Level: Intermediate

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

VB3, VB4 16/32, VB5

Level: Intermediate

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

VB3, VB4 16/32, VB5

Level: Beginning

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

VB4 32, VB5

Level: Intermediate

SET THE LISTINDEX WITHOUT THE CLICK EVENT

If you set the ListIndex property of a list-box or combo-box control, VB might generate an unwanted Click event. Instead of writing code to bypass the Click event, use SendMessage to set the ListIndex without generating the event. Call the SetListIndex function below, passing the list (either a list box or combo box) and the desired new index value. SetListIndex attempts to set the value and returns the current index so you can confirm whether your request "took." For example, this code should set the index to the tenth element:

```
    Debug.Print SetListIndex(List1, 9)
```

If an error occurred (if there were only eight elements, for example), the previous index value is returned. Code the SetListIndex function in a standard module:

```
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 Function SetListIndex(lst As Control, _
    ByVal NewIndex As Long) As Long
    Const CB_GETCURSEL = &H147
    Const CB_SETCURSEL = &H14E
    Const LB_SETCURSEL = &H186
    Const LB_GETCURSEL = &H188

    If TypeOf lst Is ListBox Then
        Call SendMessage(lst.hWnd, _
            LB_SETCURSEL, NewIndex, 0&)
        SetListIndex = SendMessage(lst.hWnd, _
            LB_GETCURSEL, NewIndex, 0&)
    ElseIf TypeOf lst Is ComboBox Then
        Call SendMessage(lst.hWnd, _
            CB_SETCURSEL, NewIndex, 0&)
        SetListIndex = SendMessage(lst.hWnd, _
            CB_GETCURSEL, NewIndex, 0&)
    End If
End Function
```

—Greg Ellis, St. Louis, Missouri

VB3, VB4 16/32, VB5, VBA

Level: Intermediate

CLEANING UP AFTER A CRASH

If your app uses temporary files, store the file name(s) in the Registry as you create them. When you exit the program, delete the temporary file and its related Registry entry. However, if you shut off the machine, Windows crashes, or your program crashes, your temporary file will stay in the Registry. This leads to wasted space, and you must then delete the files from their temporary directory. Because you stored the temporary file name in the Registry, you can check for it when your program starts up again and delete it if it still exists.

—Brian Hutchison, Seattle, Washington

VB3, VB4 16/32, VB5

Level: Beginning

SEND MAIL FROM VB5

If Microsoft Outlook is installed, you have an easy way to send e-mail from VB5. To use this technique with VB3, remove the With construct and fully qualify each object property reference:

```
    Dim olapp As Object
    Dim oitem As Object

    Set olapp = CreateObject("Outlook.Application")
    Set oitem = olapp.CreateItem(0)
    With oitem
        .Subject = "VBPJ RULES"
        .To = "MONTEZUMA;other Names;"
        .Body = "This message was sent from VB5"
        .Send
    End With
```

—Jim Griffith, Montezuma, Georgia

VB3, VB4 16/32, VB5

Level: Beginning

AVOID THE FLICKERING

Developers often need to load forms with information, which is time-consuming. The form is often a list box filled from an outside source, and this causes the list-box contents to flash annoyingly as the information goes into it. Solve this by bringing in the declaration of the LockWindowUpdate API call:

```
#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
```

The hWndLock variable refers to the hWnd property of the form where you don't want to have screen updates shown. When you reissue the LockWindowUpdate with a value of 0 for hwndLock, you'll free up the screen and all updates will be shown instantly:

```
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
```

—Bruce Goldstein, Highlands Ranch, Colorado

VB4 32, VB5

Level: Intermediate

ADDING FULL PATHS TO A TREEVIEW

Have you ever wanted to add nodes to a TreeView control using a full path instead of adding a node at a time? You can do it with this code:

```
Public Sub AddPathToTree(Tree As TreeView, Path As String)
     Dim PathItem As String
     Dim NewItem As String
     Dim PathLen As Integer
     Dim c As String * 1
     Dim i As Integer

     'ADD A BACKSLASH AS A TERMINATOR
     If Right$(Path, 1) <> "\" Then Path = Path & "\"

     PathLen = Len(Path)

     'RUN THROUGH THE PATH LOOKING FOR BACKSLASHES
     For i = 1 To PathLen
          c = Mid$(Path, i, 1)
          If c = "\" Then
               If PathItem = "" Then
                    'ADD THE ROOT ITEM TO THE TREE
                    On Error Resume Next
                    Tree.Nodes.Add , , "\" & NewItem, NewItem
                    PathItem = "\" & NewItem
               Else
                    'ADD THE NEXT CHILD TO THE TREE
                    Tree.Nodes.Add PathItem, tvwChild, PathItem _
                         & "\" & NewItem, NewItem
                    PathItem = PathItem & "\" & NewItem
               End If
               NewItem = ""100
          Else
               NewItem = NewItem & c
          End If
     Next i
End Sub
```

Simply call this routine passing the TreeView control and the full path as parameters. All the necessary nodes will be added if they don't already exist:

```
     AddPathToTree TreeView1, _
          "RootLevel\Child1\Child2\Child3\Child4"
```

—Tom Stock, St. Petersburg, Florida

VB3, VB4 16/32, VB5

Level: Beginning

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

VB4 16/32, VB5
Level: Beginning

TILE AN IMAGE ONTO A FORM
Adding this code to a form causes it to tile the image stored in Picture1 across the entire form whenever the form requires a refresh:

```
Private Sub Form_Load()
    With Picture1
        .AutoSize = True
        .BorderStyle = 0
        .Visible = False
    End With
End Sub

Private Sub Form_Paint()
    Dim i As Long, j As Long
    With Picture1
        For i = 0 To Me.ScaleWidth Step .Width
            For j = 0 To Me.ScaleHeight Step .Height
                PaintPicture .Picture, i, j
            Next j
        Next i
    End With
End Sub

Private Sub Form_Resize()
    Me.Refresh
End Sub
```

—Devin Coon, Pittsburgh, Pennsylvania

VBA
Level: Beginning

ZAP EXPIRED DOCS
This VBA Microsoft Word routine purges a document when it's opened after a predefined expiration date. I've only tested this macro with Word 97:

```
Sub Purge()
    Dim ExpirationDate As Date
    ExpirationDate = #4/1/98#
    'This particular document expires on 1 April 1998
    If Date >= ExpirationDate Then
        'Purge the document
        With Selection
            .WholeStory
            .Delete Unit:=wdCharacter, Count:=1
            .TypeText Text:= _
                "This document expired on" & _
                Str(ExpirationDate) & "."
            .TypeParagraph
        End With
        ActiveDocument.Save
        'Alert the user
        MsgBoxResult = MsgBox("This document has expired. " _
            & "Please acquire an updated copy.", , _
            "Document Purge")
    End If
End Sub
```

In order to work, you should call this macro from a document's AutoOpen macro.

—Dorin Dehelean, Dollard des Ormeaux, Quebec, Canada

VB4 16/32, VB5
Level: Intermediate

QUICK CLASS TESTS
When testing properties and methods of an object that you're writing, you don't have to run a test project or form to test it. Instead, open the Immediate window and begin typing and executing code:

```
Set c = new Class1
? c.TestProperty
```

—Trey Moore, San Antonio, Texas

VB3, VB4 16/32, VB5
Level: Advanced

A QUICKER "NEXT/PREVIOUS WEEKDAY"
In the latest tips supplement ["101 Tech Tips for VB Developers," Supplement to the February 1998 issue of *VBPJ*], I noticed a tip titled "Determine Next/Previous Weekday." This code is shorter and accomplishes the same task with no DLL calls:

```
Public Function SpecificWeekday(ByVal D As Date, Optional _
    ByVal WhatDay As VbDayOfWeek = vbSaturday, _
    Optional GetNext As Boolean = True) As Date
    SpecificWeekday = (((D - WhatDay + GetNext) \ 7) - _
        GetNext) * 7 + WhatDay
End Function
```

This code averages about 10 times faster in VB3 and up to 30 times faster in VB5. It works because VB keeps dates internally as the number of days since Saturday, December 30, 1899. A date of 1 represents Sunday, December 31, 1899, which is also its own weekday. This means the WeekDay function is equivalent to the expression (Date - 1) Mod 7 + 1. This is coded for VB5, but by altering the way the Optional parameters are handled, you can make it work in either VB3 or VB4.

—Phil Parsons, Newmarket, Ontario, Canada

VB3, VB4 16/32, VB5
Level: Intermediate

REMOVE MIN/MAX BUTTONS FROM MDI FORM
Unlike other forms, MDI forms don't have MinButton and MaxButton properties to enable or disable the form's Minimize and Maximize buttons. If you add this code to an MDI parent form's Load event, it disables the Minimize and Maximize buttons on the MDI form. If you just want to disable one or the other, comment out the appropriate line, based on which constant you don't need:

```
Sub MDIForm_Load()
    Dim lWnd as Long
    lWnd = GetWindowLong(Me.hWnd, GWL_STYLE)
    lWnd = lWnd And Not (WS_MINIMIZEBOX)
    lWnd = lWnd And Not (WS_MAXIMIZEBOX)
    lWnd = SetWindowLong(Me.hWnd, GWL_STYLE, lWnd)
End Sub
```

Add this code (which includes the required API declarations)

to a BAS module:

```
#If Win32 Then
     Private Declare Function SetWindowLong Lib "user32" _
          Alias "SetWindowLongA" (ByVal hwnd As Long, ByVal _
          nIndex As Long, ByVal dwNewLong As Long) As Long
     Private Declare Function GetWindowLong Lib "user32" _
          Alias "GetWindowLongA" (ByVal hwnd As Long, ByVal _
          nIndex As Long) As Long
#Else
     Declare Function SetWindowLong Lib "User" (ByVal hwnd _
          As Integer, ByVal nIndex As Integer, ByVal _
          dwNewLong As Long) As Long
     Declare Function GetWindowLong Lib "User" (ByVal hwnd _
          As Integer, ByVal nIndex As Integer) As Long
#End If

Const WS_MINIMIZEBOX = &H20000
Const WS_MAXIMIZEBOX = &H10000
Const GWL_STYLE = (-16)
```

—Joselito Ogalesco, Morton, Pennsylvania

VB4 32, VB5

Level: Intermediate

MAKE BUTTONS APPEAR

VB doesn't display the Min and Max buttons in a form's caption area when you specify BorderStyle Fixed Dialog. If you set the MinButton and MaxButton properties on the form to True, the Minimize and Maximize entries in the context menu are visible—but the buttons are still invisible! To fix this, add this code to a standard module:

```
Private Declare Function GetWindowLong Lib "user32" Alias _
     "GetWindowLongA" (ByVal hWnd As Long, ByVal nIndex As _
     Long) As Long
Private Declare Function SetWindowLong Lib "user32" Alias _
     "SetWindowLongA" (ByVal hWnd As Long, ByVal nIndex As _
     Long, ByVal dwNewLong As Long) As Long

Private Const GWL_STYLE = (-16)
Private Const WS_MINIMIZEBOX = &H20000
Private Const WS_MAXIMIZEBOX = &H10000

Public Sub SetCaptionButtons(Frm As Form)
     Dim lRet As Long
     lRet = GetWindowLong(Frm.hWnd, GWL_STYLE)
     SetWindowLong Frm.hWnd, GWL_STYLE, lRet Or _
          WS_MINIMIZEBOX * (Abs(Frm.MinButton)) Or _
          WS_MAXIMIZEBOX * (Abs(Frm.MaxButton))
End Sub
```

You must call the subroutine SetCaptionButtons from the Form_Load event, passing a reference to your form. This should work in VB3 and VB4 16 with the proper 16-bit API declarations (see "Remove Min/Max Buttons From MDI Form").

—Geir A. Bergsløkken, Grinder, Norway

VB4 16/32, VB5

Level: Intermediate

ADD A NEW NUMBER FORMAT

A client needed the numbers to show up in certain data files in the "x100" format to accommodate interchanging data with a legacy system. That is, if the number is "23.56," it shows up as "2356," and "23" becomes "2300." Because I didn't want to create a special case throughout my code to manage this, and the VB Format function doesn't support such a format, I subclassed the Format function and added the new format myself:

```
Public Function Format(Expression As Variant, Optional _
     sFormat As Variant, Optional FirstDayOfWeek As _
     Variant, Optional FirstWeekOfYear As Variant) As String
     If IsMissing(sFormat) Then
          Format = VBA.Format(Expression)
     ElseIf sFormat = "x100" Then
          ' handle the special x100 case
          Expression = Expression * 100
          Format = VBA.Format(Expression, "0.")
          Format = Left$(Format, InStr(1, Format, ".") - 1)
     Else
          ' wasn't my special format, so pass through to the
          ' real format function
          If IsMissing(FirstWeekOfYear) And _
               IsMissing(FirstDayOfWeek) Then
               Format = VBA.Format(Expression:=Expression, _
                    Format:=sFormat)
          ElseIf IsMissing(FirstDayOfWeek) Then
               Format = VBA.Format(Expression:=Expression, _
                    FirstWeekOfYear:=FirstWeekOfYear)
          ElseIf IsMissing(FirstWeekOfYear) Then
               Format = VBA.Format(Expression:=Expression, _
                    FirstDayOfWeek:=FirstDayOfWeek)
          End If
     End If
End Function
```

This allows me to simply call the Format function as I normally would everywhere in my code, have my "x100" format, and still support all the normal Format parameters and options. Note the use of VBA.Format in the routine to reference the built-in format function.

—Jon Pulsipher, Bellevue, Washington

VB5

Level: Intermediate

LIVE ACTION CAPTIONS

When building a TextBox or Label UserControl, the Caption or Text property sometimes doesn't work well with the standard controls, as the control's appearance doesn't change when you type the value into the Properties window.

To make your UserControl behave the way you want, go to the UserControl's code window and select Procedure Attributes from the Tools menu. Find your Caption or Text property on the combo box and click on Advanced. On the Procedure ID combo box, select Text. Test it by putting an instance of your control on a form and changing the Text or Caption property at design time. The Property Let procedure will fire with each keystroke, allowing your update routine to reflect what the user has typed.

—Leonardo Bosi, Buenos Aires, Argentina

VB5
Level: Intermediate

ISMISSING BEHAVIOR CHANGED IN VB5
In VB5, you can assign a default value to a typed optional argument. But you must then use the IsMissing function carefully, because when the optional argument is typed, IsMissing always returns False. Only when using an untyped (Variant) optional argument will IsMissing be accurate in determining whether a value was passed. If no default value is assigned and the argument is typed, VB automatically assigns the default value normally assigned to such a type—typically 0 or an empty string.

Under this condition, you shouldn't use IsMissing to detect whether the argument has been set. You can detect it with two methods. The first method is to not give the argument a type when you declare, so you can use the IsMissing function to detect it. The second method is to give a default value when you declare, but you won't have to set that value when you call it. This code gives some examples about using optional arguments and the IsMissing function:

```
Private Sub fun1(..., Optional nVal)
'- Without type (Variant)
     ...
     If IsMissing(nVal) Then
     '- You can use IsMissing here
     Else
     End If
End Sub

Private Sub fun2(..., Optional nVal As Integer)
'- With type but no default value
     ...
     If IsMissing(nVal) Then
          '- You cannot use IsMissing here to detect is
          '- nVal been set, always return true
          '- VB will give nVal a default value 0 because
          '- its type is Integer
     End If

End Sub

Private Sub fun3(..., Optional nVal As Integer = -1)
     If nVal = -1 Then
     '- You can use this to detect , in function equals to
     '- IsMissing
     '- But you must sure the the value -1 will not be
     '- used when the procedure is called
     Else
     End If
End Sub
```

—Henry Jia, received by e-mail

VB3, VB4 16/32, VB5
Level: Beginning

SET DEFAULT FONT FOR NEW CONTROLS
When you place controls on a form, the Font properties of all the controls default to a sans serif font rather than a default font that you specify. To avoid this annoyance, set the Font property for the form to the value you'd like the controls to use *before* placing controls on the form.

—Trey Moore, San Antonio, Texas

VB3, VB4 16/32, VB5
Level: Intermediate

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

VB3, VB4 16/32, VB5
Level: Intermediate

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

VB4 16/32, VB5

Level: Intermediate

USE OLE AUTOMATION TO CALL 16-BIT DLLS FROM VB5 (OR VB4 32)

First, create a VB4-16 project and call it VB16Project. In the project, create a class module that includes a function that calls your 16-bit DLL in the usual way. Name the class VB16Class and the function VB16DLLCall. In your VB4-16 project, remove the default form and add a standard module with a Main subroutine or function.

Under the Tools/Options/Project/StartMode section, click on the OLE Server radio button. Make sure your class module's Instancing Property is set to 1 (CreateableSingleUse) or 2 (CreateableMultiUse). Compile and save your project as an Executable. Now create a 32-bit application using VB4-32 or VB5. To call the 16-bit DLL in your 32-bit project, use this code:

```
Dim MyObj as Object
Set MyObj = CreateObject("VB16Project.VB16Class")
Call MyObj.VB16DLLCall()
```

—Jim Miles, Thousand Oaks, California

VB4 32, VB5

Level: Advanced

YEAH, BUT WHICH COMMON CONTROLS?

This fragment of code from the VB standard module shows the GetComCtlVersion function that retrieves the major and minor version numbers of the Comctl32.dll installed on the local system. Use this function when you subclass toolbar or listview controls from Comctl32.ocx and implement hot-tracking toolbar or full-row select in the listview. It's also useful when checking the DLL version in your setup application:

```
VersionDistribution Platform
4.00 Microsoft Windows 95/Windows NT 4.0
4.70 Microsoft Internet Explorer 3.0x
4.71 Microsoft Internet Explorer 4.00
4.72 Microsoft Internet Explorer 4.01

Option Explicit

Private Const S_OK = &H0
Private Declare Function LoadLibrary Lib "kernel32" _
    Alias "LoadLibraryA" (ByVal lpLibFileName As String) _
    As Long
Private Declare Function GetProcAddress Lib "kernel32" _
    (ByVal hModule As Long, ByVal lpProcName As String) As _
    Long
Private Declare Function FreeLibrary Lib "kernel32" ( _
    ByVal hLibModule As Long) As Long
Private Declare Function DllGetVersion Lib "comctl32.dll" _
    (pdvi As DLLVERSIONINFO) As Long

Private Type DLLVERSIONINFO
    cbSize As Long
    dwMajorVersion As Long
    dwMinorVersion As Long
    dwBuildNumber As Long
    dwPlatformID As Long
End Type

Public Function GetComCtlVersion(nMajor As Long, nMinor As _
    Long) As Boolean
    Dim hComCtl As Long
    Dim hResult As Long
    Dim pDllGetVersion As Long
    Dim dvi As DLLVERSIONINFO

    hComCtl = LoadLibrary("comctl32.dll")
    If hComCtl <> 0 Then
    hResult = S_OK
    pDllGetVersion = GetProcAddress(hComCtl, _
        "DllGetVersion")
        If pDllGetVersion <> 0 Then
            dvi.cbSize = Len(dvi)
            hResult = DllGetVersion(dvi)
        If hResult = S_OK Then
                nMajor = dvi.dwMajorVersion
                nMinor = dvi.dwMinorVersion
            End If
        End If
        Call FreeLibrary(hComCtl)
        GetComCtlVersion = True
    End If
End Function
```

—Lubomir Bruha, Czech Republic

VB3, VB4 16/32, VB5

Level: Beginning

DISABLE EASILY

You can easily give your check-box control a Locked property without making your own custom control. First, create a frame large enough to contain your check boxes. Clear the caption and set the border style to None. Put as many check boxes as needed on this frame, setting their captions and so on. When you're done, set the frame's Enabled property to False. You can use the same trick to make other controls, such as option buttons and text boxes, appear enabled but not respond.

—Dexter Jones, received by e-mail

VB3, VB4 16/32, VB5
Level: Intermediate

DETERMINE LIST ITEM BY COORDINATES
I wanted users to be able to get a definition for each item in a list box by right-clicking on the item. Unfortunately, right-clicking doesn't automatically select the item, so you need some other way of knowing your location in the list box. Simply reading the Y value from the MouseDown event and converting that value to a line number will work, unless the user scrolls the list. I used the SendMessage API, which can get information from controls, to solve the problem. Here's the code:

```
        'Declarations section
        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
Const LB_GETITEMHEIGHT = &H1A1
'value to get height of one line in listbox

        'listbox code
Private Sub List1_MouseDown(Button As Integer, Shift As _
    Integer, X As Single, Y As Single)
    Dim msg As String
    Dim TopIndex As Long
    Dim CharHeight As Long
    Dim CurIndex As Long

    With List1
        'find height of one line in listbox
        CharHeight = SendMessage(.hWnd, LB_GETITEMHEIGHT, _
            0, 0)
        'function returns height in pixels so convert to
        'twips
        CharHeight = CharHeight * Screen.TwipsPerPixelY

        If Button = 2 Then 'right click
            'find index number of item that received right
            'click
            CurIndex = Y \ CharHeight + .TopIndex
            'If index number is valid then display information
            If CurIndex < .ListCount Then
                'Code to retrieve and display item definition
                'goes here. Mine looks like this.
                msg = GetMessage(.List(CurIndex))
                frmInfo.Label1.Caption = msg
                frmInfo.Show
            End If
        End If
    End With
End Sub
```

If the items in your list box are always displayed in the same order and no deletions occur, then all you need is an array of definitions that correspond to each item in the list box. To retrieve the appropriate definition, use this code:

```
    msg = DefinitionArray(CurIndex)
```

After displaying the definition in a window, you only need code for the list-box MouseUp event:

```
    If Button = 2 Then frmInfo.Hide
```

—Kevin W. Williams, Oklahoma City, Oklahoma

VB3, VB4 16/32, VB5
Level: Intermediate

CUSTOM MENU ACCELERATORS
To set the shortcut key of a menu item to something other than what the VB menu editor displays, use this code in the Form_Load event of a form:

```
    Private Sub Form_Load()
        mnuExit.Caption = mnuExit.Caption & vbTab & "ALT+F4"
    End Sub
```

This adds the text "ALT+F4" to the caption of the mnuExit menu item and right-justifies it with any other shortcuts on the menu. ALT+F4 is already supported by Windows to close a window, so this shortcut needs no additional code for an exit menu choice. If you add shortcuts that Windows doesn't internally support, then set the KeyPreview property of the form to True and check the KeyUp event on the form to see if the shortcut was selected.

—Dave Kinsman, Renton, Washington

VB3, VB4 16/32, VB5
Level: Beginning

MENU PROPERTIES SHORTCUT
You can set the properties of any menu item by selecting the menu item in the Properties window drop-down list. This is often faster than selecting the Menu Editor menu choice and has the added benefit of showing you the changes to the menu choice immediately. It's also the only way to access the Tag property of the menu item at design time.

—Dave Kinsman, Renton, Washington

VB5
Level: Beginning

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

VB5
Level: Intermediate

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

VB5
Level: Intermediate

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

VB5
Level: Intermediate

HIDE ENUMERATIONS FOR VALIDATION

VB5 introduced support for enumerations, which are related sets of constants. Although you can declare a property as an enumerated type, VB lets you assign any long integer value to the property. Were you able to determine the lowest and highest values in the enumeration, you could easily validate an enumerated property, but the language doesn't support it. Instead, you can inspect the minimum and maximum values by creating hidden members in the enumeration:

```
Public Enum MyAttitudeEnum
    [_maMin] = 1
    maHappy = 1
    maSad = 2
    maIndifferent = 3
    [_maMax] = 3
End Enum
```

Inspecting the values of [_maMin] and [_maMax] makes it easy for you to validate against an enumerated type.

—Jeffrey McManus, San Francisco, California

VB5
Level: Beginning

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

VB5
Level: Beginning

FIND YOUR CONSTANT OR ENUM VALUES

I use constants for things like control-array index numbers, but it's a hassle to keep scrolling to the top of the module to remember the constant names. If you name all the constants with the same first few letters, you can use the new IntelliSense features of VB5 to obtain the list any time you need it. Simply type in the first few letters and press Ctrl+Space. A selection list then appears.

—Deborah Kurata, Pleasanton, California

VB3, VB4 16/32, VB5
Level: Beginning

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

VB5
Level: Beginning

USE THE SAME NAME FOR YOUR ERROR HANDLERS

Older versions of VB required a unique name for your error-handler labels in order to use On Error GoTo <label>. You had to concatenate the module name and routine name to ensure a unique error-handler label. This is no longer true. You can now use something standard, such as ERR_HANDLER, to identify every error handler. This makes them easier to find when reviewing your error handling.

—Deborah Kurata, Pleasanton, California

VB5
Level: Beginning

VIEW THE NAMES OF YOUR DATABASE FIELDS DIRECTLY FROM THE IDE

When developing code that maps to database fields, you often need to look back at the tables to determine the correct database fields. With VB5, you can now do this without leaving the comfort of the Integrated Development Environment (IDE). Start by setting a breakpoint after the code that populates the recordset, then run the application. When VB breaks at that line, drag your recordset object variable onto the Watches window, or open the Locals window. Click on the plus sign to open the recordset. You can look at any recordset properties, including the field names.

—Deborah Kurata, Pleasanton, California

VB3, VB4 16/32, VB5
Level: Beginning

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

VB3, VB4 16/32, VB5
Level: Beginning

PUT YOUR CHECK-BOX VALUE INTO YOUR DATABASE

Jeremy Boschen pointed out an easy way to load a Boolean into a check-box control in the tip, "Use Boolean Variables for Check-Box Values" [101 Tech Tips for VB Developers, Supplement to the February 1998 issue of *VBPJ*, page 1]. Conversely, you might want to put the value of the check box into a database as a number. To do so, use this code:

```
!db_field = Abs(Check1.Value = vbChecked)
```

This puts 1 into your database field. To make the value -1, change the Abs in the line to CInt. For VB3, you can't use the constant vbChecked and must use the value 1.

—Joe Karbowski, Traverse City, Michigan

VB3, VB4 16/32, VB5
Level: Intermediate

BE CAREFUL WHEN MIMICKING TOOL-TIP HELP

Be careful about tips to easily duplicate tool-tip help with only tip control and mouse events. If your "tip" control doesn't have the same parent as the control you're moving over, you'll put your tip control in the wrong spot! The coordinates for a control refer to its parent, so trying to position it against the wrong parent (as opposed to the form) results in an (x, y) position different than you want. Also, make sure your tip control has the highest z-order, or it might show up "behind" a nearby control. Lastly,

be careful about relying on the Form_MouseMove event to "turn off" your tip, because the event might not get fired as you move between two frames or move quickly over the form.

—Joe Karbowski, Traverse City, Michigan

VB4 32, VB5

Level: Beginning

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

VB3, VB4 16/32, VB5

Level: Beginning

USE REFRESH, NOT DOEVENTS

When executing code and tying up the system, developers often use a label or status bar to display messages. If you simply assign your text or message to the control (for example, lblMsg.Caption = "Still working…"), you won't see the text because your code loop isn't allowing the form to respond to the message. To make the message visible, use the Refresh method of the control. Don't use the DoEvents command to refresh the text to the user—this introduces re-entrancy issues. Note that displaying messages slows down performance, so use them intelligently:

```
Private Sub Command1_Click()
    Dim J As Long
    For J = 1 To 1000
        Label1.Caption = "Message " & J
        Label1.Refresh
    Next J
End Sub
```

—Joe Karbowski, Traverse City, Michigan

VB3, VB4 16/32, VB5

Level: Beginning

FORM-LEVEL VARIABLES WON'T CLEAR

When you use multiple forms in a project, make sure you explicitly set a form to Nothing after you unload it. If you don't, simply unloading the form won't necessarily clear out variables from the form. Setting it to Nothing does reset form-level variables:

```
Private Sub ShowNewForm()
    Load Form2
    Form2.Show vbModal
    Unload Form2
    Set Form2 = Nothing
End Sub
```

To see how the problem occurs, create a new standard executable project, with a form and a command button. Use this code:

```
Option Explicit
Private Sub Command1_Click()
    Load Form2
    Form2.Show vbModal
    Unload Form2
    'Set Form2 = Nothing
End Sub
```

Add a second form with a label and a command button on it, and paste in this code:

```
Option Explicit
Private msStuff As String
Private Sub Command1_Click()
    Hide
End Sub
Private Sub Form_Load()
    Label1.Caption = "value is " & msStuff
End Sub
Private Sub Form_Unload(Cancel As Integer)
    msStuff = "hey!"
End Sub
```

Press the command button on Form 1 to show Form 2. The label control shows that the msStuff variable is empty. Hide Form 2 by pressing the button, then pressing the button on Form 1 again. This time, Form 2 will have a value in the msStuff variable, showing that it doesn't clear out.

—Joe Karbowski, Traverse City, Michigan

VB3, VB4 16/32, VB5

Level: Beginning

FORMS NOT SHOWING UP IN TASKBAR

In VB3 you can set up an executable project to start up in the main subroutine, and it shows up in the Windows 95 taskbar:

```
Public Sub Main()

    Load frmFoo
    frmFoo.Show 1
    Unload frmFoo
End Sub
```

However, if you show a form modally in VB5, no matter if it's the first form in the program or not, it won't show up in the taskbar. If you want to see the item in the taskbar, you must show it nonmodally.

—Joe Karbowski, Traverse City, Michigan

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

Level: All

RTFM

Read The Fawcette Magazines.

***—VBPJ* Staff**

VB3, VB4 16/32, VB5

Level: Beginning

DON'T FORGET THAT MOUSE POINTER

If you turn the mouse pointer into an hourglass (and back to normal) in a routine with error handling, don't forget to turn the mouse pointer back to normal in the error-handler section. Otherwise, the program might look busy, but actually be done:

```
Private Function CalcTotal() As Long
    On Error GoTo ProcErr

    Screen.MousePointer = vbHourglass
    '
    'Code that may raise error
    '
    Screen.MousePointer = vbNormal
    Exit Function 'Don't go into error handler

ProcErr:
    Screen.MousePointer = vbNormal
    '
    'Error handling code
    '
End Function
```

—Joe Karbowski, Traverse City, Michigan

VB4 32, VB5

Level: Beginning

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

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

Level: Beginning

SIMPLIFY BOOLEAN VARIABLE UPDATES

Instead of using an If construct to set a Boolean variable, you can assign a Boolean variable to the result of any logical comparison. For example, instead of this code:

```
If MyNumber > 32 Then
    BooleanVar = True
Else
    BooleanVar = False
End If
```

Use this code:

```
BooleanVar = (MyNumber > 32)
```

—Joe Karbowski, Traverse City, Michigan

VB3, VB4 16/32, VB5

Level: Intermediate

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

VB4 32, VB5

Level: Intermediate

LIMIT SELECTED CHECK BOXES WITH COLLECTION LOGIC

Use the Count property to determine exactly how many controls are loaded. You can also use the For Each loop to perform code on each control. For instance, if you have a control array of check boxes and you don't want more than three checked, use this code:

```
Private Sub Check1_Click(Index As Integer)
    Dim chk As CheckBox
    Dim lCnt As Long
    Const cMAX = 3

    For Each chk In Check1
        If chk.Value = vbChecked Then
            lCnt = lCnt + 1
            If lCnt > cMAX Then
                Check1(Index).Value = vbUnchecked
                MsgBox "Too many checked!"
                Exit For
            End If
        End If

    Next chk

End Sub
```

—Joe Karbowski, Traverse City, Michigan

VB4 32, VB5

Level: Beginning

NUMERIC CONVERSION OF STRINGS

When dealing with numerics and strings, be advised of a couple "gotchas." The Val() function isn't internationally aware and will cause problems if you have users overseas. But you can't just blindly switch to CLng, CInt, and so on, which *are* internationally aware. These functions don't support an empty string (vbNullString or "") or strings that fail the IsNumeric test and will raise an error if used as such. Consider wrapping your own function around these calls to check for an empty text string before converting:

```
Public Function CInt(IntValue as Variant) as Integer
    If IsNumeric(IntValue) Then
        MyCInt = CInt(IntValue)
    Else
        MyCInt = 0
    End If
End Function
```

—Joe Karbowski, Traverse City, Michigan

VB3, VB4 16/32, VB5

Level: Intermediate

SQL TRICK TO JOIN MULTIPLE SELECT STATEMENTS

Don't overlook the UNION keyword in SQL as a way to simplify selections from multiple tables. For instance, to select the customer with the highest sales from three tables with basically the same layout, use the UNION keyword to allow your VB code to open only one resultset with the answer:

```
Private Function MaxCustSales() As Long
    Dim sSql as string

    sSql =      "select max(cust_sales) max_sales from " & _
                "sales.dbo.sales_east " & "UNION " & _
                "select max(cust_sales) max_sales from " & _
                "sales.dbo.sales_west " & _"UNION " & _
                "select max(cust_sales) max_sales from " & _
                "sales.dbo.sales_intl " & _
                "ORDER BY max_sales DESC"
```

Do this to open the resultset:

```
    If NOT IsNull(!max_sales) Then
            MaxCustSales = !max_sales
    End If

End Function
```

—Joe Karbowski Traverse City, Michigan

VB5

Level: Intermediate

DEBUGGER ISN'T INVOKED

When working on developing or debugging an app created by automation (for example, by ActiveX DLL), if you normally invoke that app from a calling program (EXE), leave the calling program running and run the app you're debugging in the VB Integrated Development Environment (IDE)—gotcha! Your debug and breakpoints won't get hit. The calling program still has the "real" automation project in memory and isn't using the version from the VB IDE. You must close down the calling program that's running and restart it to use your VB IDE version. You'll see an example of this when you try to make your automation project into a DLL and it balks, saying "Permission Denied." You need to close down any calling programs.

—Joe Karbowski, Traverse City, Michigan

VB4 32, VB5

Level: Intermediate

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

VB5

Level: Intermediate

VB HIJACKS THE REGISTRY

Be aware that VB's Integrated Development Environment (IDE) hijacks the Registry settings for your public classes when working on projects such as ActiveX DLLs and Controls. The IDE temporarily replaces (in the Registry) the existing InProcServer32 for your class to a LocalServer32 entry pointing to the VB IDE version being run. Should VB crash, that Registry entry won't return to its proper state. Then, when you try to run your program "normally," you'll get various messages that the item can't run in multi-instance mode—or other cryptic errors. You must restart the project inside the VB IDE and stop it again.

—Joe Karbowski, Traverse City, Michigan

VB5

Level: Beginning

CONSISTENT PROJECT DESCRIPTIONS

Always set the Description property of your ActiveX projects (found in Properties) to be prefixed with your company name or initials. That way, all your internal objects and components will be grouped together alphabetically, and you won't have to search for them in the list.

—Joe Karbowski, Traverse City, Michigan

VB5

Level: Advanced

ROLL YOUR OWN

If you roll your own controls in VB5 to support database applications, consider putting a Valid property, a Validation event, and a Required property on your controls. The Required property helps you determine whether a text box (for example) can be left blank, and it updates the Valid property. The Validation event should be fired by your control, allowing the developers to put their custom checks or links to the business-rules layer. Then, the developers can set the Valid property of your control accordingly. At the appropriate time, developers can check a control's Valid property to see if they can continue.

—Joe Karbowski, Traverse City, Michigan

VB4 32, VB5

Level: Intermediate

USE RDO TO ACCESS STORED FUNCTIONS AND PROCEDURES ON A SERVER

This code illustrates a VB5 routine that calls a given server's stored functions or procedures. The first parameter is the stored function procedure name that resides on the server (ORACLE, SQL Server, and so on). The second parameter is a dynamic array that takes an arbitrary number of input arguments for the stored function or procedure. It returns data from the server:

```
Public db As rdoEngine
Public en As rdoEnvironment
Public cn1 As rdoConnection

Public Function Get_STOREDFUN(sFun As String, ParamArray _
    sColumns() As Variant) As Variant [rdoResultset]

    Dim sSQL As String
    Dim Rs As rdoResultset
    Dim Qry As rdoQuery
    Dim X As Integer

    sSQL = "{ ? = Call " & sFun
    If UBound(sColumns) = -1 Then
        'Do Nothing here
    Else
            sSQL = sSQL & " ("
        For X = 0 To UBound(sColumns)
            sSQL = sSQL & "?,"
        Next
            sSQL = Left(sSQL, Len(sSQL) - 1) & ")"
    End If

    sSQL = sSQL & " }"

    Set Qry = cn1.CreateQuery("doFunction", sSQL)
    Qry(0).Direction = rdParamReturnValue

    For X = 0 To UBound(sColumns)
        Qry(X + 1).Direction = rdParamInput
        Qry.rdoParameters(X + 1) = sColumns(X)
    Next

    Set Rs = Qry.OpenResultset(rdOpenForwardOnly, _
        rdConcurReadOnly)

    Get_STOREDFUN = Qry(0)
[Set Get_STOREDFUN = Rs]

End Function
```

If you have three stored functions in a server, each one takes a different number of input arguments. You can call the same VB5 routine to get returning data:

```
    sPrdPlant  = Get_STOREDFUN("ZIP_2PLANT", CStr(txtZip))
    sControl   = Get_STOREDFUN("CONTRNUM")
    fItemPrice = Get_STOREDFUN("GET_UnitPrice", Cstr(prd), _
        Clng(qty))
```

—Kevin Shieh, Milton, Washington

VB4 32, VB5

Level: Intermediate

USE OLE AUTOMATION TO PRINT ACCESS REPORTS

You can print canned reports in an Access database from VB in different ways. I created this routine to print any report with any criteria or filter from any database. It opens Access with the user- or program-controlled database, then opens the report in the mode specified. In order for this to work with Access constants, you must select Access from the References dialog box (accessed from References under the Tools menu in VB4 and under the Project menu in VB5).

The PrintReport subroutine has the same parameters as the

Access Docmd.OpenReport command with the exception of DBPath. I didn't use IsMissing to test for missing parameters because Access handles it for you, but you can add it to supply your own default values for any missing parameters. Note: In VB4, any optional parameter must be a Variant data type. When you want to print a report from code, call the routine like this:

```
    PrintReport MyDBPath, MyReportName, acPreview, _
        MyCriteria

Sub PrintReport(ByVal DBPath As String, ByVal ReportName _
    As String
Optional OpenMode As Integer, Optional Filter As String, _
    Optional
Criteria As String)

    Dim appAccess As Object
    Set appAccess = CreateObject("Access.Application")
    appAccess.OpenCurrentDatabase (DBPath)

'*********************************************************
    'Access constants for OpenMode are
    'acNormal  - Print (default)
    'acPreview - Print Preview
    'acDesign  - Design Edit Mode
'*********************************************************
    appAccess.DoCmd.OpenReport ReportName, OpenMode, _
        FilterName
Criteria

'*********************************************************
    'if open mode is Preview then don't quit Access this can
    'also be deleted if you do not want Access to quit after
    'printing a report
'*********************************************************
    If OpenMode <> acPreview Then
        appAccess.Quit
    End If
    Set appAccess = Nothing

End Sub
```

—James Kahl, St. Louis Park, Minnesota

VB4 32, VB5

Level: Intermediate

FIND OUT WHO IS CONNECTED TO AN ACCESS DATABASE

If you're creating a Jet-based multiuser database system, you'll sometimes need to know who is currently connected to the shared database. To get this information in situations where you don't want to integrate a full Access security system, you have two choices. First, you can code in your own "connected users" table and require users to log on with a simple form on startup. Second, and better, you can use the simple msldbusr.dll. This is probably the best "power toy" ever created for multiuser Jet developers. It tells you the computer names connected by accessing the LDB of the database file.You can rename the MDB with any extension, but the LDB is what counts and it works fine.

Once you get the data from the DLL, you can integrate it with Mauro Mariz's tip, "Send Messages to WinPopUp from Your Application" [101 Tech Tips for VB Developers, Supplement to the February 1998 issue of *VBPJ*, page 18] to tell the remote user to finish up and shut down the remote app so you can perform maintenance with an exclusive connection.

If you want to integrate your own security system based on connected users without forcing manual logons, you can let the remote apps connect, then run the connections against a list of allowed computer names and take action before allowing them full access. Although this DLL is listed as "currently unsupported," it does only what it's supposed to. You can get it, along with a few other Jet locking utilities, at http://support.microsoft.com/support/kb/articles/q176/6/70.asp.

—Robert Smith, San Francisco, California

VB4 32, VB5

Level: Intermediate

PERFORM SOME COMMON DATABASE CHORES

These several database functions work together and perform various utility functions, such as checking if fields and tables exist, creating fields and tables, and so on. The interface hides all the code and returns True or False to report the status of the functions:

```
Function CreateDatabase(DatabasePath As String, dbLanguage _
    As String, JetVersion As Integer) As Boolean
    Dim TempWs As Workspace
    Dim TempDB As Database

    On Error GoTo Errors:
        Set TempWs = DBEngine.Workspaces(0)
        Set TempDB = TempWs.CreateDatabase(DatabasePath, _
            dbLanguage, JetVersion)
        CreateDatabase = True
    Exit Function
Errors:
    CreateDatabase = False
End Function

Function CreateTable(DatabasePath As String, NewTableName _
    As String) As Boolean
    Dim dbsTarget As Database
    Dim tdfNew As TableDef

    On Error GoTo Errors:
        If TableExists(DatabasePath, NewTableName) = False _
            Then
            'This table does not exist on the target
            'database, so it is ok to add it.
            Set dbsTarget = OpenDatabase(DatabasePath)
            Set tdfNew = _
                dbsTarget.CreateTableDef(NewTableName)
            With tdfNew
                .Fields.Append .CreateField("Temp", dbInteger)
            End With

            'The new table has been created, append it to the
            'database
            dbsTarget.TableDefs.Append tdfNew
            dbsTarget.TableDefs(NewTableName).Fields. _
                Delete ("Temp")
            dbsTarget.Close

            CreateTable = True
        Else
            'This table does exist on the target
            'database, so do not add it.
        End If
```

```
    Exit Function
Errors:
   CreateTable = False
End Function

Function CreateField(DatabasePath As String, _
    TargetTableName As String, NewFieldName As String, _
    FieldDataType As Integer) As Boolean
    Dim dbsTarget As Database
    Dim tdfTarget As TableDef

    On Error GoTo Errors:
        CreateField = False
        Set dbsTarget = OpenDatabase(DatabasePath)
        If TableExists(DatabasePath, TargetTableName) Then
            'The table exists, assign the table to the
            'tabledef and proceed.
            Set tdfTarget = _
                dbsTarget.TableDefs(TargetTableName)
            If Not FieldExists(DatabasePath, _
                TargetTableName, NewFieldName) Then
                'The Field doesn't exist, so create it.
                With tdfTarget
                    .Fields.Append _
                        .CreateField(NewFieldName, _
                        FieldDataType)
                End With
                CreateField = True
            Else
                'Field exists, we cannot create it.
            End If
        Else
            'The table does not exist, so we cannot add a new
            'field to it.
        End If
    Exit Function
Errors:
    CreateField = False
End Function

Function TableExists(DatabasePath As String, TableName As _
    String) As Boolean
    Dim dbsSource As Database
    Dim tdfCheck As TableDef

    On Error GoTo Errors:
        TableExists = False
        Set dbsSource = OpenDatabase(DatabasePath)
        With dbsSource
            ' Enumerate TableDefs collection.
            For Each tdfCheck In .TableDefs
                If tdfCheck.Name = TableName Then
                    TableExists = True
                    Exit For
                Else
            End If
            Next tdfCheck
        End With
    Exit Function
Errors:
   TableExists = False
End Function

Function FieldExists(DatabasePath As String, TableName As _
    String, FieldName As String) As Boolean
    Dim dbsSource As Database
    Dim tdfSource As TableDef
    Dim fldCheck As Field

    On Error GoTo Errors:
        FieldExists = False
        If TableExists(DatabasePath, TableName) Then
            Set dbsSource = OpenDatabase(DatabasePath)
            Set tdfSource = dbsSource.TableDefs(TableName)
            With tdfSource
                ' Enumerate TableDefs collection.
                For Each fldCheck In .Fields
                    If fldCheck.Name = FieldName Then
                        FieldExists = True
                        Exit For
                    End If
                Next fldCheck
            End With
        Else
            'The Table doesn't exist, so neither
            'can the field.
            FieldExists = False
        End If
    Exit Function
Errors:
    FieldExists = False
End Function
```

If you do frequent lookups, it's more productive to open and close your database externally to the functions. Because this code opens and closes the database each time, it's not meant for intensive or constant calling.

—Marc Mercuri, Somerville, Massachusetts

VB5

Level: Beginning

Password Protect an Access Database

For simple Microsoft Access security, set the database password from the Security item under the Tools menu in Access, select Set Database Password, and enter a password. To use the database in VB, pass a value along with the "pwd" keyword to the SOURCE value of the OpenDatabase method:

```
Dim db as Database
Dim Wkspc as WorkSpaces
Dim strPass as STRING

strPass = ";pwd=PASSWORD"

Set Wkspc = Workspaces(0)
Set db = Wkspc.OpenDatabase(DBName, False, False, strPass)
```

—Danny Valentino, Brampton, Ontario, Canada

VB4 32, VB5

Level: Beginning

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

VB4 32, VB5

Level: Intermediate

MAKE SURE ALL ACCESS QUERYDEF OBJECTS ARE CLOSED

When you're working with QueryDef (SQL instructions stored on an MDB database) and open it, DAO loads all the QueryDefs. For example, if you have an MDB with five QueryDefs named qryCustomers, qryOrders, qryContacts, qrySales, and qryPersons, and you want to use the qryCustomers, do this:

```
Dim qdCustomer as QueryDef
Dim rsCustomer as RecordSet

     Set qdCustomer= Db.QueryDefs("qryCustomers")
     qdCustomer.Parameters![Custom ID]= 195
     Set rsCustomer= qdCustomer.OpenRecordSet(dbReadOnly)
          While not rsCustomer.Eof
               txtCustomerName= rsCustomer!Name
                    ...........
               rsCustomer.MoveNext
          Wend

          rsCustomer.Close 'Close it
          set rsCustomer=Nothing
          'Free the reference to rsCustomer

          qdCustomer.Close 'Close it
          set qdCustomer =  Nothing
          'Free the reference to qdCustomer
```

The problem is that DAO only closes the qdCustomer, but the other four QueryDefs (qryOrders, qryContacts, qrySales, and qryPersons) remain open. To solve the problem, use this subroutine:

```
Public Sub ToNothing()
Dim qdGeneric as QueryDef

     'Surf the QueryDefs Collection
     For each qdGeneric in Db.QueryDefs
          qdGeneric.close 'Close it
          Set qdGeneric = Nothing
     Next

End Sub
```

Now put the call to the subroutine ToNothing:

```
     .
     .
     .
rsCustomer.Close
Set rsCustomer = Nothing
ToNothing
```

—Gonzalo Medina Galup, Miami, Florida

VB4 32, VB5

Level: Intermediate

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

VB3, VB4 16/32, VB5

Level: Intermediate

DECLARE YOUR OBJECTS PROPERLY

Never declare an Object variable as New. If you do, you'll always increment the reference count of the object, regardless of whether you use it. Also, remember to set your objects to Nothing when finished. For instance, instead of this way:

```
Private Sub Foo()
     Dim oCust as New clsCust
     'do stuff with oCust

End Sub
```

Do it this way:

```
Private Sub Foo()
     Dim oCust as clsCust
     Set oCust = New clsCust
     'do stuff with oCust
```

```
    Set oCust = Nothing

End Sub
```

—Joe Karbowski, Traverse City, Michigan

VB4 32, VB5

Level: Beginning

USE THE OBJECT LIBRARY NAME WHEN DIMMING OBJECT VARIABLES

I always put the word *DAO* in front of all references to DAO objects. Here are some examples:

```
Dim Db  As DAO.DataBase
Dim rs  As DAO.Recordset
```

This way, VBA knows what library to look in for the definition of DBEngine (the top object). If you don't do this, VBA surfs the references collections to find it. You can also use the word VBA in front of functions (Left$, Mid$, MsgBox, and so on) and write the subroutine or function like this:

```
Public Sub MsgBox ()
VBA.Msgbox "The new way to use VBA :-)", vbInformation + _
    vbOkonly, "VBPJ TechTip Section"

End Sub

Private Sub  Form_Load()
    'Call the MsgBox Sub
    MsgBox

End Sub
```

—Gonzalo Medina Galup, Miami, Florida

VB4 16/32, VB5

Level: Beginning

A BETTER USE FOR STRCONV

When using proper names, you sometimes need to capitalize the first letter of each word. For example, you need to convert "john smith" into "John Smith." With VB3, you had to write a custom function to do the job; VB4's versatile StrConv routine, on the other hand, lets you do it with one statement:

```
properName = StrConv(text, vbProperCase)
```

However, be aware that this variant of StrConv also forces a conversion to lowercase for all the characters not at the beginning of a word. In other words, "seattle, USA," is converted to "Seattle, Usa," which you don't want. You still need to write a custom routine, but you can take advantage of StrConv to reduce the amount of code in it:

```
Function ProperCase(text As String) As String
    Dim result As String, i As Integer
    result = StrConv(text, vbProperCase)
    ' restore all those characters that
    ' were uppercase in the original string
    For i = 1 To Len(text)
        Select Case Asc(Mid$(text, i, 1))
        Case 65 To 90     ' A-Z
            Mid$(result, i, 1) = Mid$(text, i, 1)
        End Select
    Next
    ProperCase = result
End Function
```

—Francesco Balena, Bari, Italy

VB4 32, VB5

Level: Beginning

LOAD A GRID FROM A SQL STATEMENT

Use this code for a generic routine to load a grid from a SQL statement. The example is for Remote Data Objects (RDO) and Sheridan Software Systems' grid, but it works with minor modification for any grid and resultset type. Also, you can load combo boxes in a similar fashion:

```
Public Sub LoadGridFromSQL(TargetGrid As SSDBGrid, rdoConn _
    As rdoConnection, Sql As String, Optional ClearGrid As _
    Boolean = True)
    Dim J As Integer
    Dim rsResult As rdoResultset
    Dim sAddItem As String

    If ClearGrid Then
        TargetGrid.RemoveAll
    End If
    TargetGrid.Redraw = False

    Set rsResult = rdoConn.OpenResultset(Sql, _
        rdOpenForwardOnly, rdConcurReadOnly, rdExecDirect)
    With rsResult
        Do Until .EOF

            'Build add item string
            sAddItem = vbNullString
            For J = 1 To .rdoColumns.Count
                If IsNull(.rdoColumns.Item(J - 1)) Then
                    sAddItem = sAddItem & vbNullString & vbTab
                Else
                    sAddItem = sAddItem & _
                        .rdoColumns.Item(J - 1) & vbTab
                End If
            Next J

            'Remove extra tab from end
            TargetGrid.AddItem Left$(sAddItem, _
                Len(sAddItem) - 1)
            .MoveNext

        Loop
        .Close
    End With 'rsResult

    TargetGrid.Redraw = True
    Set rsResult = Nothing

End Sub
```

—Joe Karbowski, Traverse City, Michigan

VB4 16/32, VB5

Level: Beginning

INVISIBLE CONTROL PLACEMENT ON MDIFORM CLIENT AREA

With VB4 or higher, you can place invisible controls—such as the standard Timer and CommonDialog or UserControls built with VB5 that have their InvisibleAtRuntime property set—directly on an MDIForm. In previous versions of VB, you could only put controls with an Align property on an MDIForm.

Because CommonDialog and Timer controls are often necessary, programmers previously had to hide "container" forms or use a PictureBox as a ToolBar and cash in on its container functionality to hide the invisible controls. This is no longer necessary, though it's not widely known.

—Ron Schwarz, Mt. Pleasant, Michigan

VB5

Level: Intermediate

Q&D Zoom Using Forms 2.0 Designer

How would you like to be able to make a form automatically resize and reposition all its controls and fonts whenever you resize the form? How would you like to do that using only two lines of executable code and *no* third-party controls? It's easy, using one of VB5's little-explored features: the Forms 2.0 Designer.

To place a Forms 2.0 Designer in your project, open the Components window (hit Control T; select Components from the Project menu; or right-click on the toolbox and select Components). Click on the Designers tab.

A few caveats: You can only use the Forms 2.0 control set—it appears in its own toolbox when you're in the designer—and ActiveX controls. Only the Forms 2.0 controls scale their fonts. You can't use control arrays in a Forms 2.0 Designer. And you're not allowed to distribute the Forms 2.0 engine, so users need to have Office or Internet Explorer installed on their machines.

Here's all you need to do:

```
Option Explicit
Private w As Long

Private Sub UserForm_Initialize()
    w = Me.Width
End Sub

Private Sub UserForm_Resize()
    Me.Zoom = (Me.Width / w) * 100
End Sub
```

—Ron Schwarz, Mt. Pleasant, Michigan

VB5

Level: Intermediate

Scrollbars and 3-D Effects on Non-MDI Forms

Standard VB forms don't support a scrollable client area. Normally, when one is needed, programmers resort to convoluted solutions such as filling the client area with a picture box, placing another picture box inside the first, manually adding two scrollbars, and placing the actual form content inside the nested picture box. Then programmers add code to handle the scrolling when the scrollbars are changed.

VB5's Forms 2.0 Designer provides the ability to automatically place scrollbars on a form and have them appear only when required. Place a Forms 2.0 Designer into your project and examine the Properties window. Click on the Categorized tab, and look at the Scrolling section for information on the scrolling properties and their usage.

For information on how to add a Forms 2.0 Designer and caveats on usage, see the "Q&D Zoom Using Forms 2.0 Designer" tip.

—Ron Schwarz, Mt. Pleasant, Michigan

VB5

Level: Intermediate

Special Effects With Forms 2.0 Designer

Forms 2.0 Designer provides a variety of special visual effects. Your forms can have flat, raised, sunken, etched, or bumpy background textures. You can also have the background picture tile, zoom, or stretch (zoom without distortion) to the form size. For background effects, check out the SpecialEffect property in the Appearance section of the Properties window. Picture properties are covered in the Picture section.

For information on how to add a Forms 2.0 Designer and caveats on usage, see the "Q&D Zoom Using Forms 2.0 Designer" tip.

—Ron Schwarz, Mt. Pleasant, Michigan

VB5

Level: Beginning

Easy Updates of Property Window for Multiple Controls

When you're editing a series of controls, you can multiselect them either by clicking while holding down the Control button or by "lassoing" them with the mouse, then enter the appropriate data in the Properties window, and apply it to all selected controls.

But what about those times when you need to edit the properties for several controls, but each needs *different* data? You're in for a tedious session of clicking on the controls one by one and hopping back and forth between the Form window and Properties window.

Unless, of course, you're in on a dirty little secret: When you click on a control—or a form—you can simply start typing! As soon as you type the first key, VB automatically switches focus to the Properties window and starts entering your keystrokes into the same property that you edited in the previous control.

—Ron Schwarz, Mt. Pleasant, Michigan

VB3, VB4-16/32, VB5

Level: Beginning

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

VB3, VB4 16/32, VB5, VBA

Level: Beginning

REPEAT PERFORMANCE

A simple loop through your main string lets you count the occurrences of a specified character or string. This function is useful for determining if enough commas appear in your comma-delimited string:

```
Function Tally(sText As String, sFind As String) As Long
    Dim lFind As Long
    Dim lLast As Long

    Do
        lFind = InStr(lLast + 1, sText, sFind)
        If lFind Then
            lLast = lFind
            Tally = Tally + 1
        End If
    Loop Until lFind = 0
End Function
```

—Jeffrey Renton, Spring, Texas

VB5

Level: Intermediate

TRANSPORT A LIST

The typical way of entering user-specified data into a list box is one entry at a time; however, you can accept multiple delimited entries at once and add them to a list box with a call to this function. Passing the function a delimited string fills the list box with the values; the list box can be cleared first if requested in the bClear parameter. If you pass an empty string, the values from the list box are used to create a delimited string:

```
Function ConvertList(cList As Control, ByVal sText As _
    String, ByVal sDelimiter As String, Optional bClear As _
    Boolean = False) As String
    Dim lLoop As Long
    Dim lFind As Long

    If Len(sText) Then
        If bClear Then cList.Clear
        Do
            lFind = InStr(sText, sDelimiter)
            If lFind Then
                cList.AddItem Left$(sText, lFind - 1)
                sText = Mid$(sText, lFind + 1)
            End If
        Loop Until lFind = 0
        If Len(sText) Then cList.AddItem sText
    Else
        For lLoop = 0 To cList.ListCount - 1
            If lLoop = cList.ListCount - 1 _
                Then sDelimiter = vbNullString
            ConvertList = ConvertList & cList.List(lLoop) _
                & sDelimiter
        Next lLoop
    End If
End Function
```

Here's how you can call it to fill a list, then output the same list using a different delimiter:

```
Call ConvertList(List1, "yellow|green|red", "|", True)
Debug.Print ConvertList(List1, "", "/")
```

—Jeffrey Renton, Spring, Texas

VB3, VB4 16/32, VB5, VBA

Level: Beginning

FIND TEXT BETWEEN TWO STRINGS

This function is useful for returning a portion of a string between two points in the string. You could, for example, extract a range name returned by Excel found between parentheses:

```
Function Between(sText As String, sStart As _
    String, sEnd As String) As String
    Dim lLeft As Long, lRight As Long

    lLeft = InStr(sText, sStart) + (Len(sStart) - 1)
    lRight = InStr(lLeft + 1, sText, sEnd)

    If lRight > lLeft Then Between = _
        Mid$(sText, lLeft + 1, ((lRight - 1) - lLeft))
End Function
```

Note that it only works for the first occurrences of the start and stop delimiters.

—Jeffrey Renton, Spring, Texas

VB5

Level: Beginning

TELL ME IT'S TRUE

The typical method of validating multiple expressions is to string together a series of If statements separated with an equal number of And statements. Shorten that process by passing one or more comma-delimited equations to return a True or False result:

```
Function IsTrue(ParamArray paOptions()) As Boolean
    Dim lLoop As Long
```

```
    IsTrue = True
    For lLoop = LBound(paOptions) To UBound(paOptions)
        IsTrue = IsTrue And paOptions(lLoop)
        If paOptions(lLoop) = False Then Exit For
    Next lLoop
End Function
```

—Jeffrey Renton, Spring, Texas

VB5

Level: Beginning

Make One Form Parent of Another

Prior to VB5, when you wanted to make a form appear on top of another form, you either made it modal or used an MDIForm with children. If you wanted to go beyond that, you had to use API calls. Starting with VB5, you can use the Show method's optional *ownerform* parameter to set one form as the parent of another. This means you always place the child form—not an *MDI* child—on top of the parent form, even though the parent form remains active. You can also use the *style* vbModal parameter to force modality, but that defeats any reason to use *ownerform*. Here's how you invoke the *ownerform* parameter, making a form a nonmodal child of a non-MDIForm:

```
Private Sub Command1_Click()
    Form2.Show , Me
End Sub
```

—Ron Schwarz, Mt. Pleasant, Michigan

VB3, VB4 16/32, VB5

Level: Intermediate

Do Easy Form-Level Keystroke Trapping

If you set a form's KeyPreview property to True, all keystrokes for items on that form first trigger the form's events. This makes it easy to do form-level filtering and keystroke trapping. For example, if you want to make all text boxes on a form force uppercase entry, you can do it with three lines of executable code:

```
Private Sub Form_KeyPress(KeyAscii As Integer)
    If KeyAscii >= 97 And KeyAscii <= 122 _
        Then 'a-z
        KeyAscii = KeyAscii - 32
    End If
End Sub
```

—Ron Schwarz, Mt. Pleasant, Michigan

VB4 32, VB5

Level: Advanced

Avoid Binary Compatibility Problems

To prevent losing the ability to maintain binary compatibility with compiled object code, take the first compiled build and move it into a separate directory, then tell VB to maintain compatibility with it. It makes intuitive sense to maintain compatibility with each previous version, but it leaves lots of room for breaking compatibility. Maintaining a separate version that is used solely as a compatibility master protects you.

—Ron Schwarz, Mt. Pleasant, Michigan

VB3, VB4 16/32, VB5

Level: Intermediate

Emulate a Click Event for Right-Clicks Over CommandButton Controls

Sometimes it's useful to trap the right click over controls such as CommandButton. Unfortunately, the Click event only fires for left button clicks. The MouseDown and MouseUp events fire for both buttons, and even report which button was clicked, but nothing in life could ever be that simple, right?

The first problem is that if you use the MouseDown event, you trap the click when the user clicks down on the key, which is counter to the way things normally work in Windows. For instance, when you click the left button, the control's Click event won't fire until you release, giving you the ability to slide off the control before releasing. A fire-on-downstroke event gives no such safety net.

So you're probably thinking, "Well, then just use the MouseUp event!" This solution creates another gotcha: Even if you slide off the control before releasing, the MouseUp event fires anyway!

Fortunately, the MouseUp event reports the mouse cursor's X and Y positions, and using simple math comparing them to the control's placement and size, it's easy to determine whether the mouse was over the control at the instant it was released. Here's how you do it:

```
Option Explicit
Private Sub Command1_MouseUp(Button As Integer, _
    Shift As Integer, X As Single, Y As Single)
Dim OffMe As Boolean
        If Button = 2 Then    'right button
            X = X + Command1.Left
            Y = Y + Command1.Top
            OffMe = False
            Select Case X
                Case Is < Command1.Left, Is > _
                (Command1.Left + Command1.Width)
                    OffMe = True
        End Select
        Select Case Y
            Case Is < Command1.Top, Is > _
                (Command1.Top + Command1.Height)
                    OffMe = True
        End Select
        If Not OffMe Then
            '*****************
            'Your code goes here
            '*****************
        End If
    End If
End Sub
```

—Ron Schwarz, Mt. Pleasant, Michigan

VB3, VB4 16/32, VB5

Level: Intermediate

Q&D Sort Using Hidden List Box

VB has no built-in Sort function. Although lots of sort routines of varying complexity and performance are available, there's a simple "cheat" that you might find useful. The standard VB ListBox control has a Sorted property. Anything you add to a list box will automatically be placed in its proper rank, if you set Sorted to True before running the program. (Sorted is read-only at run time.)

Then simply use the AddItem method to insert items, and the ListBox maintains them in sorted order. A couple things to watch out for: list boxes store everything as strings. Although you can use Evil Type Coercion (ETC) to load numbers into them, keep in mind that as strings, they're sorted according to string rules. That means that 900 are perceived as greater than 1,000. This code uses random numbers and adds leading zeros to them as needed after ETCing them to strings. This code uses two CommandButtons, one ListBox, and one TextBox with its MultiLine property set to True. Click on Command1 to load up the ListBox, and click on Command2 to extract the data one element at a time and display it in Text1.

Don't forget to set the ListBox's Visible property to False for your real applications:

```
Option Explicit
Dim c As Long
Dim i As String
Private Sub Command1_Click()
    List1.Clear
    For c = 1 To 10
        i = Int(Rnd * 10000)
        While Len(i) < 4
            i = "0" + i
        Wend
        List1.AddItem i
    Next
End Sub
Private Sub Command2_Click()
    Text1 = ""
    For c = 0 To List1.ListCount
        Text1 = Text1 & List1.List(c) & vbCrLf
    Next
End Sub
```

—Ron Schwarz, Mt. Pleasant, Michigan

VB4 32, VB5

Level: Intermediate

A Generic Routine to Fill Unbound Lists

A common need in database processing is retrieving a list of the values of a particular column field for every record in a table. This function takes arguments for a database name, table name, field name, and optional SQL criteria string, and it returns a collection that contains the list of all row values for the specified column field:

```
Public Function GetColumnData(ByVal DbName As String, _
    ByVal TableName As String, ByVal DataFieldName As _
    String, Optional WhereCriteria As String) As Collection
    Dim WS As Workspace
    Dim DB As Database
    Dim RS As Recordset
    Dim SQLQuery As String
    Dim Results As Collection
    Dim FieldValue As String
    Dim Count As Integer

    Set WS = CreateWorkspace("", "admin", "", dbUseJet)
    Set DB = WS.OpenDatabase(DbName)
    SQLQuery = "SELECT " & TableName & _
        "." & DataFieldName & " FROM " & TableName
    If WhereCriteria <> "" Then _
        SQLQuery = SQLQuery & " WHERE " & WhereCriteria
    Set Results = New Collection
    Set RS = DB.OpenRecordset(SQLQuery, dbOpenForwardOnly)
    If Not RS Is Nothing Then
        Count = 0
        'this count will be a unique key
        'in the collection
        Do While Not RS.EOF
            FieldValue = RS.Fields(DataFieldName)
            Results.Add FieldValue, CStr(Count)
            Count = Count + 1
            RS.MoveNext
        Loop
        RS.Close
        Set RS = Nothing
End If
    DB.Close
    Set DB = Nothing
    WS.Close
    Set WS = Nothing
    Set GetColumnData = Results
    Set Results = Nothing

End Function
```

This procedure is great for filling unbound lists and combo boxes or for driving other database processing based on the returned list. Here's a simple example:

```
' get a list of Social Security numbers
' for all employees over age 65
Dim lst As Collection
    Dim i As Integer

    Set lst = GetColumnData("employee.mdb", _
        "tblEmployees", "SSNum", "Age>65")

    If Not lst Is Nothing Then
        For i = 1 To lst.Count
            'do something with lst(i)
        Next i
        Set lst = Nothing
    End If
```

In this code, efficiency is traded for ease of use. The procedure opens a connection to the database each time it's called, which is an expensive operation, especially if used inside a loop. As an alternative, you could pass an optional database object. Another efficiency enhancement would be to declare the GetColumnData function as Recordset. After the recordset is open, simply Set GetColumnData = RS. By doing this, you can dispense with the collection altogether. It would also save an iteration through the recordset/collection within the GetColumnData function to assign it to the collection.

Also, note that duplicate values are allowed in the returned collection. I left out error checking intentionally to keep the code as short as possible.

—Allen Broadman, received by e-mail