Welcome to the Tenth Edition of the VBPJ Technical Tips Supplement!

These tips and tricks were submitted by professional developers using Visual Basic 3.0 through 6.0, Visual Basic for Applications (VBA), and Visual Basic Script (VBS). The editors at Visual Basic Programmer's Journal compiled the tips. Instead of typing the code published here, download the tips for free from VBPJ's Web site at www.vbpj.com.

If you'd like to submit a tip to VBPJ, please send it electronically to vbpjtips@fawcette.com. You can also send it to User Tips, Fawcette Technical Publications, 209 Hamilton Ave., Palo Alto, CA, USA, 94301-2500, or fax it to 650-853-0230. 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, VB5, or VB6. Please limit code length to 20 lines. Don't forget to include your e-mail and mailing address, and let us know your payment preference: $25 per published tip, or an extension of your VBPJ subscription by one year.

VB6
Level: Intermediate

Split Strings Cleanly

The Split function is great for parsing strings, but what happens when a string has more than one consecutive delimiter? It might seem odd that Split() returns empty substrings as placeholders for the data missing between delimiters, but that's exactly what needs to happen so these data positions aren't lost. Unfortunately, Split() does not have an option to ignore multiple delimiters. CleanSplit() uses the same arguments as Split() and efficiently discards empty substrings caused by more than one delimiter in a row:

```
Public Function CleanSplit(ByVal Expression As String, _
    Optional By Val Delimiter As String = " ", Optional _
    ByVal Limit As Long = -1, Optional Compare As _
    VbCompareMethod = vbBinaryCompare) As Variant
    Dim varSubstrings As Variant, i As Long
    varSubstrings = Split(Expression, Delimiter, _
        Limit, Compare)
    'mark empty substrings with delimiter because
    'the delimiter won't be around after Split()
    For i = LBound(varSubstrings) To UBound(varSubstrings)
        If Len(varSubstrings(i)) = 0 Then
            varSubstrings(i) = Delimiter
    Next i

    CleanSplit = Filter(varSubstrings, Delimiter, False)
End Function
```

—Mark Pickenheim, Springfield, Virginia

VB5, VB6
Level: Beginning

Define Properties to Use Standard Dialogs

It's easy to add the standard LoadPicture and Font dialogs, complete with ellipsis, in a UserControl's property list. The trick is to define the properties as StdPicture or StdFont, respectively. For example, you can paste this code into a new UserControl to load an image into the UserControl's Picture property and change the font for an embedded textbox. Start a new project in VB5 or VB6, add an ActiveX control (UserControl), drop a textbox onto it, and paste in the code:

```
Option Explicit
Private Sub UserControl_InitProperties()
    ' Start with some sample text
    Text1.Text = "Some sample text"
End Sub

Private Sub UserControl_ReadProperties(PropBag _
    As PropertyBag)
    ' Restore any changes we made during design mode
    Set UserControl.Picture = _
        PropBag.ReadProperty("Image", Nothing)
```

```
    Set Text1.Font = PropBag.ReadProperty("Font", Nothing)
    Text1.Text = PropBag.ReadProperty("Text", _
        "Some sample text")
End Sub

Private Sub UserControl_WriteProperties(PropBag _
    As PropertyBag)
    ' Save any changes we made during design mode
    PropBag.WriteProperty "Image", _
        UserControl.Picture, Nothing
    PropBag.WriteProperty "Font", Text1.Font, Nothing
    PropBag.WriteProperty "Text", Text1.Text, _
        "Some sample text"
End Sub
Public Property Get Image() As StdPicture
    ' return the UserControl's image (if any)
    Set Image = UserControl.Picture
End Property
Public Property Set Image(ByVal newBackground As StdPicture)
    ' change the UserControl's background image
    Set UserControl.Picture = newBackground
    PropertyChanged "Image"
End Property
Public Property Get Font() As StdFont
    ' get the current textbox font details
    Set Font = Text1.Font
End Property
Public Property Set Font(ByVal newFont As StdFont)
    ' update the textbox font details
    Set Text1.Font = newFont
    PropertyChanged "Font"
End Property
```

Close the UserControl's designer window and add a new project (Standard EXE). Drop a copy of the newly created UserControl onto the form, making sure the new control instance is large enough to see the textbox, and check the property list. Apart from the standard properties provided by VB, the property list contains two extra properties—Image and Font—complete with VB's standard "..." for popping up the Image and Font dialogs. For example, if you change the Font properties, the font of the text displayed in the textbox changes immediately, and selecting an image places that image in the UserControl's Picture property.

—John Cullen, Pedroucos, Portugal

VB5, VB6
Level: Intermediate

Toss the Common Dialogs Control

In Chris Barlow's article "Standardize Your Dialogs" [Getting Started, VBPJ June 1999], he explained the use of the CommonDialog control. However, you don't need this control to use Windows' common dialog boxes. Microsoft provides a replacement DLL on the VB5 and VB6 CDs. This DLL is half the size of ComDlg32.ocx (64K vs. 126K) and doesn't require placing a control on a form. And, perhaps best of all, it provides a Center property. See \Tools\Unsupprt\DlgObj on the VB5 CD-ROM (or \VB98\WIZARDS\ PDWIZARD on the VB6 CD-ROM) for installation details.

Add a reference to DlgObjs to your project by selecting Microsoft Dialog Automation Objects from the Project References dialog. This code gets a filename:

```
Public Function GetFilename(WinHandle As Long) As String
    Dim Dlg As ChooseFile
    On Error Resume Next
    Set Dlg = New ChooseFile
    With Dlg
        .Save = True
        ' We want a Save As dialog box.
        .Center = True
        ' We want the dialog centered.
        .hWnd = WinHandle ' Need a parent window.
        .HideReadOnly = True
        ' Don't need the 'Open as Read Only' box.
        .MultiSelect = False
        ' Don't select multiple files.
        .OverwritePrompt = True
        ' Ask to overwrite an existing file.
```

```
        .Filters.Add "BAS Files (*.bas):*.bas" ' File mask.
        .Filters.Add "All Files (*.*):*.*" ' File mask.
        If .Show Then
            GetFilename =.Directory & "\" & .filename
        Else
            GetFilename = ""  ' User pressed Cancel
    End With
    Set Dlg = Nothing
End Function
```

Call GetFilename by passing the handle to the window that acts as the dialog's parent:

```
Debug.Print GetFilename(Me.hWnd)
```

—Frank Mokry, Palgrave, Ontario, Canada

VB3 and up
Level: Beginning

Insert Soft Breakpoints

If you want to step through the code behind a certain screen element at run time, you might do something like this in break mode: Hit Ctrl-Break from run mode, use the Find dialog to find the specific piece of code—say cmdSave_Click—put a breakpoint on a line inside that sub, then press F5 to continue the program execution.

Here's an easier way: Hit Ctrl-Break to break while the program is running, then press F8. This causes VB to enter break mode just as the next line of code is about to be executed. It appears the program is running normally, but the next UI action you perform halts execution at the first line of code encountered. Clicking on cmdSave now puts you in break mode inside cmdSave_Click. You get to cmdSave_Click without searching for this event procedure and adding a breakpoint. If you plan on debugging the event procedure several times, add a breakpoint now.

—Janakiraman Sattainathan, Mayodan, North Carolina

VB4, VB5, VB6
Level: Beginning

Iterate Through Control Arrays

When you need to iterate through all the elements of a control array, use this code:

```
Dim J As Integer
For J = Text1.LBound to Text1.Ubound
    With Text1(J)
        ' Work here...
    End With
Next J
```

This way, if you add or remove controls from the array, your code still works. This assumes that the array has no holes.

—Dave Kinsman, Renton, Washington

VB4, VB5, VB6
Level: Intermediate

Have Your Functions Both Ways

When you create generic functions, remember that many functions are a two-way street. For example, many general utility modules contain both LoWord and HiWord functions. However, to compose a Long, a MakeDWord or some related function is required. Instead, convert your LoWord and HiWord functions into properties. That way, they can exist on both sides of the equal sign, for retrieval and for assignment:

```
x = LoWord(Y)
```

Or:

```
LoWord(y) = x
```

Use this code to implement:

```
Public Property Get HiWord(L As Long) As Integer
    ' used to retrieve the HiWord of a long

    HiWord = <code to do it!>
```

```
End Property

Public Property Let HiWord(L As Long, ByVal _
    NewValue As Integer)
    ' used to set the HiWord of a long L is the long to
    ' modify, NewValue is the integer you want to set into
    ' the HiWord of the long L
End Property
```

Also, remember you don't have to put this in a class module. Property procedures are completely valid within a regular BAS module, but if you put them in a BAS module, you must precede their names with the module name when calling them.

You could also use this technique to create a path parse routine that lets you replace arbitrary elements of the path string (such as the drive, path, or filename), a timer function, or a Split function that lets you assign elements directly into the splitting string as well as retrieve them.

—Darin Higgins, Fort Worth, Texas

VB3 and up

Level: Beginning

Create Default Directory for Project Shortcuts

This is an update to "Start Up in Your Code Folder" ["101 Tech Tips for VB Developers," Supplement to VBPJ, August 1999]. You can store your VB Project Files (VBP) and Project Group Files (VBG) under any organization system, and still open each project with its own folder as the default folder for saving and opening files. Create a shortcut—on the desktop or in your folder/toolbar—linked directly to the VBP or VBG file. The "Start in:" entry is automatically set to the same folder as the target file. Double-clicking on the shortcut starts VB, loaded with the desired project or group, and its folder is the default directory. You can create any number of such shortcuts, each linked to its own project or group in its own directory. This also saves you the extra step of first loading VB each time, then choosing the project. My boss, Harry Suber, discovered and taught me this trick.

—Robert A. Henkel, Berlin, Maryland

VB5, VB6

Level: Beginning

Hard-Code Your Watch Points

I've seen several differently angled references to the Debug.Assert method in your Tech Tips supplements. They usually advise readers to use the "Debug.Assert False" line when they want a persistent breakpoint, even when the breakpoints are cleared or files are closed and reopened later. I want to suggest another useful variation. Any expression that evaluates to False triggers the breakpoint. For example, if you are testing a routine that seems to work fine until it processes record 413 near the end of your list of records, you can insert this line into your code:

```
Debug.Assert MyRecords.RecNumber <> 413
```

This breaks the code only when the expression evaluates to False—when the record number equals 413. The use of the record number value is just an example; you can use any unique identifying variable value to stop only at the desired area, and run uninterrupted through all the other passes of the same code.

—Robert A. Henkel, Berlin, Maryland

VB4/32, VB5, VB6

Level: Intermediate

Let Users Resize Your Controls

You can allow users to resize a control—just like in VB design mode—with a mouse, using two simple API calls. You can resize the control—top-left, top, top-right, left, right, bottom-left, bottom, and right. When you make ranges for the mouse coordinates (such as x>0 and x<100), the MouseDown event activates the API functions and sizes your picture box when the mouse moves. This code assumes you have a picture box on the form:

```
Private Declare Function ReleaseCapture Lib _
    "user32" () As Long
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
Private Const WM_NCLBUTTONDOWN = &HA1
' You can find more of these (lower) in the API Viewer. Here
```

```
' they are used only for resizing the left and right.
Private Const HTLEFT = 10
Private Const HTRIGHT = 11
Private Sub Picture1_MouseDown(Button As _
    Integer, Shift As Integer, X As Single, Y As Single)
    Dim nParam As Long
    With Picture1
        ' You can change these coordinates to whatever
        ' you want
        If (X > 0 And X < 100) Then
            nParam = HTLEFT
        ElseIf (X > .Width - 100 And X < .Width) Then
        ' these too
            nParam = HTRIGHT
        End If
        If nParam Then
            Call ReleaseCapture
            Call SendMessage(.hWnd, _
                WM_NCLBUTTONDOWN, nParam, 0)
        End If
    End With
End Sub
Private Sub Picture1_MouseMove(Button As _
    Integer, Shift As Integer, X As Single, Y As Single)
    Dim NewPointer As MousePointerConstants
    ' You can change these coordinates to whatever you want
    If (X > 0 And X < 100) Then
        NewPointer = vbSizeWE
    ElseIf (X > Picture1.Width - 100 And X < _
        Picture1.Width) Then ' these too
        NewPointer = vbSizeWE
    Else
        NewPointer = vbDefault
    End If
    If NewPointer <> Picture1.MousePointer Then
        Picture1.MousePointer = NewPointer
    End If
End Sub
```

—Fran Pregernik, Zagreb, Croatia

VB4/32, VB5, VB6, VBA
Level: Intermediate

Fill In the E-Mail Fields

ShellExecute is one of the most flexible Win32 APIs. Using ShellExecute, you can pass any filename, and if the file's extension is associated to a registered program on the user's machine, the correct application opens and the file is played or displayed.

In the February 1998 101 Tech Tips supplement, Jose Rodriguez Alvira showed ShellExecute's Internet power ("Create Internet-Style Hyperlinks"). If you pass an HTTP URL, the user's default 32-bit Web browser opens and connects to the site. If you pass an email address that has been prefaced with "mailto:", the user's default 32-bit e-mail client opens a new e-mail note with the address filled in.

Here's how to automatically get a lot more than just the e-mail addresses filled in. If you want to include a list of CC recipients, BCC recipients, or your own subject text or body text, you can create a string variable, add the list of primary addresses (separated by semicolons), then a question mark character and element strings prefaced like this:

For CCs (carbon copies): &CC= (followed by list)
For blind CCs: &BCC= (followed by list)
For subject text: &Subject= (followed by text)
For body text: &Body= (followed by text)
To add an attachment: &Attach= (followed by a valid file path within chr(34)'s)

To use this trick, create a new VB project, add a form, and add six textboxes and a button (cmdSendIt). Paste this into the form's Declarations section:

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

```
Private Const SW_SHOWNORMAL = 1
```

Paste this code into the button's Click event:

```
Private Sub cmdSendIt_Click()
    Dim sText As String
    Dim sAddedText As String
    If Len(txtMainAddresses) Then
        sText = txtMainAddresses
    End If
    If Len(txtCC) Then
        sAddedText = sAddedText & "&CC=" & txtCC
    End If
    If Len(txtBCC) Then
        sAddedText = sAddedText & "&BCC=" & txtBCC
    End If
    If Len(txtSubject) Then
        sAddedText = sAddedText & "&Subject=" & txtSubject
    End If
    If Len(txtBody) Then
        sAddedText = sAddedText & "&Body=" & txtBody
    End If
    If Len(txtAttachmentFileLocation) Then
        sAddedText = sAddedText & "&Attach=" & _
            Chr(34) & txtAttachmentFileLocation & Chr(34)
    End If
    sText = "mailto:" & sText
    ' clean the added elements
    If Len(sAddedText) <> 0 Then
        ' there are added elements, replace the first
        ' ampersand with the question character
        Mid$(sAddedText, 1, 1) = "?"
    End If
    sText = sText & sAddedText
    If Len(sText) Then
        Call ShellExecute(Me.hWnd, "open", sText, _
            vbNullString, vbNullString, SW_SHOWNORMAL)
    End If
End Sub
```

You can't have spaces between the ampersands and tags, or between the tags and the equal signs. You don't have formatting options, so body text will be one paragraph. However, when you use this technique, program errors are emailed to you with full details, and you can create real e-mail applets in a just a few seconds. It beats automating a full e-mail program.

In addition, almost all this functionality is possible in HTML MailTo tags. Here is a sample:

```
<A HREF="mailto:smith@smithvoice.com?subject=
Feedback From VisualBasic ett
smithvoice.com/vbfun.htm&CC=smith@smithhome.org&BC
C=fred@fred.net;bill@home.com&body=hello how are
you">feedback@smithvoice</A>
```

I have yet to get HTML to do the attachments, but attachments are no problem in VB.

Editor's Note: The full functionality of these extra fields is available in e-mail clients that are totally Exchange-compliant. Some or all of the extra fields might not work with noncompliant e-mail clients.

—Robert Smith, Kirkland, Washington

VB4, VB5, VB6

Level: Beginning

Maintain Call Stack for Error Tracing

I program all reusable components into DLLs or OCXs. To provide a consistent error-handling technique across all my projects, I use the Raise method of the Err object in all these components and display the error only in the code module of first entry—such as Command1_Click. Because an error can be generated several layers deep in the code, I propagate the location of the error using this Raise statement in all my reusable components:

```
ThisProcEH:
    Err.Raise Err.Number, "ThisProc" & vbCr & Err.Source
Exit Sub
```

This way, the whole call stack is returned to—and can be displayed in—the calling procedure (through

Err.Source), making errors much easier to find and solve.

—Harvey Flaisher, McLean, Virginia

VB5, VB6
Level: Intermediate

Create Close-All Add-In

I often open up many windows in a VB project when I'm doing a search and/or replace in code. On large projects, it becomes cumbersome to close all those windows one by one. Here's a quick add-in you can create to close those windows for you. Start a new project and select the Add-In type. Paste this code into the frmAddIn that's created for you:

```
Private Sub Form_Load()
    Dim w As Window
    For Each w In VBInstance.Windows
        ' close either code or form design windows
        ' that are visible
        If (w.Type = vbext_wt_CodeWindow Or _
            w.Type = vbext_wt_Designer) And w.Visible Then
            w.Close
        End If
    Next
    Unload Me
End Sub
```

Highlight the Connect class in the Object Browser, right-click on it, and edit the Description property to change the name and description of the add-in. Also, search the template code and replace "My Add-In" with whatever you decided to call it. After building the DLL, you can add it from the Add-In Manager and close all those pesky windows in no time.

—Rick Lalliss, received by e-mail

VB3 and up
Level: Intermediate

Right Click on a VB File to Open in Notepad

You might have wished you could right-click on a VB file and open it in Notepad, or copy a snippet of code for another app you're working on. Try this: Drop this code in a text file named FormEdit.reg, save it, and close the file:

```
REGEDIT4
[HKEY_CLASSES_ROOT\VisualBasic.Form\shell\edit]
@="&Edit"
[HKEY_CLASSES_ROOT\VisualBasic.Form\shell\edit\ command]
@="Notepad.exe \"%1\""
```

When you double-click on the file, the contents are automatically loaded into the System Registry. Right-click on any VB form and you should see Edit in the popup menu. Do the same for the other VB text files—VisualBasic.ClassModule, VisualBasic.Module, and VisualBasic.Project—and use Regedit.exe in the Windows folder to verify the results. You can substitute Word or any other text editor for Notepad by setting the command entry:

```
@="C:\\Program Files\\Microsoft
    Office\\Office\\Winword.exe %1"
```

Be mindful of the double backslashes and where you load Office—they must be correct for this to work.

Although the locations of the Registry entries for VB4, VB5, and VB6 are different, you can get this to work for any VB text file including BAS, CLS, FRM, and VBP. It does not work on FRX or other binary files. This also works with HTML files. Make a similar entry for your default browser to edit the HTML file as plain text with nothing more than a right-click. Or set it up for your GIF files to run under your browser to see how they will look. Most important, be careful anytime you do anything to the Registry.

—Greg Moss, Amarillo, Texas

VB4/32, VB5, VB6
Level: Intermediate

Use a Class Module for Persistent Global Variables

When I use variables that need to be saved to the Registry or to an INI file, I sometimes forget to place the

API call after the variables in a program have changed. This might cause problems later when the program is exited and restarted. Instead, use the variable in a class module that can be used as a local variable for a form/routine, or make the class a global class in the BAS module. When assigning a value to the variable when it's a class, you use Property Let. In this routine, place your API call to save the new value to the Registry or INI file. Place this code in your BAS module:

```
Public Const ProgramName = "YourProgram"
Public Const AppName = "General"
Public Const AppKey = "Last User"
Public Const INIFile = "YourINIFile.INI"
' This makes all objects in ProjectVariables
' global to the program.
Public Variables As New ProjectVariables
```

Place this code in your class module:

```
' ProjectVariables.cls
Option Explicit
Private Declare Function _
    WritePrivateProfileString Lib "kernel32" _
    Alias "WritePrivateProfileStringA" _
    (ByVal lpApplicationName As String, _
    ByVal lpKeyName As Any, ByVal lpString As _
    Any, ByVal lpFileName As String) As Long
Private m_sUserName As String
Property Get UserName() As String
    UserName = m_sUserName
End Property
Property Let UserName(ByVal sUserName As String)
    If sUserName <> m_sUserName Then
    ' Use this to save to the Registry
    SaveSetting ProgramName, AppName, AppKey, sUserName
    ' Use this to save to an INI File
    Call WritePrivateProfileString(AppName, _
        AppKey, sUserName, INIFile)
    End If
    m_sUserName = sUserName
End Property
```

With this code behind the Property Let, whenever you change the UserName variable to a new value, the key/value pair is automatically written to the INI file or Registry.

—John Zenkavich, Clarks Summit, Pennsylvania

VB4/32, VB5, VB6

Level: Intermediate

Use Faster Floating-Point Division

If you do a lot of floating-point division operations in VB, you can optimize these operations by multiplying by the reciprocal value. For example, instead of performing this calculation:

```
X/Y
```

Do this:

```
X * (1 / Y)
```

You can see how this works in VB by adding this code to a form in a new project:

```
Private Declare Function GetTickCount Lib _
    "kernel32" () As Long
Const NORMAL As Double = 1453
Const RECIPROCAL As Double = 1 / NORMAL
Const TOTAL_COUNT As Long = 10000000
Private Sub Form_Click()
    Dim dblRes As Double
    Dim lngC As Long
    Dim lngStart As Long
    On Error GoTo Error_Normal
    lngStart = GetTickCount
    For lngC = 1 To TOTAL_COUNT
```

```
        dblRes = Rnd / NORMAL
        Next lngC
    MsgBox "Normal Time:  " & GetTickCount - lngStart
    lngStart = GetTickCount
    For lngC = 1 To TOTAL_COUNT
        dblRes = Rnd * RECIPROCAL
    Next lngC
    MsgBox "Reciprocal Time:  " & GetTickCount - lngStart
Exit Sub
Error_Normal:
    MsgBox Err.Number & " - " & Err.Description
End Sub
```

I've seen consistent performance gains of 15 percent with the reciprocal technique. But be careful of rounding issues—for example, 3/3 = 1, but 3 * (0.333333...) = 0.999999....

—Jason Bock, Germantown, Wisconsin

VB5, VB6
Level: Intermediate

Dual Procedure IDS

When using VB to create a new ActiveX control with a Caption or Text property, you can set the corresponding Procedure ID to cause the Property Browser to update the property value with each keystroke, just as the Label and Text controls do. However, it's not obvious how to make it the default property as well. That's because (Default), Caption, and Text all appear in the Procedure ID list, allowing only one ID to be assigned to your new property. To get around this limitation, create another hidden property (check "Hide this member" in the Procedure Attributes dialog) that updates the same variable and you can make that the default property.

—Chuck Liem, Olathe, Kansas

VB4/32, VB5, VB6
Level: Intermediate

Use GetInputState in Loops

Some developers suggest putting DoEvents in loops to keep your application responsive. This is never a good idea. If the loop is short, you don't need it; if the loop is long, you'll take an unacceptable performance hit. But what if you want your user to be able to click on a Cancel button or perform some other action while a long loop is executing?

A good compromise is to call GetInputState. This API function returns 1 if the user has clicked on a mouse button or hit a key on the keyboard. The overhead for GetInputState is much less than for DoEvents, so your loop runs faster. If a keyboard or mouse event occurs, then you can call DoEvents. In other words, you call the expensive DoEvents only when you actually need it to process an event. You can further reduce overhead by checking only every x iterations (the exact number being dependent on how time-consuming each loop is):

```
Option Explicit
Private Declare Function GetInputState Lib "user32" () _
    As Long
Private m_UserCancel As Boolean
Private Sub cmdCancel_Click()
    m_UserCancel = True
End Sub
Private Sub cmdGo_Click()
    Dim lCounter As Long
    m_UserCancel = False
    Me.MousePointer = vbHourglass
        For lCounter = 0 To 10000000
        'any long loop that may need to be interrupted
            If lCounter Mod 100 Then
                If GetInputState <> 0 Then
                'a mouse or keyboard event is in the
                'message queue so we call DoEvents
                'so it can be processed
                    DoEvents
                If m_UserCancel Then Exit For
                End If
            End If
        Next lCounter
    Me.MousePointer = vbDefault
End Sub
```

—Daniel R. Buskirk, Bronx, New York

VB4/32, VB5, VB6
Level: Intermediate

Generate GUIDs With One API Call

I read a great advanced tip on how to create a GUID in "Generate Unique String IDs" ["101 Tech Tips for VB Developers," Supplement to VBPJ, August 1999]. However, you can use only one API call instead of four. The OLE32.dll contains a function called WinCoCreateGUID that does all the math for you. I have included a second function called PadZeros to format the GUID:

```
Option Explicit
Private Type GUIDTy pe
    D1      As Long
    D2      As Integer
    D3      As Integer
    D4(8)   As Byte
End Type
Private Declare Function WinCoCreateGuid Lib "OLE32.DLL"
    Alias "CoCreateGuid" (g As GUIDType) As Long
Public Function CreateGUIDString() As String
    Dim g As GUIDType
    Dim sBuf As String
    Call WinCoCreateGuid(g)
    sBuf = PadZeros(Hex$(g.D1), 8, True) & _
        PadZeros(Hex$(g.D2), 4, True) & _
        PadZeros(Hex$(g.D3), 4, True) & _
        PadZeros(Hex$(g.D4(0)), 2) & _
        PadZeros(Hex$(g.D4(1)), 2, True) & _
        PadZeros(Hex$(g.D4(2)), 2) & _
        PadZeros(Hex$(g.D4(3)), 2) & _
        PadZeros(Hex$(g.D4(4)), 2) & _
        PadZeros(Hex$(g.D4(5)), 2) & _
        PadZeros(Hex$(g.D4(6)), 2) & _
        PadZeros(Hex$(g.D4(7)), 2)
    CreateGUIDString = sBuf
End Function

Private Function PadZeros(ByVal sBit As String, _
    ByVal iStrLen As Integer, Optional bHyphen _
    As Boolean) As String
    If iStrLen > Len(sBit) Then
        sBit = Right$(String$((iStrLen - Len(sBit)), _
            "0") & sBit, iStrLen)
    End If
    If bHyphen Then sBit = sBit & "-"
    PadZeros = sBit
End Function
```

—Dj Hackney, received by e-mail

VB5, VB6
Level: Beginning

Avoid Line Input Null Problems

Sometimes you have to move information from a flat file (mainframe or ASCII text file) to a database. Usually, this flat file is a set of records, and the delimiter between records is a carriage return/linefeed pair. Records might be different sizes and can contain Null characters. This code is a standard way to read a file line by line, but fails because strBuff loses Null characters, and the structure of the current record is incorrect:

```
Do Until EOF(1)
    '//-------After the next statement strBuff
    'will be without Null characters
    Line Input #1, strBuff
Loop
```

I know at least three ways to fix this problem. Here's my favorite way. You have to use Microsoft Scripting Runtime DLL (scrun.dll):

```
Dim FSO  As New FileSystemObject
Dim TS As TextStream
Dim strBuff As String
Set TS = FSO.OpenTextFile("c:\MyFlatFile.txt", ForReading)
```

```
Do Until TS.AtEndOfStream
    '//-------Reading current line and replacing
    'Null characters to spaces
    strBuff = Replace(TS.ReadLine, Chr$(0), " ")
    '//-------Your parsing code here
Loop
TS.Close
Set FSO = Nothing
```

This code returns a correct result because the ReadLine method doesn't lose Null characters. I used the Replace function only to prepare the current line for parsing. VB5 users need to use another approach to replace Nulls with space characters.

—Vladimir Olifer, Brooklyn, New York

VB5, VB6
Level: Intermediate

Use WithEvents to Communicate Between MDI and MDIChild Forms

Here's a neat way to pass events such as toolbar clicks and menu selections from an MDIForm to an active MDIChild form. Suppose the MDI parent has a toolbar control called tbrMain. Add this code to the MDIForm:

```
Event ButtonClick(strKey As String)
Private Sub tbrMain_ButtonClick(ByVal Button _
    As MSComctlLib.Button)
    RaiseEvent ButtonClick(Button.Key)
End Sub
```

Then add this code to each MDIChild form you want to receive the custom ButtonClick event:

```
Private WithEvents m_mdiParent As mdiParent
Private Sub Form_Activate()
    Set m_mdiParent = mdiParent
End Sub
Private Sub Form_Deactivate()
    Set m_mdiParent = Nothing
End Sub
Private Sub m_mdiParent_ButtonClick(strKey As String)
    ' Sample code assuming Button.Key values of
    ' "New", "Change", "Delete" and "Save"
    Select Case strKey
        Case "New"
            PerformNewAction
        Case "Change"
            PerformChangeAction
        Case "Delete"
            PerformDeleteAction
        Case "Save"
            PerformSaveAction
    End Select
End Sub
```

The effect is almost the same as declaring a control called m_mdiParent that has a ButtonClick event on your form. Use the Activate and Deactivate events to ensure that the active MDIChild is the only one that receives the ButtonClick event.

—Pat Dooley, Cleveland, Ohio

VB4/32
Level: Intermediate

Add Option/Checkbox Toggle Capability in VB4/32

VB5 introduced a Style property of the checkbox or option button, with a graphical setting that makes the button or checkbox appear to be a command button instead of its default appearance. It behaves like a toggle button in Access. However, if you're using VB4, you need help from the Windows API to get this behavior. Create a new module and enter this code:

```
Option Explicit
Public Const BS_PUSHLIKE& = &H1000&
Public Const GWL_STYLE = (-16)
Public Declare Function GetWindowLong _
    Lib "user32" Alias "GetWindowLongA" _
```

```
    (ByVal hWnd As Long, ByVal nIndex As Long) As Long
Public Declare Function SetWindowLong Lib _
    "user32" Alias "SetWindowLongA" _
    (ByVal hWnd As Long, ByVal nIndex As Long, _
    ByVal dwNewLong As Long) As Long
Public Sub MakeButton(ctl As Control)
    Dim lngReturn As Long
    Dim lngStyle As Long
    lngStyle = GetWindowLong(ctl.hWnd, GWL_STYLE)
    lngStyle = lngStyle Or BS_PUSHLIKE
    If lngStyle Then
        lngReturn = SetWindowLong(ctl.hWnd, _
            GWL_STYLE, lngStyle )
    End If
End Sub
```

Then create a form and add a checkbox and an option box (Check1 and Option1). Make them the size you want them to appear as a button. In the Form_Load event, add this code:

```
MakeButton Check1
MakeButton Option1
```

At run time, the checkbox and option button both appear as command buttons, except they toggle up or down depending on the Value property. Try clicking on them to see the effect—you'll need a group of option buttons in a frame to change the original option button.

—Mike Lyons, Coquitlam, British Columbia, Canada

VB6

Level: Intermediate

Default Button Prevents Firing of Validate Event

VB6 introduced the Validate event as a way to handle field-level validation without using LostFocus, which doesn't always fire in the order you would expect. The Validate event is a great idea, but doesn't work correctly on forms with a Default button. If the user presses Enter instead of clicking on the default button, neither the Validate event nor the LostFocus event fire. If the user clicks on the button with the mouse, however, both events fire correctly. You can get around the LostFocus problem by entering this code in the first part of the button's Click event:

```
Private Sub cmdOK_Click()
    If Not Me.ActiveControl Is cmdOK Then
        cmdOK.SetFocus
        DoEvents
    End If
    ' [remainder of click processing]
End Sub
```

With this code, if the user presses Enter to pick the default, the focus changes to the default button before any other Click event code executes, causing whichever control currently has focus to fire its LostFocus event. You would expect the Validate event would also fire that in this situation, but it doesn't. So, for forms with default buttons, LostFocus is the better way to handle field-level validation.

—Gordon Lawson, Billings, Montana

VB3 and up

Level: Beginning

Generate Ordinal Strings Simply

Sometimes, you have an integer value in code that you'd like to display in a string as an ordinal. For example, a variable contains the number 3, and you want to display a message like "The 3rd item..." You can use this function to convert the number into a string with "st," "nd," "rd," or "th" attached properly:

```
Public Function GetOrdinal(ByVal Num As Long) As String
    Dim n As String
    ' Num is assumed to be greater than zero
    n = CStr(Num)
    Select Case Right$(n, 2)
        Case "11", "12", "13"
            GetOrdinal = n & "th"
        Case Else
            Select Case Right$(n, 1)
                Case "0", "4" To "9"
```

```
                GetOrdinal = n & "th"
            Case "1"
                GetOrdinal = n & "st"
            Case "2"
                GetOrdinal = n & "nd"
            Case "3"
                GetOrdinal = n & "rd"
        End Select
    End Select
End Function
```

—Thomas Weiss, Deerfield, Illinois

VB4/32, VB5, VB6
Level: Intermediate

Get Dropdown's hWnd Without Subclassing

I've written code to get the hWnd for a combo's dropdown. When a combo's DropDown event is fired, the dropdown is not yet visible on the screen. Post a WM_KEYDOWN to the combo, which causes the combo's KeyDown event to fire after the dropdown is visible. Then use the Win32 API calls ClientToScreen, WindowFromPoint, and GetClassName to locate the dropdown window. Once you have the dropdown's hWnd, you can move and resize the dropdown window using Win32 API calls such as SetWindowPos. You'll find this technique useful where the width of the combo is less then the width of the combo's longest item text. To shorten the code listing, I'm leaving to you the case where the dropdown is dropped above the combo:

```
Option Explicit
Private Type POINTAPI
    x As Long
    y As Long
End Type
Private Declare Function PostMessage Lib "user32" Alias _
    "PostMessageA" (ByVal hWnd As Long, ByVal wMsg As Long, _
    ByVal wParam As Long, ByVal lParam As Long) As Long
Private Declare Function WindowFromPoint Lib "user32" _
    (ByVal xPoint As Long, ByVal yPoint As Long) As Long
Private Declare Function ClientToScreen Lib "user32" (ByVal _
    hWnd As Long, lpPoint As POINTAPI) As Long
Private Declare Function GetClassName Lib "user32" Alias _
    "GetClassNameA" (ByVal hWnd As Long, ByVal lpClassName As _
    String, ByVal nMaxCount As Long) As Long
Private Const WM_KEYDOWN = &H100
Private Const KEY_CODE_DROPDOWN = 256
Private Sub Combo1_DropDown()
    'This will cause Combo1_KeyDown to fire
    'after the DropDown is shown
    Call PostMessage(Combo1.hWnd, WM_KEYDOWN, KEY_CODE_DROPDOWN, 0)
End Sub

Private Sub Combo1_KeyDown(KeyCode As Integer, Shift As _
    Integer) Dim hwndDropdown As Long

    If KeyCode = KEY_CODE_DROPDOWN Then
        hwndDropdown = GetHwndDropdown(Combo1)
        Debug.Print hwndDropdown
    End If
End Sub

Private Function GetHwndDropdown(cbo As ComboBox) As Long
    Dim ptDropDown As POINTAPI
    Dim hwndDropdown As Long
    Dim sClassName As String
    Dim lRetLen As Long
    ptDropDown.x = (cbo.Width / 2) / Screen.TwipsPerPixelX
    ptDropDown.y = (cbo.Height * 1.1) / _
        Screen.TwipsPerPixelY
    Call ClientToScreen(cbo.hWnd, ptDropDown)
    hwndDropdown = WindowFromPoint(ptDropDown.x, _
        ptDropDown.y)
    sClassName = String$(255, Chr$(0))
    lRetLen = GetClassName(hwndDropdown, sClassName, _
        Len(sClassName))
    If lRetLen > 1 Then
        If Left$(sClassName, lRetLen) = "ComboLBox" Then
```

```
            GetHwndDropdown = hwndDropdown
        End If
    End If
End Function
```

—Mike Hill, Northridge, California

VB5, VB6
Level: Beginning

Enumerate Flags for Easier Coding

If you do a lot of work with the Windows API, you might notice that some API functions have flag-type parameters, and you usually pass API constants as the values for these parameters. Instead of putting multiple Public Const statements in a module the way the API Viewer gives them to you, you can group similar constants into enumerations and change the type of the parameter in the API function prototype to be the enumeration instead of a Long integer. This technique works only with Longs. The benefit is that the possible parameter values are displayed in the constant list as you're coding your API function calls.

—Thomas Weiss, Deerfield, Illinois

VB5, VB6
Level: Intermediate

Use System Icons for MsgBox Lookalikes

When VB's MsgBox function doesn't provide everything you need, you have to create a message-box–like form. You might find yourself in a quandary if you want to include one of the icons that normally displays in a Windows message box. Instead of getting a screenshot of a standard message box, editing it in Paintbrush to get only the 32-by-32 icon, and loading the resulting bitmap into an image box control on your form, you can use the Windows API to extract these icons from the system directly. Add these declarations to a module in your application:

```
Private Enum StandardIconEnum
    IDI_ASTERISK = 32516&    ' like vbInformation
    IDI_EXCLAMATION = 32515& ' like vbExlamation
    IDI_HAND = 32513&        ' like vbCritical
    IDI_QUESTION = 32514&    ' like vbQuestion
End Enum

Private Declare Function LoadStandardIcon Lib "user32" Alias _
    "LoadIconA" (ByVal hInstance As Long, ByVal lpIconNum As _
    StandardIconEnum) As Long
Private Declare Function DrawIcon Lib "user32" (ByVal hDC _
    As Long, ByVal x As Long, ByVal y As Long, _
    ByVal hIcon As Long) As Long
```

Then make your message box's Paint event look like this:

```
Private Sub Form_Paint()
    Dim hIcon As Long
    hIcon = LoadStandardIcon(0&, IDI_EXCLAMATION)
    Call DrawIcon(Me.hDC, 10&, 10&, hIcon)
End Sub
```

The LoadStandardIcon prototype is a tweaked version of the normal LoadIcon prototype, edited to use StandardIconEnum instead of a Long for the lpIconNum parameter.

—Thomas Weiss, Deerfield, Illinois

VB4, VB5, VB6
Level: Beginning

Position and Size Controls Using Keyboard

You can move controls using Ctrl with the arrow keys, and you can change control size using Shift with the arrow keys. The controls move or resize according to the Grid Width and Grid Height set in the Options dialog's General page. Unlike performing this task with the mouse, you can use this technique even when controls are locked. You'll find it more convenient when you must position and size controls accurately.

—Grace Li, Christchurch, New Zealand

VB4/32, VB5, VB6
Level: Intermediate

Query Objects for Initialization State

In a large app, or even a small one, you can use Property Let and Property Get to make sure necessary variables and subsystems are initialized. This code is from a large ADSI-based program in production:

```
Public Property Get ADSIInitialized() As Boolean
    If dso Is Nothing Then
        ADSIInitialized = False
    Else
        ADSIInitialized = True
    End If
End Property
Public Property Let ADSIInitialized(aValue As Boolean)
    If aValue = False Then ' Shut everything off
        Set cont = Nothing
        Set dso = Nothing
    Else
        ' Make sure we aren't already initialized
        If dso Is Nothing Then
        ' Turn everything back on
            Set dso = GetObject("WinNT:")
            Set cont = dso.OpenDSObject("WinNT://" _
                & Server, "", "", ADS_SECURE_AUTHENTICATION)
        End If
    End If
End Property
```

Now it's trivial to verify this component has been initialized and initialize it if necessary:

```
If Not ADSIInitialized Then ADSIInitialized = True
```

—Gregory Gadow, Seattle, Washington

VB6
Level: Intermediate

Use TreeView Control With Checkboxes

When the NodeCheck event triggers, you receive as a parameter the node that was checked. Say you need to do some validation and uncheck the node when there's an error. You set Node.Checked = False, right? Wrong! That unchecks the node until the NodeCheck event finishes, but at the end of the event, the node changes to its previous value. The reason for that is that the Node parameter is passed ByVal. To work around this problem, add a timer to your form (Interval=50, Enabled=False). Enable the timer in the NodeCheck event:

```
Dim mNode As Node
Private Sub Timer1_Timer()
    Timer1.Enabled = False
    mNode.Checked = False
    Set mNode = Nothing
End Sub
Private Sub TreeView1_NodeCheck(ByVal Node As MSComctlLib.Node)
    If Node.Checked Then
        '...If Invalid Then...
        MsgBox "This Node Cannot be Checked."
        Set mNode = Node
        Timer1.Enabled = True
    End If
End Sub
```

—Gerardo Villeda, Drexel Hill, Pennsylvania

VB4, VB5, VB6
Level: Intermediate

Treat a Form Like a Function

Some forms are merely dialog boxes that show something to the user and sometimes get something in return. For example, you might have to create a form that displays a tabulated listbox of information—contacts, for example. The form needs to know which item should be selected initially, and you want to know which item the user chose in the end. You can share this information through public variables, but wrapping the form into a function proves a better way. Create a standard EXE project and add an extra form to it. Place a command button on the first form and a listbox with a button on the second. Place this code in the first form:

```
Private Sub Command1_Click()
```

```
    MsgBox Form2.ShowList(4)
End Sub
```

Place this code in the second form:

```
Dim iSelectedIndex
Private Sub Command1_Click()
    Me.Hide
End Sub
Private Sub Form_Load()
    Dim i As Long
    For i = 0 To 20
        List1.AddItem "Item " & i
    Next i
    List1.ListIndex = iSelectedIndex
End Sub

' This is where the magic happens. Note that we have to
' display a modal form to prevent the continuation of this
' function until we are ready.
Public Function ShowList(Initial As Integer) As String
    iSelectedIndex = Initial
    ' Store the parameter for later use
    Me.Show vbModal
    ' Display the form
    ShowList = List1.List(List1.ListIndex)
    ' Return
    Unload Me
End Function
```

This technique not only lets you avoid using public variables, but it also gives you excellent portability because you can simply copy the form into a different project. None of the code is affected. You have wrapped a form into a function.

—Konstantin Komissarchik, Brier, Washington

VB3 and up

Level: Intermediate

Functions Parse Command Lines

Handling multiple command-line arguments has always been ugly in VB, especially when some of the arguments are quoted because they contain characters such as spaces. For example, if you want to write a program that takes as an argument a filename, you must quote the filename to ensure a space inside it doesn't confuse your application. Unfortunately, there's no built-in functionality for handling this mess. Here are two functions—GetParam and GetParamCount—that I use all the time. Each can handle a mix of quoted and unquoted parameters:

```
Public Function GetParam(ByVal Count As Integer) As String
    Dim i As Long
    Dim j As Integer
    Dim c As String
    Dim bInside As Boolean
    Dim bQuoted As Boolean
    j = 1
    bInside = False
    bQuoted = False
    GetParam = ""
    For i = 1 To Len(Command)
        c = Mid$(Command, i, 1)
        If bInside And bQuoted Then
            If c = """" Then
                j = j + 1
                bInside = False
                bQuoted = False
            End If
        ElseIf bInside And Not bQuoted Then
            If c = " " Then
                j = j + 1
                bInside = False
                bQuoted = False
            End If
        Else
            If c = """" Then
```

```
                If j > Count Then Exit Function
                bInside = True
                bQuoted = True
            ElseIf c <> " " Then
                If j > Count Then Exit Function
                bInside = True
                bQuoted = False
            End If
        End If

        If bInside And j = Count And c <> """" _
            Then GetParam = GetParam & c
    Next i
End Function

Public Function GetParamCount() As Integer
    Dim i As Long
    Dim c As String
    Dim bInside As Boolean
    Dim bQuoted As Boolean
    GetParamCount = 0
    bInside = False
    bQuoted = False
    For i = 1 To Len(Command)
        c = Mid$(Command, i, 1)
        If bInside And bQuoted Then
            If c = """" Then
                GetParamCount = GetParamCount + 1
                bInside = False
                bQuoted = False
            End If
        ElseIf bInside And Not bQuoted Then
            If c = " " Then
                GetParamCount = GetParamCount + 1
                bInside = False
                bQuoted = False
            End If
        Else
            If c = """" Then
                bInside = True
                bQuoted = True
            ElseIf c <> " " Then
                bInside = True
                bQuoted = False
            End If
        End If
    Next i

    If bInside Then GetParamCount = GetParamCount + 1
End Function
```

—Konstantin Komissarchik, Brier, Washington

VB4/32, VB5, VB6
Level: Intermediate

Account for Taskbars When Centering Forms

Most VB programmers must display a form centered on a screen. You can do this in a variety of ways, but most ignore aspects of the environment such as the taskbar or the office launchbar. This function takes these aspects into account to center a form within the client area perfectly:

```
Private Declare Function GetSystemMetrics Lib "user32" _
    (ByVal nIndex As Long) As Long
Private Declare Function GetWindowLong Lib "user32" _
    Alias "GetWindowLongA" (ByVal hwnd As Long, _
    ByVal nIndex As Long) As Long
Private Const SM_CXFULLSCREEN = 16
Private Const SM_CYFULLSCREEN = 17
Public Sub CenterForm(Frm As Form)
    Dim Left As Long, Top As Long
    Left = (Screen.TwipsPerPixelX _
        * (GetSystemMetrics(SM_CXFULLSCREEN) / 2)) - _
        (Frm.Width / 2)
    Top = (Screen.TwipsPerPixelY * _
```

```
        (GetSystemMetrics(SM_CYFULLSCREEN) / 2)) - _
        (Frm.Height / 2)
    Frm.Move Left, Top
End Sub
```

—Konstantin Komissarchik, Brier, Washington

VB3 and up
Level: Beginning

Enhance the Trim Function

The Trim function has a serious shortcoming: It handles only space characters—not all the usual white spaces such as tabs, carriage returns, and line feeds. Instead of the standard Trim function, use my TrimAll function, which handles all white spaces. In fact, you can extend it to trim off any character by editing the assignment to the ToEliminate string variable:

```
Public Function TrimAll(ToTrim As String) As String
    Static ToEliminate As String
    Dim Start As Long, Finish As Long
    ' Base condition test
    If Len(ToTrim) = 0 Then
        TrimAll = ""
        Exit Function
    End If
    ' Define the characters (once) that we want to trim off
    If Len(ToEliminate) = 0 Then
        ToEliminate = Chr(0) & Chr(8) & Chr(9) _
            & Chr(10) & Chr(13) & Chr(32)
    End If
    ' Find the beginning of nonblank string by checking
    ' each char against a list of blank chars.
    Start = 1
    Do While Start <= Len(ToTrim)
        If InStr(ToEliminate, Mid$(ToTrim, Start, 1)) Then
            Start = Start + 1
        Else
            Exit Do
        End If
    Loop
    ' Find the end of nonblank string.
    Finish = Len(ToTrim)
    Do While Finish > 1
        If InStr(ToEliminate, Mid$(ToTrim, Finish, 1)) Then
                Finish = Finish - 1
        Else
            Exit Do
        End If
    Loop
    If Start > Finish Then
        ' If the string is completely blank,
        ' Start will be greater than Finish.
        TrimAll = ""
        Exit Function
    Else
        ' Trim out the real contents
        TrimAll = Mid$(ToTrim, Start, Finish - Start + 1)
    End If
End Function
```

—Konstantin Komissarchik, Brier, Washington

VB3 and up
Level: Beginning

Format Names Consistently

People's names come in many separate parts, some of which might not be present or known. The hassle begins when you're dealing with a storage system—database or otherwise—where the parts are stored separately. You're faced with the formidable task of putting it all together with correct formatting. A common mistake is formatting a person's name whose middle initial is not known: John . Doe instead of John Doe. Using these functions, you can correct this problem without repeating effort:

```
Public Function FormatName(firstname As String, lastname As _
    String, Optional mi As String, Optional title As String, _
```

```
    Optional Suffix As String) As String
    Dim sRet As String
    If Len(Trim$(title)) > 0 Then
        sRet = StrConv(title, vbProperCase)
        If Right$(sRet, 1) <> "." Then sRet = sRet & "."
        sRet = sRet & " "
    End If
    If Len(Trim$(firstname)) > 0 Then
        sRet = sRet & StrConv(firstname, vbProperCase) & " "
    End If
    If Len(Trim$(mi)) > 0 Then
        sRet = sRet & StrConv(mi, vbProperCase)
        If Right$(sRet, 1) <> "." Then sRet = sRet & "."
        sRet = sRet & " "
    End If
    If Len(Trim$(lastname)) > 0 Then
        sRet = sRet & StrConv(lastname, vbProperCase) & " "
    End If
    If Len(Trim$(Suffix)) > 0 Then
        sRet = Trim$(sRet) & ", " & StrConv(Suffix, vbProperCase)
    End If
    FormatName = Trim$(sRet)
End Function
```

The next function resembles the previous one, except that it puts the last name first:

```
Public Function FormatNameReverse(firstname As String, _
    lastname As String, Optional mi As String) As String
    Dim sRet As String
    sRet = StrConv(lastname, vbProperCase)
    If Len(Trim$(firstname)) > 0 Or Len(Trim$(mi)) > 0 _
        Then
        sRet = sRet & ","
    End If
    If Len(Trim$(firstname)) > 0 Then
        sRet = sRet & " " & Trim$(StrConv(firstname, _
            vbProperCase))
    End If
    If Len(Trim$(mi)) > 0 Then
        sRet = sRet & " " & Trim$(StrConv(Left$(mi, 1), _
            vbProperCase)) & "."
    End If
    FormatNameReverse = Trim$(sRet)
End Function
```

—Konstantin Komissarchik, Brier, Washington

VB3 and up

Level: Intermediate

Wrap I/O for Text Files

In a production application, every time you want to access a file for reading or writing, you must retrieve a free handle using the FreeFile() function to ensure you don't overwrite an existing handle. You also must remember to close the file after you finish with it. In some cases, you can avoid the trouble by encasing this functionality inside utility functions. For example, I wrote these two functions to go between strings and text files in my apps:

```
Public Function ReadFile(FileName As String) As String
    Dim hFile As Integer
    hFile = FreeFile
    On Error GoTo ErrorTrap
        Open FileName For Input As #hFile
        ReadFile = Input(LOF(hFile), hFile)
        Close # hFile
    Exit Function
ErrorTrap:
    ReadFile = ""
End Function
Public Sub WriteFile(FileName As String, Contents As    _
    String)
    Dim hFile As Integer
    hFile = FreeFile
    On Error Resume Next
    Open FileName For Output As #hFile
```

```
    Print #hFile, Contents;
    Close #hFile
End Sub
```

Once you put these functions in your project, you can read and write text files quickly. For example, here's a way you might use these functions to copy text files:

```
Call WriteFile("c:\b.txt", ReadFile("c:\a.txt"))
```

—Konstantin Komissarchik, Brier, Washington

VB6
Level: Beginning

Enhance the Replace Function

If you're faced with a string that needs to have certain characters removed from it, use the Replace() function to make the problem more manageable. For instance, use this code to remove all a's from a particular string:

```
Debug.Print Replace("abababa", "a", "")
```

This statement works fine when you want only a single character removed, but if you have a long list of suspects, you have to do serious copy-and-paste. Avoid that by using this function:

```
Public Function StripOut(ByVal From As String, _
    ByVal What As String) As String
    Dim i As Integer
    For i = 1 To Len(What)
        From = Replace(From, Mid$(What, i, 1), "")
    Next i
    StripOut = From
End Function
```

Just place this code somewhere in your program—preferably in a module—and call it like this:

```
Debug.Print StripOut("abcdefg", "bdf")
```

This call returns a string with all b, d, and f characters removed.

—Konstantin Komissarchik, Brier, Washington

VB3 and up
Level: Beginning

Create a Safer Mid Function

If you often write complex string-parsing and manipulation algorithms, the last thing you want is to add more checks to ensure your string positions are not negative. Avoid the hassle by using this function when you need to use Mid. It wraps around native VB functionality and handles this common error case:

```
Public Function FlexiMid(From As String, ByVal Start _
    As Long, Optional Length As Long = -1) As String
    If Start < 1 Then Start = 1
    If Length = -1 Then ' they want the rest of it
        FlexiMid = Mid$(From, Start)
    Else ' just give what they want
        FlexiMid = Mid$(From, Start, Length))
    End If
End Function
```

Once you paste this function into your program—I recommend a module, so you can access it from anywhere—you can use it as you would Mid. In fact, once I wrote this, I ran a search-and-replace on my project to start using it throughout.

—Konstantin Komissarchik, Brier, Washington

VB4/32, VB5, VB6
Level: Advanced

Use Screen-Saver Passwords

When you write a screen saver in C and the Windows SDK, a static library (SCRNSAVE.lib) allows you to create custom dialogs to change and request the password. But in VB you can't use this library. If you don't

want to create forms to replace the custom dialogs, use these two undocumented functions:

```
Declare Sub PwdChangePassword Lib "mpr.dll" Alias _
    "PwdChangePasswordA" (ByVal lpcRegkeyname As String, _
    ByVal hWnd As Long, ByVal uiReserved1 As Long, ByVal _
    uiReserved2 As Long)
Declare Function VerifyScreenSavePwd Lib _
    "password.cpl" (ByVal hWnd As Long) As Boolean
```

PwdChangePassword is in MPR.dll, the Multiple Provider Router. It does all the password management associated with Regkeyname, including popping up a dialog—as a child of hWnd. The two reserved parameters should be zero. VerifyScreen-SavePwd, in password.cpl, pops up a dialog box as a child of hWnd, prompting for the screen saver's password. If the user gets it wrong, it prints a message saying so and prompts for the password again. If the user presses OK, VerifyScreenSavePwd returns True; if the user presses Cancel, it returns False. These calls—and the DLLs—exist in Windows 95/98 but not in NT because NT handles password management at the system level. Here's how you can use the PwdChangePassword call:

```
Private Sub cmdChange_Click()
    PwdChangePassword "SCRSAVE", Me.hWnd, 0, 0
End Sub
```

You must use the string "SCRSAVE" as the first parameter to PwdChangePassword, because it has special meaning and the function fails if another string is passed. Call VerifyScreenSavePwd on detection of mouse or keyboard activity, passing the hWnd the dialog should be owned by. Here's a simple example of how to test this function:

```
Private Sub cmdTest_Click()
    Dim bRes As Boolean
    bRes = VerifyScreenSavePwd(Me.hWnd)
    MsgBox bRes
End Sub
```

—Marco Bellinaso, Treviso, Italy

VB3 and up
Level: Beginning

Write an IsTime Function

Use this function to determine whether a string represents a valid time:

```
Public Function IsTime(sTimeArg As String) As Boolean
    IsTime = IsDate(Format(Date, "short date") & _
        " " & sTimeArg)
End Function
```

—Geir Villmones, Mosjøen, Norway

VB3 and up
Level: Beginning

Turn a Textbox or Label Into a Marquee

Sometimes you need to display information longer than the biggest textbox or label control you can have onscreen. I've written a routine that displays a textbox's or label's contents in marquee style, with the text moving from right to left. Animate your controls by adding a timer to the form and setting its Interval property to 250 (to update the display four times per second). On each timer tick, pass the control you want to animate to the ShiftChars routine. This routine works by reading the control's contents, shifting the first character to the end, and reassigning the contents:

```
Private Sub Timer1_Timer()
    Call ShiftChars(Text1)
    Call ShiftChars(Label1)
End Sub
Private Sub ShiftChars(ctl As Control)
    Dim Buffer As String
    Select Case TypeName(ctl)
        Case "TextBox", "Label"
            ' Rely on default property to accept/return contents.
            Buffer = ctl
            If Len(Buffer) > 1 Then
                ctl = Mid$(Buffer, 2) & Left$(Buffer, 1)
            End If
```

```
    End Select
End Sub
```

For a more natural display, make sure your text strings have a trailing space character.

—Rafat Sarosh, Tacoma, Washington

VB4/32, VB5, VB6
Level: Intermediate

Test for Illegal Characters

Use this fast function to test for the occurrence of nonalphanumeric characters in a string:

```
Private Declare Function StrSpn Lib "SHLWAPI" Alias _
    "StrSpnW" (ByVal psz As Long, ByVal pszSet As Long) As Long
Public Function IsAlphaNum(ByVal sString As String) As Boolean
    Dim lPos As Long
    Const ALPHA_NUM As String = "abcdefgihjklmnopqrstuvwxyz" & _
        "ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890"
    ' Returns the first occurrence of nonmatching characters
    lPos = StrSpn(StrPtr(pString), StrPtr(pAlphaNum))
    ' If the return position is not equal to the length of the
    ' input string, nonalphanumeric chars were found.
    IsAlphaNum = (lPos = Len(sString))
End Function
```

You can easily modify this function to scan for invalid characters by editing the ALPHA_NUM constant so it includes only characters you consider legal.

Editor's Note: The StrSpn function relies on the version of shlwapi.dll that ships with Internet Explorer 4.0 and later. Handle errors—and expectations—accordingly.

—Geir Arnesen, Oslo, Norway

VB3 and up
Level: Beginning

Use an Easier Autohighlight Method

Most of us have a routine we call to autohighlight the entire contents of a textbox when it receives focus. And most of us type the name of the textbox when we pass it to this routine. Instead of typing the control's name in each GotFocus event, you can use this sub to highlight the currently active textbox on the passed form. Place this code in a module:

```
Sub HiLite(frm As Form)
    frm.ActiveControl.SelStart = 0
    frm.ActiveControl.SelLength = _
        Len(frm.ActiveControl.Text)
End Sub
```

Then place this line of code in your textbox control's GotFocus() event:

```
Private Sub Text1_GotFocus()
    HiLite Me
End Sub
```

Using this code, you can forget about typing your controls' names over and over.

—Achim Wengeler, San Jose, California

VB5, VB6
Level: Beginning

Remove Unused Controls From Projects

If you usually load many controls into your VB project during development, you often have controls loaded that aren't used any more by the time the project's finished. If the controls were added to the project, they are stored in the VBP file regardless of whether they're used. Here's a quick way to clean up your VBP files so only the controls really being used are stored in your project: Open your VB project. Open your component window under the Project | Components menu or right-click on your toolbox and select Components. Hold down the Shift key and click on OK. All components not being used will be unclicked and removed from your project. VB6 developers: You must not eliminate references to controls you create at run time with the Controls.Add method.

—Mike Ellis, Charlotte, North Carolina

VB4/32, VB5, VB6
Level: Intermediate

Save Forms' Size and Location at Run Time
You've noticed how some apps display forms and toolboxes in the same location and size as when you last closed them. Here's some simple code that gives your VB app the same effect by using the Registry. First, fill in an appropriate Tag property for your form at design time—something like Main Application Form or Color Tool Box. Then keep a global string constant called ApplicationName that holds the title for your application. It's used here to distinguish the Registry key, but it can also be used for error messages. Place this line in a module:

```
Public Const ApplicationName = "My Application Name"
```

Finally, place this code in a module:

```
Public Sub SaveFormDisplaySettings(frm As Form)
    If frm.Tag = "" Then Exit Sub
    SaveSetting ApplicationName, frm.Tag & _
        " Display Settings", "Top", Str(frm.Top)
    SaveSetting ApplicationName, frm.Tag & _
        " Display Settings", "Left", Str(frm.Left)
    SaveSetting ApplicationName, frm.Tag & _
        " Display Settings", "Height", Str(frm.Height)
    SaveSetting ApplicationName, frm.Tag & _
        " Display Settings", "Width", Str(frm.Width)
End Sub

Public Sub LoadFormDisplaySettings(frm As Form)
    Dim FormSettings As Variant
    Dim intSettings As Integer
    If frm.Tag = "" Then Exit Sub
    FormSettings = GetAllSettings(ApplicationName, frm.Tag & _
        " Display Settings")
    If IsEmpty(FormSettings) Then Exit Sub
    For intSettings = LBound(FormSettings, 1) _
        To UBound(FormSettings, 1)
    Select Case FormSettings(intSettings, 0)
        Case "Left"
            frm.Left = Val(FormSettings(intSettings, 1))
        Case "Top"
            frm.Top = Val(FormSettings(intSettings, 1))
        Case "Height"
            frm.Height = Val(FormSettings(intSettings, 1))
        Case "Width"
            frm.Width = Val(FormSettings(intSettings, 1))
    End Select
    Next intSettings
End Sub
```

Add this line to the Form_Load events of the forms you want to save:

```
Call LoadFormDisplaySettings(Me)
```

Add this line to the Form_Unload events:

```
Call SaveFormDisplaySettings(Me)
```

Note one side effect: These Registry settings remain in the Registry even after the application has been uninstalled. They're stored at or below HKEY_CURRENT_USER\Software\VB and VBA Program Settings\My Application Name\.

—Kevin Fizz, Reading, Pennsylvania

VB3 and up
Level: Intermediate

Multiply Conditions for Boolean Result
You often have to test for multiple conditions when enabling a confirmation button or other control that commits user input. Instead of using complex If...ElseIf statements or inline If functions, you can manage

multiple conditions by multiplying the many conditions together. This way, any condition that hasn't been met evaluates to zero and the rules of multiplication will keep your confirmation control disabled. For example, assume you have two textboxes that must contain text before enabling a command button. Call a common subroutine in each textbox's Change event:

```
Private Sub Text1_Change()
    Call DoEnables
End Sub
Private Sub Text2_Change()
    Call DoEnables
End Sub
```

In the common subroutine, set the command button's Enabled property:

```
Private Sub DoEnables
    cmdOk.Enabled = Len(Trim$(Text1)) * Len(Trim$(Text2))
End Sub
```

Also, adding new conditions is a simple task. Simply include the new condition into the series of multiplication:

```
Private Sub TextMustBeGreaterThan5_Change()
    Call DoEnables
End Sub
Private Sub DoEnables()
    cmdOk.Enabled = Len(Trim$(Text1)) * Len(Trim$(Text2)) * _
        (Val(TextMustBeGreaterThan5) > 5)
End Sub
```

You can add as many conditions as you need. Any condition not met evaluates to zero before being included into the equation. Make sure to enclose your logical operators in parentheses. Even one zero results in zero for the entire calculation, which VB treats as False for the Enabled property. If all conditions are met, you get a nonzero number, which VB treats as True.

—Larry Kehoe, Rio Rancho, New Mexico

VB4/32, VB5, VB6
Level: Intermediate

Send a Click Message

Recently, I turned to Windows messaging to manipulate certain dialogs by simulating button clicks programmatically. I looked through my API references and found only the WM_LBUTTONDOWN and WM_LBUTTONUP messages. I couldn't get them to work until I found, on the MSDN Web site, a message that's not documented in the API text that comes with VB—BM_CLICK = &HF5. You set lParam and wParam both to zero to use this message. It works perfectly when it's sent directly to the button. SendMessage is a synchronous call. If the button you want to click might take some time to process its work, and you'd rather make an asynchronous click, use PostMessage instead. This sample shows how to use the BM_CLICK message. Paste this code into a new form, with two command buttons, one option button, and one checkbox:

```
Option Explicit
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
Private Declare Function PostMessage Lib "user32" Alias _
    "PostMessageA" (ByVal hWnd As Long, ByVal wMsg _
    As Long, ByVal wParam As Long, ByVal lParam As Long) _
    As Long
Private Const BM_CLICK = &HF5
Private Sub Check1_Click()
    Debug.Print "  Check1_Click"
End Sub
Private Sub Command1_Click()
    Debug.Print "  Command1_Click"
End Sub
Private Sub Command2_Click()
    Debug.Print "Entering Command2_Click"
    Call SendMessage(Check1.hWnd, BM_CLICK, 0, ByVal 0&)
    Call SendMessage(Option1.hWnd, BM_CLICK, 0, ByVal 0&)
    Call SendMessage(Command1.hWnd, BM_CLICK, 0, ByVal 0&)
    Debug.Print "Exiting Command2_Click"
```

```
End Sub
Private Sub Option1_Click()
    Debug.Print "  Option1_Click"
End Sub
```

The BM_CLICK message works on any button-class control. This includes option buttons and checkboxes.

—Marc Boorshtein, Framingham, Massachusetts

VB4/32, VB5, VB6
Level: Intermediate

Prevent Duplicate Listbox Entries

This useful code listing prevents users from entering duplicate entries into a listbox or modifying existing listbox entries. While the code prevents users from adding duplicate entries, you can modify the True condition of the If block in the Add_New procedure to suit your needs. Declare the constant, LB_FINDSTRINGEXACT and the SendMessage function declaration in a BAS module:

```
Declare Function SendMessageByString Lib _
    "user32" Alias "SendMessageA" (ByVal hWnd As _
    Long, ByVal wMsg As Long, ByVal wParam As _
    Long, ByVal lParam As String) As Long
Public Const LB_FINDSTRINGEXACT = &H1A2
```

This function uses the SendMessageByString API and returns either the existing item's index or -1. If it returns -1, you can add the item:

```
Function ChkListDuplicates(chwnd As Long, _
    StrText As String) As Boolean
    ChkListDuplicates = (SendMessageByString(chwnd, _
        LB_FINDSTRINGEXACT, -1, StrText) > -1)
End Function
```

If the LB_FINDEXACTSTRING message returns a value of -1, no match was found, so ChkListDuplicates returns False. You can use this value to determine whether to add a new item to your list:

```
Private Sub Add_New()
    Dim user As String
    user = InputBox("Add ListBox Entry", "Test Project")
    If Len(user) Then
        If Not ChkListDuplicates(List1.hWnd, _
            Trim(user)) Then
            List1.AddItem Trim(user)
        Else
            MsgBox "Duplicate Names can not be " & _
                "added." & vbCrLf & "Please " & _
                "make sure you are not adding " & _
                "duplicate names.", vbExclamation, _
                "Test Project: Invalid Operation"
        End If
    End If
End Sub
```

—Kedar Sathe, Houston, Texas

VB4, VB5, VB6
Level: Intermediate

Load UI Graphics From the Resource File

Many VB programmers haven't harnessed the power of VB6's resource editor. They still use traditional LoadPicture or other primitive calls to load bitmaps and icons. Before VB5, loading pictures into controls was somewhat harder, because of the inherent troubles associated with specifying the path and filename of the resource.With the resource editor, it's easy to store icons, strings, and bitmaps in a single RES file. I had to load the same picture for several CommandButton controls in a current project. I set the Style property of the CommandButton to 1 (Graphical) and the Tag property to "calendar" so my app would know to load calendar.bmp. I then added and saved this BMP in a resource file and set the ID to "calendar."

First paste this code into the general section of a form:

```
'loads icons/bmps from resource files.
Sub FillPictures(psdFrm As Form)
    ' desired form is passed as an parameter.
    Dim lcl_Ctrl As Control
```

```
    For Each lcl_Ctrl In psdFrm.Controls
        ' controls collection is used here.
        If LCase(lcl_Ctrl.Tag) = "calendar" Then
        ' checking the tag property.
            Set lcl_Ctrl.Picture = _
                LoadResPicture(lcl_ctrl.tag, vbResBitmap)
        End If
        ' extra code could have been added for
        ' loading other picture files by setting
        ' the tag property accordingly.
    Next
End Sub
```

Then call this function from the Form_Load event:

```
Private Sub Form_Load()
    Call FillPictures(Me)
End Sub
```

—Jishnu Bhattacharya, Jersey City, New Jersey

VB3 and up
Level: Intermediate

Convert a Decimal Number to Base N

Here's a function that converts a decimal number (base 10) to another base number system. Each digit position corresponds to a power of N, where N is a number between 2 and 36. In other words, if a number system's base is N, then N digits are used to write numbers in that system. For example, the base 2 number system (binary) uses the digits 0 and 1, while the base 20 system uses digits 0 through K.

The ConvertDecToBaseN function accepts a double-value decimal number and a byte-value representing the base number between 2 and 36. By default, the base value used is 16 (hexadecimal). The decimal number is converted to a positive number if it's negative. This function is useful for representing large numbers as strings, using fewer digit positions. I developed it to help reduce the footprint of several large numbers used in constructing a 16-character unique string ID. (Creating a complementary function to convert a base N number back into a decimal would be a great exercise.)

```
Public Function ConvertDecToBaseN(ByVal dValue As Double, _
    Optional ByVal byBase As Byte = 16) As String
    Const BASENUMBERS As String = _
        "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    Dim sResult As String
    Dim dRemainder As Double
    On Error GoTo ErrorHandler
    sResult = ""
    If (byBase < 2) Or (byBase > 36) Then GoTo Done
    dValue = Abs(dValue)
Do
    dRemainder = dValue - (byBase * Int((dValue / byBase)))
    sResult = Mid$(BASENUMBERS, dRemainder + 1, 1) & sResult
    dValue = Int(dValue / byBase)
Loop While (dValue > 0)
Done:
    ConvertDecToBaseN = sResult
    Exit Function
ErrorHandler:
    Err.Raise Err.Number, "ConvertDecToBaseN", _
        Err.Description
End Function
Sample usage:
ConvertDecToBaseN(999999999999#, 36)
'Returns 'CRE66I9R
```

—Peter Rodriguez, received by e-mail

VB3 and up
Level: Beginning

Tile With Lightweight Image Control Arrays

I write apps for companies where computer know-how is at an all-time low. Most applications scare people to death. However, Web-style forms and graphic buttons present soothing enough interfaces that they can be inviting to users.

I wanted my forms to look just like a Web page, so I devised a method to replicate and tile one Image control across a form. To achieve this effect, add an Image control to your form, set its Index to zero, set

Visible to False, and set an appropriate background image for the Picture property. Pass your form to the CreateBackground routine during its Load event:

```
Private Sub Form_Load()
    Call CreateBackground(Me)
End Sub
Public Sub CreateBackground(frmCallingParent)
    Dim lTilesY As Long, lTilesX As Long
    Dim lLeft As Long, lTop As Long
    Dim oImg As Image
    Dim X As Integer
    Set oImg = frmCallingParent.Image1(0)
    lTilesX = oImg.Width
    lTilesY = oImg.Height
    Do While lTop < Screen.Height
        X = X + 1
        Load frmCallingParent.Image1(X)
        Set oImg = frmCallingParent.Image1(X)
        With oImg
            .Left = lLeft
            .Top = lTop
            .Visible = True
        End With
        lLeft = lLeft + lTilesX
        If lLeft > Screen.Width Then
            lLeft = 0
            lTop = lTop + lTilesY
        End If
    Loop
End Sub
```

—Craig R. Gray, Clinton Township, Michigan

VB6
Level: Intermediate

Use Components and the Internet for Easy Maintenance

I'm developing a database system for a distant customer with a slow Internet connection. The system imports data and generates reports. The problem with importing and generating reports is that, over time, both the import file's format and the report's layout change. I can't send the whole system to him because it takes several hours to download.

To ease maintenance, I've separated the importing and reporting functions in an ActiveX DLL component of "Internet downloadable" size. When the customer asks me to make a change, I change the code and publish the ActiveX DLL component with a Package and Deployment Wizard to my Web site. I then send my customer an email with a URL for the HTML file generated by the Package and Deployment Wizard. The updated importing and reporting functions are quickly downloaded and installed automatically on his system when he clicks on a hyperlink.

—Thomas U. Nielsen, Copenhagen, Denmark

VB4, VB5, VB6
Level: Intermediate

Implement a Context Stack

Isolating the source of an error in a method containing many nested method calls can be difficult. If you haven't written comprehensive error-handling code in every method and property you write, or if you propagate errors up the call chain, an error handler in a high-level method won't be able to identify whether a trapped error occurred in the high level method itself, or whether it's the result of an unhandled error encountered further down the call chain.

Tracking execution context can be a relatively painless strategy to isolate the source of an error. Do this by declaring a global object that implements a call stack used by each method in your project. I call this object an ErrorContext object. You use the object by pushing the context—the component.module.method name—as each method begins, and popping it off the stack as each method terminates normally. Methods without error handlers terminate immediately and pass control up the call stack when an error occurs. The procedure containing the error can be identified by checking the last context pushed onto the call stack.

The ErrorContext object I use for this purpose contains three methods and one read-only property:

- PushContext (called at the start of each method).
- PopContext (called at the normal termination of each method).
- Resynch (to resynchonize the stack after a runtime error).
- ErrContext (a read-only property that returns the last context string pushed onto the stack).

To use the object, declare a global instance of the object in a public module, and call PushContext at the beginning of each method and PopContext immediately before exiting the procedure. Use ErrContext to check the context string at the top of the error stack. Use Resynch to resynchronize the call stack after an error, because PopContext won't be called by the lower-level procedures. This sample code omits explicit declarations and some error-checking code:

```
ErrorContext Object:
Private Contextstack() As String
Sub PushContext(Context As String)
    upper = UBound(Contextstack) + 1
    ReDim Preserve Contextstack(upper)
    Contextstack(upper) = Context
End Sub
Sub PopContext()
    upper = UBound(Contextstack) - 1
    ReDim Preserve Contextstack(upper)
End Sub
Property Get ErrContext() As String
    upper = UBound(Contextstack)
    ErrContext = Contextstack(upper)
End Property
Sub Resynch(Context As String)
    Do While ErrContext <> Context
        PopContext
    Loop
End Sub
Private Sub Class_Initialize()
  ReDim Contextstack(0)
End Sub
```

Here's a sample of how you'd use the Error context object in a form module. First, declare the ErrorContext object in a standard module:

```
Public Erx as New ErrorContext
```

In each method and property of your project, use the PushContext and PopContext methods as you enter and exit each routine:

```
Option Explicit
Const mconModPath = "MyProject.MyForm."

Private Sub Command1_Click()
    Const Context = mconModPath & "Command1_Click"
    erx.PushContext Context
    On Error GoTo StdError
    MySub1  'call chain starts here
    erx.PopContext
Exit Sub
StdError:
    MsgBox "An error occurred in " & _
        erx.ErrContext & vbNewLine & Err.Number & _
        vbTab & Err.Description
    erx.Resynch Context
    'if the error occurred in a subprocedure, use resynch
    'to pop context until Context = erx.ErrContext
    erx.PopContext
    'make sure to use PopContext at each exit point.
End Sub
Sub MySub1()
    Const Context = mconModPath & "MySub1"
    erx.PushContext Context
    MySub2
    erx.PopContext
End Sub
Sub MySub2()
    Const Context = mconModPath & "MySub2"
    erx.PushContext Context
    Dim a As Long
    a = 1 / 0 ' force a divide by zero error.
    erx.PopContext
End Sub
```

The error handler in Command_Click correctly identifies that the division by zero error occurred in MySub2.

While this approach entails adding three lines to each procedure, it can significantly decrease the time necessary to trace deep into call chains to locate the source of an error.

—Josh Kramer, Los Angeles, California

VB6
Level: Advanced

Use the VB Response Object to Generate Dynamic HTML Pages

I researched Windows Script Components (WSC) and was interested in their capability to integrate with ASP. I tried integrating a VB DLL with ASP. I included a reference to the Microsoft Active Server Pages Object Library (ASP.dll) in the DLL and used the Response object to write my HTML. I was surprised to find it worked when I tested it. It meant I could create a DLL that had common reusable routines for creating lists and filling combo boxes. The possibilities are endless, all in compiled VB DLL code.

The DLL has a method called ShowRecordSet that has four parameters:

- StrConnectString (the database connect string)
- StrSQL (the SQL command to execute)
- StrHeading (a heading for the table)
- ASPResponse (the reference to the Response object of the ASP page)

The DLL connects to the database, executes the query, and writes out an HTML table with a header and detail line for each row in the recordset. Use this DLL CRShtml class code:

```
Option Explicit
'Color constants
Const PageBgColor As String = """#F5F5F5"""
Const TableBgColor As String = """#8F9FE9"""
Const ShadeBgColor As String = """#C9C9C9"""
Public Sub ShowRecordSet(strConnectString _
    As String, strSQL As String, strHeading _
    As String, ASPResponse As ASPTypeLibrary.Response)
'Purpose: This method connects to datasource, retrieves
'the recordset, and writes HTML output via the Response
'object. Must have a reference to the Microsoft Active
'Server Pages Object Library (ASP.dll)
Dim intDetailCount As Integer
Dim strRowBgColor As String
Dim cnn As ADODB.Connection
Dim rs As ADODB.Recordset
Dim fldField As ADODB.Field
    Set cnn = New ADODB.Connection
    Set rs = New ADODB.Recordset
    cnn.Open strConnectString
    Set rs = cnn.Execute(strSQL)
    With ASPResponse
        .Write "<TABLE CELLPADDING=3 BORDER=""0"">"
        .Write "<tr>"
        .Write "<td width=""100%"" height=""18" _
            " colspan = " & rs.Fields.Count & _
            " bgcolor=""#666699""><font SIZE=""4" _
            " FACE=""Verdana"" color=""#FFFFFF""> " _
            & strHeading & "</font></td>"
        .Write "</tr>"
        'create the column headings from the field names
        .Write " <TR>"
        For Each fldField In rs.Fields
            .Write "  <TD bgcolor=" & TableBgColor _
                & ">" & fldField.Name & "</TD>"
        Next
        .Write " </TR>"
        'process each record building rows in table
        intDetailCount = 1
        Do While Not rs.EOF
            .Write " <TR>"
            'Shade each alternating row
            If (intDetailCount Mod 2) = 0 Then
                strRowBgColor = ShadeBgColor
            Else
                strRowBgColor = PageBgColor
            End If

            For Each fldField In rs.Fields
```

```
                .Write "  <TD bgcolor =" & _
                    strRowBgColor & ">" & _
                    fldField.Value & "" & "</TD>"
            Next fldField
            .Write " </TR>"
            rs.MoveNext
            intDetailCount = intDetailCount + 1
        Loop
        'close the table and objects
        .Write "</TABLE>"
    End With
    rs.Close
    cnn.Close
    Set rs = Nothing
    Set cnn = Nothing
End Sub
```

Test Active Server Page:

```
<HTML>
<BODY bgcolor="#F5F5F5">
<H2>Using a Server-side VB DLL with a reference to
the Response Object</H2>
<%
Dim objRSServer
'Instantiate object
Set objRSServer = Server.CreateObject("RSServer.CRShtml")
'Call Method
objRSServer.ShowRecordSet "DRIVER={SQL
Server};SERVER=ServerName;DATABASE=DBName;
    UID=UserID;PWD=UserPwd;", "SELECT
    FirstName + ' ' + LastName as Employee,
     Phone, Email FROM tblEmployee ORDER BY
    LastName, FirstName", "Employee List",
    Response
%>
</BODY>
</HTML>
```

—Brian Barnett, Woodstock, Georgia

VB4, VB5, VB6
Level: Intermediate

Wrap Date Functions in a Class

I've been working on a class module called clsDateInfo that returns various properties of a given date, and using it in some monthly trend graphs. I had to come up with the number of weekdays—not counting weekends—a given date was from the first of the month. The clsDateInfo.WeekDayOfMonth property returns the answer in a flash:

```
Option Explicit
' clsDateInfo
' Chuck Spohr 9/23/1999
' Set the DateToCheck property of this object and
' the other properties will return various useful
' values about that date
Private mdtDate As Date
Public Property Let DateToCheck(pdtDate As Date)
    mdtDate = pdtDate
End Property
Public Property Get DateToCheck() As Date
    DateToCheck = mdtDate
End Property
Public Property Get WeekDayOfMonth() As Integer
    If Me.DayOfWeek = vbSunday Or Me.DayOfWeek = _
        vbSaturday Then
        WeekDayOfMonth = 0
    Else
        WeekDayOfMonth = (5 * (Me.WeekOfMonth - 1)) - _
            Me.FirstDayOfWeekOfMonth + Me.DayOfWeek + 1
    End If
End Property
Public Property Get WeekOfMonth() As Integer
    WeekOfMonth = Week  Me.FirstWeekOfMonth + 1
```

```
End Property

Public Property Get FirstWeekOfMonth() As Integer
    FirstWeekOfMonth = DatePart("ww", Me.FirstDayOfMonth)
End Property
Public Property Get FirstDayOfWeekOfMonth() As Integer
    FirstDayOfWeekOfMonth = DatePart("w", Me.FirstDayOfMonth)
End Property
Public Property Get Week() As Integer
    Week = DatePart("ww", mdtDate)
End Property
Public Property Get FirstDayOfMonth() As Variant
    FirstDayOfMonth = DateSerial(DatePart("yyyy", _
        mdtDate), DatePart("m", mdtDate), 1)
End Property
Public Property Get DayOfWeek() As Integer
    DayOfWeek = DatePart("w", mdtDate)
End Property
Private Sub Class_Initialize()
    mdtDate = Now()
End Sub
```

—Chuck Spohr, Balwin, Missouri

VB5, VB6
Level: Beginning

Enter Data on an MSFlexGrid

You can use MSFlexGrid for data entry without using additional ActiveX controls. For this, use the KeyPress and KeyUp events. To use the MSFlexGrid for data entry, add the grid—named FlxGrdDemo—to a form and copy this code:

```
Private Sub FlxGrdDemo_KeyPress(KeyAscii As Integer)
    Select Case KeyAscii
        Case vbKeyReturn
            ' When the user hits the return key
            ' this code'll move the next cell or row.
            With FlxGrdDemo
                If .Col + 1 <= .Cols - 1 Then
                    .Col = .Col + 1
                ElseIf .Row + 1 <= .Rows - 1 Then
                    .Row = .Row + 1
                    .Col = 0
                Else
                    .Row = 1
                    .Col = 0
                End If
            End With
        Case vbKeyBack
        ' Delete the previous character when the
        ' backspace key is used.
            With FlxGrdDemo
                If Trim(.Text) <> "" Then _
                    .Text = Mid(.Text, 1, Len(.Text) - 1)
            End With
        Case Is < 32
        ' Avoid unprintable characters.
        Case Else 'Else print everything
            With FlxGrdDemo
                .Text = .Text & Chr(KeyAscii)
            End With
    End Select
End Sub
Private Sub FlxGrdDemo_KeyUp(KeyCode As _
    Integer, Shift As Integer)
    Select Case KeyCode
        ' Copy
        Case vbKeyC And Shift = 2 ' Control + C
            Clipboard.Clear
            Clipboard.SetText FlxGrdDemo.Text
            KeyCode = 0
        ' Paste
        Case vbKeyV And Shift = 2 'Control + V
            FlxGrdDemo.Text = Clipboard.GetText
```

```
            KeyCode = 0
        ' Cut
        Case vbKeyX And Shift = 2 'Control + X
            Clipboard.Clear
            Clipboard.SetText FlxGrdDemo.Text
            FlxGrdDemo.Text = ""
            KeyCode = 0
        ' Delete
        Case vbKeyDelete
            FlxGrdDemo.Text = ""
    End Select
End Sub
```

You can set the FillStyle property to FlexFillRepeat, which makes the changes to all the selected cells.

—Srinivasa S. Sivakumar, Chicago, Illinois

VB6
Level: Advanced

Make Windowless, Transparent UserControls Clickable

Microsoft has documented a problem with windowless User-Controls that have a transparent Backstyle. Once a form contains such a control, you can't select it by clicking on it with the mouse; this makes it tough to move the control. (See Microsoft Knowledge Base article Q188234 for details.) Use this code workaround that allows you to click on and move these controls at design time. First, it uses the HitTest event to make the control always act as if it is clicked, regardless of mouse coordinates. This usage causes the UserControl_Click event to fire, which the owner can observe through the raised Click event:

```
Private Sub UserControl_HitTest(X As Single, Y _
    As Single, HitResult As Integer)
    ' Always act as if the control was hit
    If HitResult = vbHitResultOutside Then
        HitResult = vbHitResultHit
    End If
End Sub
Public Event Click()
Private Sub UserControl_Click()
    ' Let the form handle the click
    RaiseEvent Click
End Sub
```

In production code, if only portions of the control should be clickable in run mode, test for design mode vs. run mode in the HitTest event. Use this method only in design mode, and your own custom test in run mode.

—Don Benson, Hudson, Ohio

VB3 and up
Level: Intermediate

Convert Static Variables for More Speed

Referencing a static local variable in a procedure is two to three times slower than a regular local, dynamic variable. If your procedure demands every last bit of speed possible, convert all static variables into module-level variables. However, this approach has a nasty design implication—the procedure becomes less self-contained. You must remember to copy and paste the module-level variable when you reuse the procedure in another project. But this approach might make sense in an extremely intense routine. Further, referencing a variable declared at module level is faster than referencing a global variable declared in a separate BAS module. If you don't need to share a variable among all of an app's forms and modules, you should declare it in the only form or module that uses it.

—Jai Bardhan, Lowell, Massachusetts

VB3 and up
Level: Beginning

Change Dates With Plus and Minus Keys

This simple piece of code saves a lot of headaches when it comes to altering dates. It allows you to use the plus and minus keys to increment and decrement date values easily. This example assumes there is a textbox named Text1 on a standard VB form:

```
Private Sub Form_Load()
    ' Make sure that there is a valid date in Text1.
```

```
    Text1.Text = Date
End Sub

Private Sub Text1_KeyPress(KeyAscii As Integer)
    ' Pass handling to generic routine.
    KeyAscii = DateHandler(KeyAscii)
End Sub

Private Function DateHandler(KeyAscii As _
    Integer) As Integer
    Dim nRet As Integer
    ' This routine adds or subtracts days, based on the
    ' key pressed, from a date value found in the control
    ' represented by the form's ActiveControl property
    ' (usually a TextBox). The routine can be altered to
    ' add and subtract months and years too.
    On Error GoTo ErrorHandler
    ' Constants which represent the '+' & '-' keys.
    Const KeyAdd = 43
    Const KeySubtract = 45
    ' This constant is here because '+' & '=' are on the
    ' same key for most keyboards, but are sometimes inverted.
    Const KeyEquals = 61
' Determine the value of the key pressed, and
    ' take the necessary action.
    Select Case KeyAscii
        Case KeyAdd, KeyEquals
            Me.ActiveControl.Text = DateAdd("d", _
                1, Me.ActiveControl)
            nRet = 0
        Case KeySubtract
            Me.ActiveControl.Text = DateAdd("d", _
                -1, Me.ActiveControl)
            nRet = 0
        Case Else
            nRet = KeyAscii
    End Select
    ' Move the start position to the end of the
    ' text for a cleaner look.
    If nRet = 0 Then
        Me.ActiveControl.SelStart = Len(Text1.Text)
    End If

    ' Return a new KeyAscii value.
    DateHandler = nRet
    Exit Function

ErrorHandler:
    DateHandler = 0
    Exit Function
End Function
```

—Sheppe Pharis, Kelowna, British Columbia, Canada

VB6
Level: Beginning

Count the Occurrences of a Character or Substring

VB6 introduced the Split function, which returns a zero-based, one-dimensional array containing a specified number of substrings. Although this function is useful in itself, you can also use it in other useful ways. For example, by combining the UBound and Split functions, you can count how many times a substring—or single character—appears inside another string:

```
Function InstrCount(Source As String, _
    SearchString As String) As Long
    If Len(Source) Then
        InstrCount = UBound(Split(Source, SearchString))
    End If
End Function
```

—Gilberto Zayas Ramos, Velasco, Cuba

VB3 and up
Level: Beginning

Simplify Programatic Selection in Combos

Here's a useful procedure to position a ComboBox according to a value of the ItemData property or the List property. It's useful to position a ComboBox with values taken from a database, and this way, become independent of the index property. For example, you might fill a ComboBox with the serial port's baud speeds, including a description in List and the value in bauds in ItemData:

```
Public Sub LlenarCombo(pCombo As ComboBox)
    With pCombo
        .Clear
        .AddItem "1200 bps"
        .ItemData(0) = 1200
        .AddItem "2400 bps"
        .ItemData(1) = 2400
        .AddItem "4800 bps"
        .ItemData(2) = 4800
        .AddItem "9600 bps"
        .ItemData(3) = 9600
        .AddItem "14400 bps"
        .ItemData(4) = 14400
        .AddItem "28800 bps"
        .ItemData(5) = 28800
        .ListIndex = 0
    End With
End Sub
Public Sub PosicionarCombo(pCombo As _
    ComboBox, ByVal pValor As Variant)
    Dim i As Integer
    If IsNumeric(pValor) Then
        ' Search by ItemData
        For i = 0 To pCombo.ListCount - 1
            If pCombo.ItemData(i) = pValor Then
                pCombo.ListIndex = i
                Exit For
            End If
        Next i
    Else
        ' Search by List
        For i = 0 To pCombo.ListCount - 1
            If pCombo.List(i) = pValor Then
                pCombo.ListIndex = i
                Exit For
            End If
        Next i
    End If
End Sub
Private Sub cmdPosItemData_Click()
    PosicionarCombo cboTest, 9600
End Sub
Private Sub cmdPosList_Click()
    PosicionarCombo cboTest, "4800 bps"
End Sub
```

—Diego M. Basélica, Cordoba, Argentina

VB4, VB5

Level: Beginning

Duplicate the Split Function for VB4 and VB5

It's too bad Microsoft didn't create the Split function until VB6, but here's a function that duplicates it for VB4 and VB5 users. The only difference is that with VB4 and VB5, you must use a Variant to receive the Split data, whereas VB6 can also use a dynamic String array:

```
Public Function Split(aText As String, _
    Optional vSrch As Variant) As Variant
    If IsMissing(vSrch) Then vSrch = " "
    Dim j As Long, k As Long, a As String
    ReDim s(0) As String
    a = aText
    k = InStr(a, vSrch)
    Do While k
        If j > UBound(s) Then
            ReDim Preserve s(0 To j) As String
```

```
        End If
        s(j) = Left$(a, k - 1)
        a = Mid$(a, k + Len(vSrch))
        k = InStr(a, vSrch)
        j = j + 1
    Loop
    If Len(a) Then
        If j > UBound(s) Then
            ReDim Preserve s(0 To j) As String
        End If
        s(j) = a
    End If
    Split = s
End Function
Dim vDat As Variant
vDat = Split("This is a test")
' vDat(0) = "This"  vDat(1) = "is", etc...
```

—Matt Hart, Tulsa, Oklahoma

VB6
Level: Intermediate

Web-Enable Your Apps

In today's world, you probably want to let your users browse the Web inside your app. You can add the Microsoft Internet Control to do this; however, the user must have Internet Explorer installed. Without it, the app fails to start. To solve this problem, remove the Microsoft Internet Control reference and load the control dynamically when Internet Explorer is installed. To load the control, use this code:

```
Private ie As VBControlExtender
Private Sub Form_Load()
    On Error GoTo IEMissing
  Set ie = Form1.Controls.Add("Shell.Explorer", "wcIE")
    ie.Visible = True
IEMissing:
End Sub
Private Sub Form_Resize()
    If Not ie Is Nothing Then
        ie.Move 0, 0, Me.ScaleWidth, Me.ScaleHeight
    End If
End Sub
```

You can do multiple things with this object, such as change the visibility, but then the unique Internet Explorer properties and methods aren't available. For example, if you type "obj.Navigate sMyURL", VB tells you the object doesn't support this property or method. The secret is to use your object variable's Object property like this:

```
Private Sub Form_Activate()
    If Not ie Is Nothing Then
        ie.Object.Navigate "http://www.vbpj.com"
    End If
End Sub
```

—Eric Quist, Gothenburg, Sweden

VB3 and up
Level: Beginning

Link the DataField to the Recordset

The data control allows you to quickly link controls and databases; however, not only is it far from flexible compared with the database-objects coding interface, it doesn't look great. I use the latter solution and assume most VB programmers do. But it's quite painful to add or alter several lines of code every time you put a new text field on a dialog. It's a waste of the DataField property, however; it can be just as useful as the Tag property, and in this case, it's more descriptive. So what if you write a few routines to simulate the data control's basic operations through the DataField property? This simple routine loads data from a Recordset to all controls on a form:

```
Public Sub ReadData(frm As Form, rc as Recordset)
    Dim ctrl As Control
    ' Need to ignore errors on controls
    ' that don't support databinding.
    On Error Resume Next
```

```
    For Each ctrl In frm.Controls
        If ctrl.DataField <> "" Then
            ctrl = rc.Fields(ctrl.DataField)
        End If
    Next 'ctrl
End Sub
```

You only need to put the field name into the DataField property of involved controls at design time. By altering such routines, you can achieve more complex data handling than if you use data controls, as well as make reading, writing, and validation a lot simpler than doing everything manually.

—Martin Girard, Chicoutimi, Quebec, Canada

VB3 and up

Level: Intermediate

Use Loop Counters Even After Looping

The value of a loop counter variable is incremented one beyond the set range when the loop is completed. For example, if you use a For i = 0 to 5...Next loop, i equals 6 when all loop iterations have been completed. If you use the For...Next statement with an Exit For statement, you can use the value of the loop counter variable to determine whether a condition was met during the loop. Here's one possible application of this technique, which can be applied when you have an array of values, each element of which has a unique value:

```
Dim sText(10) As String
    Dim i As Long
    ' Initialize strings - A, B, C, D, ...
    For i = 1 To 10
        sText(i) = Chr$(Asc("A") + i)
    Next i
```

Suppose you want to find which element, if any, has a given value. You could use a Do...Until Loop and a flag:

```
Dim i As Long
    Dim IsFound As Boolean
    i = 0
    Do While Not IsFound
        i = i + 1
        If i > 10 Then Exit Do
        If sText(i) = "J" Then IsFound = True
    Loop
    If IsFound Then
        MsgBox "Found J as element: " & i
    Else
        MsgBox "Could not find J"
    End If
```

Or you can use a For...Next loop without a flag:

```
Dim i As Long
    For i = 1 To 10
        If sText(i) = "J" Then Exit For
    Next i
    If i <= 10 Then
        MsgBox "Found J at element: " & i
    Else
        MsgBox "Could not find J"
    End If
```

This search routine is shorter and faster in the second case.

—Tom Mcormick, Chelmsford, Massachusetts

VB4, VB5, VB6

Level: Beginning

Clear Form's Controls With Quick Loop

This code allows you to set a form's controls to a specific value. You can use it, for example, when you want to clear all textboxes in a form or when you want to uncheck all checkboxes:

```
Public Sub Clear(frm As Form)
    Dim ctl As Control
    For Each ctl In frm
        Select Case TypeName(ctl)
            Case "TextBox"
                ctl.Text = ""
            Case "CheckBox"
                ctl.Value = vbUnchecked
            Case Else
                ' handle others as needed
        End Select
    Next ctl
End Sub
```

—Daniel Augusto Ramírez Villasana, México City, México

VB3 and up
Level: Intermediate

Replace a String Within a String, Recursively

I recently needed a substring replacement function for inserting code into a module, by reading the code from a file. Unfortunately, in my case, commas are interpreted as delimiters, and the insertion requires a lot of post formatting. So, I replaced all the commas in the original file with a question mark. That way, when the file is inserted into a module, the ReplaceString function checks each line, and the question mark is replaced with a comma, then inserted into the module. I initially considered using the fConvert function published in "Remove Unwanted Characters" ["101 Tech Tips for VB Developers," Supplement to VBPJ, February 1999]. I compared the speed of the two functions, ReplaceString and fConvert, in a separate project, using the Windows API GetTickCount function. The recursive function is nearly four times faster than the For...Loop. In situations where a single character needs to be replaced with something different, it's a good way to go:

```
Public Function ReplaceString(strT As String) As String
    Dim iposn As Integer
    Dim strF As String
    Dim strR As String
    ' Function replaces one character with another. Using
    ' recursion if the character is found to check if any
    ' more such characters need to be replaced within
    ' the string. strT is the string in which a character
    ' or string in which replacement will take place.
    ' strF is the string which is to be replaced.
    ' strR the new or replacing string.
    strF = "?"
    strR = ","
    iposn = InStr(1, strT, strF)
    If iposn > 0 Then
        Mid(strT, iposn, 1) = strR
        strT = ReplaceString(strT)
    End If
    ReplaceString = strT
End Function
```

—Matthew Grygorcewicz, Athelstone, Australia

VB4/32, VB5, VB6
Level: Intermediate

Determine the File System Type

With the advent of the FAT32 file system, you might want to use VB to determine the type of file system being used for a particular drive. This example is set for the C drive; change the variable sDrive to test other drives. Run this routine; the variable sResult contains the file system name string:

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

```
    Dim sVolBuf As String * 255
    Dim sSysName As String * 255
    Dim lSerialNum As Long
    Dim lSysFlags As Long
    Dim lComponentLength As Long
    Dim lRes As Long
    lRes = GetVolumeInformation(Drive, sVolBuf, _
        255, lSerialNum, lComponentLength, _
        lSysFlags, sSysName, 255)
    If lRes Then
        WhichFileSystem = Left$(sSysName, _
            InStr(sSysName, Chr$(0)) - 1)
    Else
        WhichFileSystem = "<unknown>"
    End If
End Function
```

—Dan Verkade, Perris, California

VB4, VB5, VB6
Level: Beginning

Iterate MDIChild Forms

Sometimes it's useful to close—or perform some other common operation on—all the child forms in your MDI parent simultaneously. For example, when a user relogs on to a database system, all the old child windows' data comes from a previous logon, and you just want to close all child forms. Simply iterate the Forms collection, checking the TypeOf on each form before testing its MDIChild property:

```
Dim vForm As Variant
For Each vForm In Forms
    If Not TypeOf vForm Is MDIForm Then
        If vForm.MDIChild Then
            Unload vForm
        End If
    End If
Next  'vFrom
```

—Orville P. Chomer, Berwyn, Illinois

VB4/32, VB5, VB6
Level: Beginning

Force Slider to Specific Intervals

Try using a slider control in your UI if you're tired of combo boxes. Users find this control intuitive to understand and operate. For example, you might use the slider control to obtain values from a user in increments of $1,000. First, add the slider control to a form. Set the slider's Min and Max properties to the appropriate range for your app. Then, set the TickFrequency property equal to any interval of interest. Using the example, try setting the slider properties to: Min=1,000 and Max=10,000. Unfortunately, if a user drags the slider instead of clicking on it, values between the tick marks are returned. Here's a cool way to easily control this behavior. Place this code in the slider's Change event, substituting your control's name:

```
Private Sub slBWidth_Change()
    slBWidth = CInt(slBWidth.Value / slBWidth. _
        TickFrequency) * slBWidth.TickFrequency
End Sub
```

Now try it out to see how the control behaves. The result is the same as scaling the slider from 1 to 10, then multiplying by a scale factor for the value. The difference is that it now free-slides instead of jerking between ticks.

—Christopher K. Hausner, Sterling Heights, Michigan

VB3 and up
Level: Beginning

Use the MsgBox Wrapper to Replace OK

Replace all MsgBox calls that display only an OK message with the following OkMsg sub. It automatically defaults the icon to vbInformation, and the title to a global constant defined at startup. None to all of the parameters can be passed to override the defaults. Another advantage is that the OkMsg sub saves and restores the state of the mousepointer, so you don't get an hourglass outside the MsgBox:

```
Sub OkMsg(Optional sMsg As String = _
```

```
    "Press OK to Continue", Optional vIcon = _
    vbInformation, Optional sTitle As String = APPNAME)
    Dim iMouse As Integer
    iMouse = Screen.MousePointer
    Screen.MousePointer = vbDefault
    MsgBox sMsg, vIcon, sTitle
    Screen.MousePointer = iMouse
End Sub
```

To call this sub, use this syntax:

```
    OkMsg  "The Record is Saved"
    OkMsg "The date entered is out of range!", _
        vbExclamation, "INPUT ERROR"
```

Always declare the global constant:

```
    Global Const APPNAME = "Management System"
```

Note: VB3 users must change the optional parameters to required, and VB4 users must insert IsMissing checks to assign defaults to missing optional parameters.

—Fabio A. Mir Sr., Gaithersburg, Maryland

VB4/32, VB5, VB6

Level: Beginning

Sort Non-String Items in a ListView

Sorting ListView columns with numeric data can be a real pain. Nonstring sorting is possible with callbacks using custom comparison functions, but this method's drawback is that the synchronization between the display and ListItems collection is lost. It's easier and more reliable to simply provide sortable data. Normally a list with the values 1, 2, 3, 4, 10, and 20 sorts as 1, 10, 2, 20, 3, and 4—that's not very useful. A simple workaround is to "left-pad" the numeric data with spaces before setting the text value. Assume that you load a listview from a recordset with last name, first name, and salary:

```
Const MAX_WIDTH = 15
Dim szSpaces As String
Dim rs As Recordset
szSpaces = Space$(MAX_WIDTH)
Do Until rs.EOF
    With ListView1.ListItems.Add(, , rs("LastName"))
    .SubItems(1) = rs("FirstName")
    .SubItems(2) = Right$(szSpaces & _
        rs("Salary"), MAX_WIDTH)
    End With
    rs.MoveNext
Loop
```

Now setting the ListView's Sorted property sorts the Salary column in correct numerical order.

—Amir Liberman, Pembroke Pines, Florida

VB3 and up

Level: Beginning

Cache Properties for Repeated References

If you have to reference a control's property repeatedly, it's better to assign the property to a temporary variable, then use that variable. This technique is called property caching. For example, if you need to assign text value in the Text1 textbox to all elements in an array named arr, it's better to assign the value to a temporary variable and use it for the assignment in the loop:

```
tmp = Text1.Text
For i = 1 To Ubound(arr)
    arr(i) = tmp
    ' Here use tmp instead of using Text1.Text repeatedly
Next i
```

—Jai Bardhan, Lowell, Massachusetts

VB6

Level: Intermediate

Load a Combo Box Array With a Compound Recordset in One Call

This code takes an array of combo boxes and fills them using a compound recordset. This allows all combo boxes on a form to be loaded with one sub call. Match the recordsets in the same order as the combo boxes in the array. Specify the display item as the first field, and ItemData as the second field in the select statements:

```
Sub FillComboBoxArray(cbArray As Variant, rsCbo _
    As ADODB.Recordset)
    Dim cb As ComboBox
    For Each cb In cbArray
        cb.Clear
        If rsCbo.Fields.Count = 1 Then
            ' If only 1 column then no index
            Do Until rsCbo.EOF
                cb.AddItem rsCbo(0)
                rsCbo.MoveNext
            Loop
        Else
            Do Until rsCbo.EOF
                ' If 2 fields then 2nd is index
                cb.AddItem rsCbo(0)
                ' Numbers only
                If IsNumeric(rsCbo(1)) Then _
                    cb.ItemData(cb.NewIndex) = rsCbo(1)
                rsCbo.MoveNext
            Loop
        End If
        Set rsCbo = rsCbo.NextRecordset
    Next
    Set rsCbo = Nothing
End Sub
```

Here's an example of how to create a compound resultset and call the FillComboArray subroutine:

```
Private Sub Form_Load()
    Dim sql As String
    Dim rs As New ADODB.Recordset
    Dim cn As ADODB.Connection
    Set cn = New ADODB.Connection
    With cn
        .ConnectionString = "driver={SQL Server};" & _
            "server=YOURSERVER;uid=sa;" & "pwd=;database=pubs"
        .Open
    End With
    sql = "SELECT au_lname FROM Authors; " & _
        "SELECT lname, job_id FROM Employee; " & _
        "SELECT pub_name FROM Publishers"
    Set rs = New ADODB.Recordset
    rs.Open sql, cn
    FillComboBoxArray cboData, rs
End Sub
```

—Kirk Ward, Hendersonville, Tennessee

VB4/32, VB5, VB6

Level: Intermediate

Determine the Correct Screen Dimensions

The latest video drivers can change the display resolution without rebooting. Unfortunately, the Screen object doesn't always properly return the new display size; it only remembers the display size when the app first used it. This behavior appears to be driver-dependent, although it might be produced by the operating system (it occurs on my Windows NT machine but not on my Windows 98 system). If you need to determine screen dimensions at any time other than the Form_Load event, use the Windows API rather than the Screen object:

```
Private Type RECT
    Left As Long
    Top As Long
    Right As Long
    Bottom As Long
End Type
Private Declare Function GetDesktopWindow Lib _
    "user32" Alias "GetDesktopWindow" () As Long
Private Declare Function GetWindowRect Lib _
```

```
    "user32" Alias "GetWindowRect" (ByVal hwnd _
    As Long, lpRect As RECT) As Long
Public Function ScreenWidth() As Single
    Dim R As RECT
    GetWindowRect GetDesktopWindow(), R
    ScreenWidth = R.Right * Screen.TwipsPerPixelX
End Function
Public Function ScreenHeight() As Single
    Dim R As RECT
    GetWindowRect GetDesktopWindow(), R
    ScreenHeight = R.Bottom * Screen.TwipsPerPixelY
End Function
```

—Matt Hart, Tulsa, Oklahoma

VB3 and up

Level: Beginning

Produce Shrinking Text

Use this code to get the shrinking text effect—similar to the opening of Star Wars:

```
' Requires a Label and Timer on the form
Private Sub Form_Load()

    With Me
        .BackColor = vbBlack
        .WindowState = vbMaximized
    End With
    With Label1
        .Alignment = vbCenter
        .AutoSize = True
        .BackColor = vbBlack
        .Caption = "Shrinking Text"
        .Font.Name = "Arial"
        .Font.Size = 150
        .ForeColor = vbGreen
        .Visible = True
    End With
    With Timer1
        .Interval = 1
        .Enabled = True
    End With

End Sub
Private Sub Timer1_Timer()
    With Label1
        If .FontSize > 2 Then
            .FontSize = .FontSize - 2
            .Left = (Me.Width - .Width) / 2
            .Top = (Me.Height - .Height) / 2
        Else
            .Visible = False
            Timer1.Enabled = False
        End If
    End With
End Sub
```

—Michael Unger, Brandon, Florida

VB5, VB6, VBA (Access)

Level: Intermediate

Reattach and Refresh SQL Links

You often need to reattach or refresh links in Microsoft Access. This code refreshes all currently linked tables to sync the attached tables with the server, then remove “dbo_” from all attached SQL Server tables. The two table Def loops allow you to re-sync tables after “dbo_” is removed:

```
Dim tbDef As TableDef
Dim db As Database
Dim strDBLocation As String
On Error Resume Next
CommonDialog1.ShowOpen
strDBLocation = CommonDialog1.FileName
If strDBLocation = "" Then
```

```
    End
End If
Set db = OpenDatabase(strDBLocation)
For Each tbDef In db.TableDefs
    ' Refresh table links
        db.TableDefs(tbDef.Name).RefreshLink
Next tbDef
For Each tbDef In db.TableDefs
' Remove all dbo_'s from tables
If Left(tbDef.Name, 4) = "dbo_" Then
    tbDef.Name = Mid(tbDef.Name, 5, Len(tbDef.Name) - 4)
End If
Next tbDef
```

—Michael Finley, Clarendon Hills, Illinois

VB5, VB6
Level: Advanced

Undocumented Boolean Field Constant

Consider the Data Definition Language statements: "ALTER TABLE [My Table] ADD COLUMN [My New Field] Single" and "ALTER TABLE [My Table] ADD COLUMN [My New Field] Double". According to Microsoft documentation, Double and Single are correct constants for field types dbDouble and dbSingle. Boolean is also indicated as the Type property for dbBoolean, yet "ALTER TABLE [My Table] ADD COLUMN [My New Field] Boolean" doesn't work. Why?

Further reading about the Boolean datatype in the DAO documentation declares that Boolean is "A True/False or yes/no value. Boolean values are usually stored in Bit fields in a Microsoft Jet database; however, some databases don't support this datatype directly." "ALTER TABLE [My Table] ADD COLUMN [My New Field] Bit" works. Ironically, nothing is listed in the table of Type properties for dbBit or Bit itself.

—Al Meadows, Oklahoma City, Oklahoma

VB5, VB6, VBS
Level: Beginning

Retrieve Recordset Fields Faster

Suppose you have a table with this field:

```
Customer_Code
```

You can retrieve the field in many ways:

```
rs.(0)
rs("Customer_Code")
rs.fields(0)
rs.fields("Customer_Code")
rs.fields.item(0)
rs.fields.item("Customer_Code")
```

In VBScript, the versions that use field indexes instead of names are faster and the extended syntax—rs.fields.item(0)—is fastest. The reason: Although VBScript, like VB, supports default properties, VBScript doesn't have to search manually for the default property of the object being referenced.

—Mostafa Fiad, Tanta, Egypt

VB5, VB6
Level: Intermediate

Use Tag Property in SQL Statements

When building SQL statements, use the Tag property to hold the Field Name and Data Format. I use a naming convention of str, int, dat, and so on to determine the datatype; the rest of the tag holds the field name in the database—such as strCompanyName, datStartDate, or intQuantity. This routine iterates through the controls on the form and determines whether the control has a tag (you should tag only the controls that hold data). The routine then checks the type of control—Textbox or MaskedEditBox—to determine whether it should use the Text or FormattedText property. If the control has a value, it builds a string consistent with the datatype. Otherwise, it adds a Null value. After running through the controls, it combines the strings. You can then use the resulting SQL statement to execute an Insert operation into the database:

```
Dim strSQL as String
Dim strColumns as String
Dim strValues as String
strSQL = "INSERT INTO [TableName] "
```

```
' start SQL statement
strColumns = "("
' hold column names
strValues = "VALUES("
' holds corresponding column values

For Each ctrl In frmSite.Controls
' iterate thru controls
    If Len(ctrl.Tag) > 0  Then
    ' if no tag, do not use
        strColumns = strColumns & Mid(ctrl.Tag, _
            4) & ", "   ' add column name
        Select Case TypeName(ctrl)
        ' find control type
            Case "TextBox", "ComboBox"
                If Len(ctrl.Text) > 0  Then
                    Select Case Left(ctrl.Tag)
                    ' find datatype, whether to
                    ' include single quotes or not
                        Case "str"
                          strValues = strValues & _
                            """ & ctrl.Text & "', "
                        Case "int"
                          strValues = strValues & _
                            ctrl.Text & ", "
                    End Select
                Else
                    strValues = strValues & "Null, "
                End If
            Case "MaskEdBox"
                If Len(ctrl.Text) > 0  Then
                    Select Case Left$(ctrl.Tag, 3)
                    ' find data type, whether to
                    ' include single quotes or not
                        Case "dat", "phn", "ipa"
                          strValues = strValues & _
                            """ & ctrl.FormattedText _
                            & "', "
                        Case "int"
                          strValues = strValues & _
                            ctrl.FormattedText _
                            & ", "
                    End Select
                Else
                    strValues = strValues & "Null, "
                End If
        End Select
    End If
Next

'remove last comma and space
strColumns = Left$(strColumns, Len(strColumns) - 2)
strValues = Left$(strValues, Len(strValues) - 2)
'add last parentheses
strColumns = strColumns & ") "
strValues = strValues & ")"
'combine strings
strSQL = strSQL & strColumns & strValues
```

—Blake Thomas, Highlands Ranch, Colorado

VB4, VB5, VB6
Level: Advanced

Export Records to CSV File for Excel

Most of my end users use laptops, which can have a wide variety of spreadsheet software installed. I often use this function when working with database tables or queries to produce a spreadsheet when I don't know what program will be used to open it. The function takes any database table or SQL Select statement and turns it into a comma-delimited text file a user can open using Notepad, Excel, or any spreadsheet program, allowing you to send data easily to another user or program. In this sample, Db is a global object variable equal to the database and has already been set by the calling program. sSource is the table or SQL statement that needs to be written to a spreadsheet:

```
Public Function TableToSpreadsheet(sSource _
```

```
    As String, sFile As String) As Boolean
    On Error GoTo TableToSpreadsheet_Err
    ' SYNTAX:
    ' If TableToSpreadsheet("SELECT * FROM
    ' Authors", "C:\Temp\Authors.csv") = True
    ' Then....
    Dim rsTemp As Recordset
    Dim sHeader As String
    Dim sRow As String
    Dim i As Integer
    Set rsTemp = Db.OpenRecordset(sSource)
    With rsTemp
        '  Make sure there are records to write
        If .RecordCount = 0 Then
            TableToSpreadsheet = False
            .Close
            Set rsTemp = Nothing
            Exit Function
        End If

        '  Create new target file
        Open sFile For Output As #1
        '  Write the header row
        For i = 0 To .Fields.Count - 1
            If i = 0 Then
                sHeader = .Fields(i).Name
            Else
                sHeader = sHeader & "," & .Fields(i).Name
            End If
        Next i
        Print #1, sHeader

        ' Loop through the table and write data rows
        .MoveFirst
        Do Until .EOF
            For i = 0 To .Fields.Count - 1
                If i = 0 Then
                    sRow = .Fields(i).Value & ""
                Else
                    sRow = .Fields(i).Value & ""
                End If
            Next i
            Print #1, sRow
            .MoveNext
        Loop
        .Close
    End With
    Close #1 ' Target file is complete
    Set rsTemp = Nothing
    ' Release recordset, but NOT database objects
    TableToSpreadsheet = True
TableToSpreadsheet_Exit:
    Exit Function
TableToSpreadsheet_Err:
    LogIt "TableToSpreadsheet : " & Err.Description
    ' LogIt is a function that creates an error log
    Resume Next
    ' Most errors result in a blank cell and can be ignored.
End Function
```

—Robert Feldsien, Hillsboro, Missouri

VB6
Level: Beginning

Use the Data Environment to Build Connection Strings

If you usually use DSN-less connections with ADO, you know it can sometimes be a pain to figure out the correct connect string. In that case, you can let the VB Data Environment do the work. Start a dummy project, bring up the Data View windows, and connect to the database you're interested in. Add a Data Environment to the project and drag a table to it. Press F4 on the connection to bring up the properties for the connection. The ConnectionSource property then has the connection string you need.

—Gary Merrifield, Madison, Wisconsin

VB6
Level: Advanced

Handle Advanced Arrays With RDS

Often you need a data structure similar to a two-dimensioned array or collection, but you need to manipulate it. For example, you need to sort on certain columns, filter certain rows, or find certain values. These functionalities are already available in the ADO Recordset object. The Microsoft Remote Data Services provides a way to store nondatabase data in a recordset using the DataFactory object. This class can help you create in-memory recordsets. Set the reference to Microsoft Remote Data Services Server 2.1 Library:

```
' code forRInMemoryRS
Option Explicit
Private df As New RDSServer.DataFactory
Private vColInfo()
Private nTotalCols As Long
Public Function Create() As ADODB.Recordset
    If nTotalCols > 0 Then
        Set Create = df.CreateRecordSet(vColInfo)
    End If
End Function
Public Sub Clear()
    ReDim vColInfo(0)
    nTotalCols = 0
End Sub
Public Sub AddColumn(szName As String, _
    nColType As ADODB.DataTypeEnum, Optional _
    nColSize As Long = -1, Optional bNullable _
    As Boolean = True)
    Dim vCol(3)
    ReDim Preserve vColInfo(nTotalCols)
    vCol(0) = szName
    vCol(1) = CInt(nColType)
    vCol(2) = CInt(nColSize)
    vCol(3) = bNullable
    vColInfo(nTotalCols) = vCol
    nTotalCols = nTotalCols + 1
End Sub
Private Sub Class_Initialize()
    nTotalCols = 0
End Sub
```

Use code like this:

```
Dim rsMem As ADODB.Recordset
Dim RMemRS As new RInMemoryRS
' Create Two Column Table
RMemRS.AddColumn "Name", adChar, 10, False
RMemRS.AddColumn "Age", adchar, 10
Set rsMem=RMemRS.Create
' Now, you can add the data to the "Memory
' Recordset" for example
rsMem.AddNew
rsMem!Name = "John"
rsMem!Age = 15
rsMem.Update
rsMem.AddNew
rsMem!Name = "Kevin"
rsMem!Age = 25
rsMem.Update
```

You can manipulate rsMem like this:

```
rsMem.Filter = "Age > 15"
rsMem.Sort = "Age ASC"
rsMem.Save szFileName
rsMem.Find "Name = 'John'"
```

—Rajesh Pohuja, Singapore

VB6
Level: Beginning

Clean Quotes From SQL Parameters With Replace

If you've ever used SQL commands against the ADO Connection object, you might have had a problem allowing the user to enter text that contains an apostrophe:

```
ADOCon.Execute "Insert Into Emp(Name) Select '" _
    & txtName.Text & "'"
```

This works fine if the name is Smith, but fails if the name is O'Connor. You can easily solve this problem with VB6's Replace function. Use the Replace function to parse the string and replace the single apostrophe with two apostrophes (not double quotes):

```
ADOCon.Execute _
    "Insert Into Emp(Name) Select '" _
    & Replace(txtName.Text, "'", "''") & "'"
```

—Scott Summers, Denver, Colorado

VB5, VB6
Level: Beginning

Dynamically Populate MSFlexGrid Control

If you use an MSFlexGrid control to display data returned in an ADO recordset, you can use this code to dynamically populate the grid—including the header row—with the information in the recordset. You need an open ADO recordset named rst and a form containing an MSFlexGrid control named msfGrid:

```
Dim cln As Field
With msfGrid
    .Rows = 2
    .Cols = rst.Fields.Count
    'get the number of grid cols
    .FixedRows = 1
    .FixedCols = 0
    .Row = 0
    .Col = 0
    For Each cln In rst.Fields
        .Text = cln.Name
        'populate header row with names of fields
        If .Col < .Cols - 1 Then .Col = .Col + 1
    Next
    Do While Not rst.EOF
    'loop thru recordset to populate grid
        .Row = rst.AbsolutePosition
        'move to the next row
        .Col = 0
        'reset ourselves back to column(0)
        For Each cln In rst.Fields
            If Not IsNull(cln.Value) Then
                .Text = Trim(CStr(cln.Value))
            Else
                .Text = ""
            End If
            If .Col < .Cols - 1 Then .Col = .Col + 1
        Next
        rst.MoveNext
        .Rows = .Rows + 1
        'add a new row to the grid
    Loop
    .Rows = .Rows - 1
    'remove the last row because it's blank
    .Row = 0
End With
```

—David George, Glen Burnie, Maryland

VB5, VB6
Level: Intermediate

Optimize Parametrized Queries With ADO Objects

When you write Insert statements, it can be difficult to accommodate the possible values end users might enter into a textbox. The most common task is replacing single quotes with double quotes. However, parameterized queries provide two benefits: You do not have to parse data entered by users—except for

business rules; and SQL Server 7.0 immediately caches the SQL statement:

```
Dim cmd As ADODB.Command
Dim prm As ADODB.Parameter
Set cmd = New ADODB.Command
Set prm = New ADODB.Parameter
With cmd
    .ActiveConnection = CONNECT_STRING
    .CommandText = "INSERT INTO employees " & _
        "(name) VALUES(?)"
    .CommandType = adCmdText
    Set prm = .CreateParameter(, adChar, _
        adParamInput, 50, Me.txtName.Text)
    .Parameters.Append prm
    .Execute
End With
Set cmd = Nothing
Set prm = Nothing
```

—Christopher P. Madrid, Austin, Texas

VB4, VB5, VB6
Level: Intermediate

Avoid Installation Problems With the Msldvusr.dll in Win95

If you're developing multiuser Jet-based applications, you probably know about Microsoft's msldbusr.dll. It allows you to read the Jet lock files and get the correct number of connected users and computers they connect from. However, if you use this unsupported DLL, keep this gotcha in mind: If you include the DLL in a VB application installation and the user is running an early Windows 95 version, the DLL disappears when the user shuts down his or her machine.

To solve this problem, include an uncompressed copy of the DLL with your setup. If, after a reboot, the user manually copies the file back to the Windows system folder, it will stay there forever and give you back your functionality. This oddness does not occur when installing to Windows 98 or NT.

—Robert Smith, Kirkland, Washington

VB5, VB6
Level: Advanced

Avoid Cursor Problems in Oracle With Precompiled Queries

ADO and RDO do not support Oracle cursor types; neither Microsoft nor Oracle drivers provide appropriate conversion. As a result, you cannot use Oracle stored procedures from VB or Active Server Pages (ASP) to retrieve a multiple-row recordset. Instead, dynamic embedded SQL statements have to be passed. At this point, the performance degrades and the code becomes difficult to maintain. A better alternative—besides using a third-party driver—is to use precompiled (prepared) queries with parameters. These queries can be declared and precompiled when your application or component is first initialized (in the Sub Main, Initialize, or Load events, or include file). Later, you can assign the parameters and call queries—including stored procedures consisting of single SQL statements—by their names. This approach can be faster (both in development and performance) for implementing business logic using VB built-in functionality instead of customized PL/SQL functions. It also can be applied with any RDBMS if you want to separate the database logic. (Queries are created by your component, are invisible in DBMS environment, and gone with your app). It's also easier to adapt and port components against different RDBMS-modifying SQL statements to a particular dialect—or using standard SQL—than convert vendor specific "glue" languages such as PL/SQL or Transact-SQL.

—Victor Karlovich, Bayonne, New Jersey

VB4, VB5, VB6
Level: Intermediate

Sort DBGrid Contents With Recordset Refresh

It's often useful to sort a DBGrid field in either descending or ascending order. You do this by using the HeadClick event and the DataField property of columns. You must change the query string (Qry) with one of your own (be sure it contains the code in bold):

```
Private Sub DBGrid1_HeadClick(ByVal ColIndex As Integer)
    Dim Qry as string
    Qry = "SELECT * FROM MyTable WHERE Key='" _
        & txtKey & "' "
    Qry = Qry & "ORDER BY " & _
        DBGrid1.Columns(ColIndex).DataField _
        & " " & DBGrid1.Tag
```

```
    data1.RecordSource = Qry
    data1.Refresh
    DBGrid1.ReBind
    'toggle ASC and DESC keywords
    If DBGrid1.Tag = "ASC" Then DBGrid1.Tag _
        = "DESC" Else DBGrid1.Tag = "ASC"
End Sub
```

—Juan Jose Ochoa, Nogales, Arizona

VB6
Level: Beginning

Edit Field in DataGrid on F2

Sometimes you want to give your DataGrid the ability to edit fields, while the original data in the field is highlighted. Normally, you can click on the field to start editing it. However, in case your users prefer to use the keyboard instead of the mouse, put this code in the grdDataGrid_KeyDown event:

```
Private Sub grdDataGrid_KeyDown(KeyCode _
    As Integer, Shift As Integer)
    Select Case KeyCode
        Case vbKeyF2
            grdDataGrid.SelStart = 1
            SendKeys "{End}"
    End Select
End Sub
```

—Decha Srivorapun, Bangkok, Thailand

ASP
Level: Intermediate

Display Client-Side Message Box From Server-Side Script

If you have a form validated using server-side Active Server Pages (ASP) code and you need to display an error message, you would normally display it at the top or bottom of your form and send the form back so the user can correct his or her mistake. For example: "The password you have entered is invalid. Please try again."

However, it would be nice if the error message popped up in a message box instead. The issue is how to make a client-side message box pop up when your code is executing on the server. The answer is simple: If your message is in the variable strErrMsg, use this code at the bottom of your ASP page displaying the form:

```
<%
    if strErrMsg <> "" Then
        ' There is an error, pop it up
%>
    <SCRIPT LANGUAGE="JavaScript">
    <!—
        alert('<%= strErrMsg %>');
    // —>
    </SCRIPT>
<%
    End if
%>
```

After your page loads, it displays the error message in a message box.

—Rama Ramachandran, Stamford, Connecticut

VB6
Level: Beginning

Fix Justification Glitch in MSFlexGrid

The MSFlexGrid tries to automatically determine how to justify text. If the first character is numeric, then that cell will be right-justified. If it is an alphanumeric character, then that cell will be left-justified. The problem arises when you try to display a freeform note in one of the cells. If the note starts with a number, such as "30 days until renewal," MSFlexGrid right-justifies that cell. The solution is to prefix all cells with a space:

```
Sub FillGrid(rs As RecordSet)
    Dim sItem As String
    Dim i as Long
    '//Loop through the recordset
```

```
    rs.MoveFirst
    Do Until rs.EOF
        '//Loop through the fields
        sItem$=""
        For i = 0 To rs.Fields.Count -1
            'Build the row to be inserted, vbTab
            'first so that we skip the fixedcol and
            'space so that everything is left justified
        sItem = sItem & vbTab & " " & rs.Fields(i)
        Next i
        '//Add The row to the grid
        grd.AddItem sItem
        '//Move to the next record
        rs.MoveNext
    Loop
End Sub
```

—Pat Labelle, Ottawa, Ontario, Canada

VB6, ASP

Level: Advanced

Pass Arrays ByVal From ASP Scripts to VB COM Objects

In Microsoft Knowledge Base article Q217114, "How to: Implement Array Arguments in Visual Basic COM Objects for Active Server Pages," Microsoft says you can't pass an array to a COM method by value. However, you want to do this for Microsoft Transaction Server (MTS), so here is a workaround that does it ByVal. Add a file called test.asp with this Active Server Page (ASP) code to a virtual Internet Information Server (IIS) directory:

```
<%
dim PassArrayByValWorks
dim ary(1)
dim iReturn
ary(0) = "firstone"
ary(1) = "2ndone"
' pass the array to a non array declared variable then
' pass the non array variable instead
PassArrayByValWorks = ary
dim obj
set obj = server.createobject ("prjFormCheck.clsFormCheck")
iReturn = obj.formcheck(PassArrayByValWorks, 0)
%>
<%=iReturn%>
' build ActiveX dll named "prjFormCheck", name
' class "clsFormCheck" add the function below
' and start it in the VB IDE
Public Function FormCheck(ByVal _
    arrFldNameValuePairs As Variant, ByVal _
    ErrLogType As Variant) As Variant
    If IsArray(arrFldNameValuePairs) Then
        FormCheck = "You can do it!"
        Debug.Print arrFldNameValuePairs(0)
        Debug.Print arrFldNameValuePairs(1)
    Else
        FormCheck = "Didn't work"
    End If
End function
```

Right-click on the test.asp file in the virtual directory of IIS and click on Browse. The browser should show "You can do it!"

—Mark Kanter, received by e-mail

VB5, VB6

Level: Intermediate

Add a Scripting Engine to Your Application

It's easy to add scripting functionality to your VB project, especially if you have been developing through classes all along. The more classes you program, the more objects you can expose to your script language. You can use both VBScript and JScript as the basis for your scripting engine.

First, download the Microsoft Script Control from msdn.

microsoft.com/scripting/scriptcontrol. Install the control according to the instructions provided. You might need to register the control manually (run regsvr32 on it). The footprint on this control is low; the whole download including help is only 243K. Next, create a script file with a text editor such as Notepad:

```
Sub Main()
    MsgBox "Hello, world"
End Sub
```

Save it as c:\temp.txt and add this code to your application:

```
Private Sub Command1_Click()
    Dim iFileNum As Long
    Dim sFileBuffer As String
    Dim sTemp As String
    iFileNum = FreeFile()
    Open "c:\temp.txt" For Input As #iFileNum
        Do While Not EOF(iFileNum)
            Line Input #iFileNum, sTemp
            sFileBuffer = sFileBuffer & sTemp & _
                vbCrLf
        Loop
    Close #iFileNum
    ScriptControl1.Reset
    ScriptControl1.AddCode (sFileBuffer)
    ScriptControl1.Run "Main"
End sub
```

You have now successfully implemented a scripting engine. You can expose objects in your application like this:

```
Private Sub Command1_Click()
    Dim objMyClass As New MyClassNameHere
    With dlgCommon
        .ShowOpen
        sFileName = .FileName
    End With
    iFileNum = FreeFile()
    Open sFileName For Input As #iFileNum
        While Not EOF(iFileNum)
            Line Input #iFileNum, sTemp
            sFileBuffer = sFileBuffer & sTemp & vbCrLf
        Wend
    Close #iFileNum
    ScriptControl1.Reset
    ScriptControl1.AddObject "Database", objMyClass
    ScriptControl1.AddCode (sFileBuffer)
    ScriptControl1.Run "Main"
End sub
```

You can even try code such as this to give ad hoc capabilities to an application:

```
ScriptControl1.ExecuteStatement "x = 100"
MsgBox ScriptControl1.Eval("x = 100")  ' True
MsgBox ScriptControl1.Eval("x = 100/2")  ' False
```

—Dan Newsome, Denver, Colorado

VB5, VB6
Level: Intermediate

Connect to Microsoft Excel Using OLE DB

Microsoft documentation says you can connect to Excel 97 or Excel 2000 using the Microsoft.Jet.OLEDB 4.0 provider. If you use the Microsoft ADO Data Control, however, you will have problems. From the property page for the ADO Data Control, choose the Use Connection String radio button and click on the Build button. Then, select the database name, choosing an Excel file as your database. Now, if you click on Test Connection, you get an error message saying the connection failed because the file is in an unrecognized format.

But wait, there's hope! Acknowledge the error message and return to the General tab of the property pages. In the Connection String textbox, add this code to the end of the connection string:

```
Extended Properties = Excel 8.0;
```

Your full connection string now looks like this:

```
Provider=Microsoft.Jet.OLEDB.4.0; Data Source = FileName; Extended Properties=Excel 8.0;
```

Now if you click on the Build button, then click on the Test Connection button, the connection is successful.

—Michael J. McElwee, Highland Park, Illinois

VB4, VB5, VB6
Level: Intermediate

Define Named Ranges in Excel Before Executing Queries Against Worksheets

Once you've established a connection to Microsoft Excel using OLE DB, you're not out of the woods. You still have to define a Named Range in Excel; then you can treat this named range like a database table to perform queries against. To do this from Excel, select the range of cells you want to represent the table—column headers in the first row—then choose Name | Define from the Insert menu to bring up the Define Name dialog. Choose a name for your table and click on OK. Be sure to have valid column names or they will be renamed for you in the recordset or table you bring into your application.

As a different approach, you might wish to do things through VB code. Using the Excel 8.0 object as a reference, this example takes a file specified by the FileName string and creates a named range whose name is specified by the variable TableName. This example chooses the used portion of the first sheet as the table range:

```
Public Sub MakeExcelTable(FileName As String, _
    Tablename As String)
    Dim BookXL As Excel.Workbook
    Dim RangeXL As Excel.Range
    Dim SheetXL As Excel.Worksheet
    Set BookXL = GetObject(FileName)
    With BookXL
        Set SheetXL = BookXL.ActiveSheet
        Set RangeXL = SheetXL.UsedRange
        ' Selects the entire used range of the first sheet
        .Names.Add TableName, RangeXL
        .Windows.item(1).Visible = True
        .Save
    End With
    Set BookXL = Nothing
End Sub
```

—Michael J. McElwee, Highland Park, Illinois

VB6
Level: Advanced

Pass MTS Object References Safely Between Processes

SafeRef returns a reference to the context wrapper instead of a reference to the object itself. You should never pass a direct reference of a Microsoft Transaction Server (MTS) object to a client application. If you do, the client application can make a call on the MTS object without going through the context wrapper. This defeats the interception scheme set up by the MTS runtime. You should use the SafeRef function, which returns the outside world's view of an MTS object. Let's say you're writing a method implementation for an MTS object. In the method, you want to create an object of some class, then pass your own reference to the newly created object. To do this, you might write this code:

```
Dim pSomeClass As CSomeClass
Set pSomeClass = New CSomeClass
pSomeClass.SomeMethod Me
' Incorrect code
```

However, this code is incorrect under MTS. The child object—pSomeClass—can invoke method calls on your object without going through the context wrapper. You should never bypass the interception scheme set up by the MTS runtime. Instead, you should pass a reference to your object:

```
pSomeClass.SomeMethod SafeRef(Me)
' Correct code
```

By calling SafeRef, you allow the child object to establish a connection that passes through the context wrapper. This technique keeps things in line with what the MTS runtime expects. The Me keyword is the only valid parameter you can pass when calling SafeRef with Visual Basic.

—Deepak Pant, Lowell, Massachusetts

VB4, VB5, VB6
Level: Intermediate

Convert Numbers to Excel Column Names

Microsoft Excel labels its columns A though Z, then AA, AB, and so on. To access a given cell of an Excel sheet using the Excel object library reference, use a statement like this:

```
Dim xlapp as New Excel.Application
xlapp.Range("A1").Value = 6
```

This statement sets the first cell of the sheet to 6. If you convert the column names from alphabet-like (A through IV) to numbers, you can then go through a loop to access every cell in a given sheet or range. This function performs the required conversion. It works only through Column 701, but Excel doesn't allow nearly that many columns, so it's a nonissue:

```
Private Function GetXLCol(Col As Integer) As String
    ' Col is the present column, not the number of cols
    Const A = 65    'ASCII value for capital A
    Dim iMults As Integer
    Dim sCol As String
    Dim iRemain As Integer
    ' THIS ALGORITHM ONLY WORKS UP TO ZZ. It fails on AAA
    If Col > 701 Then
        GetXLCol = ""
        Exit Function
    End If
    If Col <= 25 Then
        sCol = Chr(A + Col)
    Else
        iRemain = Int((Col / 26)) - 1
        sCol = Chr(A + iRemain) & GetXLCol(Col _
            Mod 26)
    End If
    GetXLCol = sCol
End Function
```

—Michael J. McElwee, Highland Park, Illinois

VB6
Level: Advanced

Use Asynchronicity for Speed

If you need to run a complicated query that returns a large recordset, ADO 2.1 gives you the best of both worlds. Sometimes you just need to put a recordset into an AddItem type grid, or prepare it for a report. So, if you need to process the recordset as soon as the first record is fetched, you should start processing in the Execute_Complete event of the connection object. If you can also use a disconnected recordset, you can set the ActiveCon-nection property equal to nothing. This code might be the fastest way to process a large recordset with ADO:

```
Private WithEvents m_adoConEvent As ADODB.Connection
' the RS that enables the event Fetch_Complete
' to be fired off Attribute
Private WithEvents m_adoRstEvent As ADODB.Recordset
Private Sub GetRecordSet()
    Dim sSQL As String
    'A large or complicated SQL statemtent
    sSQL = "select a large complicate query"
    Set m_adoConEvent = New ADODB.Connection
    Set m_adoRstEvent = New ADODB.Recordset
    m_adoConEvent.Open "Connection String"
    m_adoRstEvent.CursorLocation = adUseClient
    Me.Caption = "Started"
    'Do something to tell the user where the process is
    'at. Have the command execute and fetch at the same
    'time without interrupting workflow.
    m_adoRstEvent.Open sSQL, m_adoConEvent, _
        adOpenStatc, adLockReadOnly, adCmdText _
        Or adAsyncFetch Or adAsyncExecute
```

```
End Sub
Private Sub m_adoConEvent_ExecuteComplete(ByVal _
    RecordsAffected As Long, ByVal pError _
    As ADODB.Error, adStatus As ADODB.EventStatusEnum, _
    ByVal pCommand As ADODB.Command, ByVal pRecordset As _
    ADODB.Recordset, ByVal pConnection As ADODB.Connection)
    Do Until pRecordset.EOF
        'start processing the recordset
    Loop
End Sub
Private Sub m_adoRstEvent_FetchComplete(ByVal pError _
    As ADODB.Error, adStatus As ADODB.EventStatusEnum, _
    ByVal pRecordset As ADODB.Recordset)
    Set pRecordset.ActiveConnection = Nothing
    'this will speed processing time
End Sub
Private Sub m_adoRstEvent_FetchProgress(ByVal Progress _
    As Long, ByVal MaxProgress As Long, _
    adStatus As ADODB.EventStatusEnum, ByVal _
    pRecordset As ADODB.Recordset)
    'let the user know work is happening
End Sub
```

By using both adAsyncFetch and adAsyncExecute, you can start processing even while you're returning data.

—Darren McBratney, Leawood, Kansas

VB5, VB6
Level: Beginning

Encase Query Names in Brackets to Avoid ADO/Jet Errors

While using OLE DB Provider for Jet to manage Access databases through ADO from VB, follow this tip to help you access a query defined in your database. If the query name contains blank characters (ASCII code 32), supply this code, enclosing the name in brackets:

```
'To access the query by the
'Recordset object's Open method:
rs.Open "[Name with blanks]", _ 'Etc.

'To access the query by a Command object:
cm.CommandText = "[Name with blanks]"
```

If you don't, your program will show a runtime error when you try to open the recordset. The OLE DB Provider for Jet databases interprets the supplied string as a SQL statement, which doesn't match that language syntax. Specifying different values for the Command object's CommandType property—or specifying different values on the Options argument when invoking the recordset's Open method—doesn't fix the problem. You must use the brackets.

—Leonardo Bosi, Buenos Aires, Argentina

VB4, VB5, VB6
Level: Advanced

Binary Search Routine For RDO

RDO does not have a FindFirst or Seek method, and as a programmer, you sometimes need to quickly move to a particular record. I had a user who wanted to be able to scroll through more than 3,000 records and also be able to search for a particular record. A linear search was too slow, so I decided to write a binary search routine. The routine is case-insensitive and finds the matching entry. It can be copied into the form module and used for searches on any sorted column for a given resultset. It takes three arguments: the rdoResultset being searched, the column name within the resultset to be searched, and the search string:

```
Private Sub BinarySearch(ByRef rs _
    As rdoResultset, ByVal strColName As _
    String, ByVal varSearch As Variant)
    Dim lngFirst&, lngLast&, varBookMark
    varBookMark = rs.Bookmark 'set a bookmark
    lngLast = rs.RowCount
    If lngLast = 0 Then Exit Sub
    lngFirst = 0
```

```
    rs.AbsolutePosition = (lngLast - lngFirst) _
        \ 2 'move to middle
    varSearch = Trim(UCase(varSearch))
    Do While ((lngLast - lngFirst) \ 2) > 0
        Select Case StrComp(UCase(Left(Trim _
            (rs.rdoColumns(strColName)), _
            Len(varSearch))), varSearch, vbTextCompare)
            Case 0 'found
                Exit Sub
            Case -1 'still ahead
                lngFirst = rs.AbsolutePosition
                rs.Move (lngLast - lngFirst) \ 2
            Case 1 'left behind
                lngLast = rs.AbsolutePosition
                rs.Move (-1) * ((lngLast - lngFirst) \ 2)
        End Select
    Loop
    rs.MoveLast
    If (StrComp(UCase(Left (Trim(rs.rdoColumns(strColName)), _
        Len(varSearch))), varSearch, _
        vbTextCompare)) <> 0 Then
        ' record not found. Return to bookmark and
        ' display message.
        rs.Bookmark = varBookMark
        MsgBox "Entry not found for " _
            & varSearch, vbOKOnly Or _
            vbInformation, "Binary Seacrh"
    End If
End Sub
```

Because this search always misses the last item, the last record is checked specifically. Also, this routine finds the last matching record if the record is positioned before the middle of a resultset, or the first matching record if the record is positioned after the middle of a resultset. For example, say there are three Smith's—A. Smith, B. Smith, and C. Smith—in a resultset and the user is searching by last name for Smith. Also assume there are 100 records in the resultset. If C. Smith is at absolute position of 25 (< 50 = 100/2), then this search routine finds C. Smith first. However, if A. Smith has an absolute position of 59 (>50 = 100/2), then this search routine finds A. Smith first. In case the Smiths happen to be somewhere in the middle, this search routine finds the first Smith encountered. This routine works best for searching on unique keys, such as Social Security numbers.

—Rajnish Kashyap, Miami, Florida

VB4, VB5, VB6

Level: Intermediate

Delete All Records in a Table

If you find yourself repeating the same Execute method in different parts of your code when clearing tables, use this method instead to automate the process. When you already have a global variable set to the open database, delete all the records in a table with this function, where DB is the database object:

```
Function ZapTable(sTable As String, _
    Optional sWhere As String = "") As Integer
    Dim sSQL As String
    On Error GoTo Err_ZapRecs
    ' For Access Apps only:
    ' docmd.SetWarnings False
    sSQL = "DELETE * FROM " & sTable & " "
    If sWhere <> "" Then
        sSQL = sSQL & "WHERE " & sWhere
    End If
    DB.Execute sSQL, dbFailOnError
    'docmd.SetWarnings True
    ZapTable = True
Exit_ZapRecs:
    Exit Function
Err_ZapRecs:
    ZapTable = False
''ERROR HANDLING IF DESIRED
    Resume Exit_ZapRecs
End Function
```

Use this function in the code as in these examples:

```
If Not ZapTable("locLookup") Then
    MsgBox "Cannot delete Table."
End If
```

Or:

```
If Not ZapTable("locCities", "STATE = 'NY'") Then
    MsgBox "Cannot delete Table."
End If
```

—Fabio A. Mir, Sr., Gaithersburg, Maryland