Supplement to

A FAWCETTE TECHNICAL PUBLICATION, FEBRUARY 1996 VOL. 6, NO. 2,
INCLUDES WINDOWS PROGRAMMING & VISUAL PROGRAMMING

VISUAL BASIC®

PROGRAMMER'S JOURNAL

2nd Edition

Featuring

99

of the HOTTEST

TECH TIPS

For VB Developers

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

WELCOME TO THE VBPJ *TECHNICAL TIPS SUPPLEMENT!*

***VBPJ's* 99 Tech Tips are back by popular demand. These tips and tricks were compiled by professional developers and the editors at *Visual Basic Programmer's Journal* using both Visual Basic 3.0 and Visual Basic 4.0. To save hours of needless rekeying, and to get the tips that wouldn't fit in this booklet, download them from *VBPJ's* Developer's Exchange on the World Wide Web. This service is available in our subscriber hotline section—absolutely free to *VBPJ* subscribers. To visit the Developer's Exchange point your Web browser to http://www.windx.com.**

VB3 VB4

BUILD DEBUGGING/PROFILING INTO YOUR CODE

To build debugging and profiling into your code, add this to the end of code lines:

```
'Code line   Add this line (n should be replaced
'by a unique number)
i = j + 1    '*dbg* "Debug n" & time
```

Turn debugging/profiling on by simply doing a replace on all modules of '*dbg* with :debug.print. When you run your code, the output will be sent to the debug window. To turn off debugging/profiling, replace :debug.print with '*dbg.*

Even more interesting is the fact that you can do this at the procedure level as well. For example:

```
Sub Form_Load ()  '*dbg*  "form_load:" & Time;
```

after replacement:

```
Sub Form_Load () : Debug.Print "form_load:" & Time;
```

Finally, if you like, you can change '*dbg* to a procedure name that could write all output to a file.

—Robert Meyering

VB3 VB4

EXTRACT FILE NAMES FROM FILE SPECIFICATIONS

Here is a simple way to extract file names from file specifications:

```
Function FName (filespec As String) As String
    Dim i As Integer
    Dim size As Integer
    size = Len(filespec)
    For i = size To 1 Step -1
        If Mid$(filespec, i, 1) Like "[\:]" Then
            FName = Right(filespec, size - i)
            Exit Function
        End If
Next i
End Function
```

For example:

```
    File_name = FName("A:\test.dat")
    File_name = FName("A:test.dat")
    File_name = FName("A:\test\test.dat")
```

—Robert Meyering

VB3 VB4

OPTIMIZE LABEL CREATION

Place one label on your form, tailor its properties (such as left-align and autosize), and copy and paste it to create the rest of your labels.

—Robert Meyering

VB3

CREATE "TAG TIPS"

Use the Tag property and the MouseMove event to create "tag tips," which look like tool tips. Create a label to be used as the tool tip box and set its visible property to False, and its AutoSize property to True. Then add this code to the MouseMove event of the control to which you are adding tag tip text:

```
Label1.Caption = Command1.Tag
Label1.Top = Command1.Top + Command1.Height
Label1.Left = Command1.Left + Command1.Width / 2
Label1.Visible = TRUE
```

Set the Tag property of the control to the text you want to display as the tag tip:

```
Command1.TAG = "This is the TAG TIP"
```

Add this to the MouseMove event of the form. It turns off the tag tip when you move your mouse off the control:

```
Label1.Visible = FALSE
```

—Robert Meyering

VB3

USE "&," NOT "+," WHEN CONCATENATING STRINGS

Use an ampersand (&) instead of a plus sign (+) when concatenating strings. Depending on the data type of the operands, using the plus sign may result in an addition.

—Robert Meyering

VB4

CENTER YOUR FORMS

Add a method in VB 4.0 to center your forms against the screen or a parent form. Create a new project with two forms. Add this code to Form2:

```
Public Sub ShowCentered(Optional vParent, _
    Optional vShowMode)
Dim oParent As Object
Dim iMode%, iLeft%, iTop%

    If IsMissing(vParent) Then 'default is Screen object
        Set oParent = Screen
    ElseIf TypeOf vParent Is Screen Or _
        TypeOf vParent Is Form Then
        Set oParent = vParent
        'can add MDIForm to this condition
    Else
        Exit Sub
    End If

    If IsMissing(vShowMode) Then iMode = vbModeless Else _
        iMode = Abs(vShowMode) Mod 2
    'default is Modeless

    If TypeOf oParent Is Form Then iLeft = oParent.Left: _
        iTop = oParent.Top
    'cannot use Left and Top for Screen

    Move iLeft + (oParent.Width - Width) / 2, _
        iTop + (oParent.Height - Height) / 2

    Show iMode

End Sub
```

Put this code in Form1's Form_Load event:

```
    Show

    Form2.ShowCentered Me, vbModal
```

Omit either or both of the arguments to see how the Optional parameter keyword in VB 4.0 increases the versatility of such procedures.

—Ian Carter

VB4

DOCUMENT YOUR OLE SERVERS

Document the interface for your OLE server by using the Options button in the Object Browser. To provide more full-featured documentation, use a help file. Use the Object Browser to tie the help file to the OLE server.

—Deborah Kurata

VB4

RAISE YOUR OLE SERVER ERRORS

Don't display errors from the OLE server. Instead, raise an error to the client application. Use error numbers greater than vbObjectError + 512 and less than vbObjectError + 5535. Values between vbObjectError and vbObjectError + 512 can conflict with OLE error values. Be sure to document error numbers in the help file for your OLE server.

—Deborah Kurata

VB4

WATCH WHAT YOU PASS TO YOUR OLE SERVERS

Don't use any objects in the Visual Basic object library as parameters or return values for exposed properties or methods in public classes. These objects are not intended to be used from outside of a single project. Using them in that manner may cause unexpected results. Use the Object Browser to review this list of the Visual Basic (VB) objects.

—Deborah Kurata

VB4

OLE SERVER OBJECT MANIPULATION

Don't return references to Form and Control objects from your OLE server. Rather, provide wrappers for the properties and methods of forms and controls if they must be manipulated by your OLE server. For example, instead of returning a text box and allowing the client application to manipulate the text box, provide wrapper methods in the OLE server for Move, and wrapper properties for Text.

—Deborah Kurata

VB4

BE CAREFUL WHEN TERMINATING OLE SERVERS

Don't provide a method that terminates your server. Let the server automatically terminate when no references to the server exist. If you provide an Exit menu option in an OLE server that displays a user interface, simply unload the forms the user opened and free all object instances that were created.

—Deborah Kurata

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

VB4

SET ALL OBJECT VARIABLES TO NOTHING

Do set all object variables to nothing before ending the application. When the End statement is executed in the OLE client application, the OLE server is shut down and the Terminate events of any objects that have not yet been terminated are not executed.

—Deborah Kurata

VB4

USE THE MOST SPECIFIC OBJECT TYPE AVAILABLE

Instead of declaring an object As Object, use the specific object type, such as CTask. This improves performance by minimizing the OLE lookup requirements.

—Deborah Kurata

VB4

USE OBJECT SUBSTITUTION

Use object substitution to substitute a simple name with an extended object reference. Each "." in the syntax represents an OLE lookup. Better performance is achieved the fewer times lookups need to be performed. For example:

```
For i = 1 to 10
    <some code>
    txt(i) = myApp.TimeSheet.Employee.Name
Next I

Performs better if rewritten using substitution:

Dim emp as New Employee

' Create the substitution object variable
Set emp = myApp.TimeSheet.Employee

For i = 1 to 10
    <some code>
    txt(i) = emp.Name
Next I
```

—Deborah Kurata

VB4

MINIMIZING THE NUMBER OF REPEATED OLE LOOKUPS

Use With...End With to minimize the number of repeated OLE lookups. This has the added advantage of not requiring the temporary substitution object.

—Deborah Kurata

VB4

USE IN-PROCESS SERVERS WHENEVER POSSIBLE

Use in-process servers (OLE DLLs) whenever possible to improve performance. Calls to objects within the application's process space, either in the application or in an in-process server, are significantly faster than calls to objects outside the application's process space.

—Deborah Kurata

VB4

SPEED OLE SERVER CALLS

Pass as much data to and from the OLE server as possible in each call. Data transfer is fast, but calling is slow, especially with out-of-process servers. For example, you could pass a set of parameters in an array instead of calling the OLE server multiple times.

—Deborah Kurata

VB4

INITIALIZING AN IN-PROCESS OLE SERVER DLL

Start the in-process OLE server using a Sub Main procedure. Include all server initialization code in the Sub Main procedure. Don't show forms from the Sub Main procedure and don't use the Command Function to retrieve command line contents. There is no command line when initializing an in-process server.

—Deborah Kurata

VB4

TESTING AN IN-PROCESS OLE SERVER DLL

Test an in-process OLE server DLL by building it first as an out-of-process server. This makes it possible to debug and test. Also, the stability of the entire application will be affected by the in-process server. It is a good idea to ensure the OLE server is stable as an out-of-process server before converting it to an OLE DLL.

—Deborah Kurata

VB4

TEST THE IN-PROCESS OLE SERVER AS AN OUT-OF-PROCESS SERVER

Use the OLE Restrictions option to test the in-process OLE server as an out-of-process server. The OLE DLL Restrictions option is in the Advanced tab of the Options dialog box. This option enforces the DLL restrictions although you are using an out-of-process server. This provides a better test of how the server will function as an in-process server.

—Deborah Kurata

VB4

UNLOADING THE SERVER FROM MEMORY

Shut down the instance of Visual Basic that uses the compiled in-process OLE server to unload the server from memory. Changes to the in-process OLE server will not be seen until the current version of the server is unloaded from memory and the newest version is loaded.

—Deborah Kurata

VB4

SETTING THE MOUSEPOINTER IN AN IN-PROCESS OLE SERVER

Don't set the MousePointer in an OLE DLL unless absolutely necessary. The in-process OLE server can not retrieve the current mouse pointer, so there is no way to set it back to what the application expects. If you must set the MousePointer in an in-process OLE server, always return it to the default value before returning to the client application. If the client application has a different MousePointer set, it is then responsible for returning it to a desired value after calling the server property or method.

—Deborah Kurata

VB4

REDEFINE YOUR TAB ORDER

When you need to reset your tab indexes quickly (without buying a third party VB extension product), set the tabs in the reverse tab order assigning each tab index to zero (0). That is, go to the very last control on the page that will receive a tab stop. Bring up the properties window, locate TabIndex, hit the zero key, click the second from the last control to receive a tab stop, type zero, click the third from the last control, type zero, and so on. This little trick will save you lots of time and aggravation, and it works perfectly every time.

—John Chmela

VB4

CONVERTING STRINGS TO TITLE CASE

I'll present this tip the VB3 way and the VB4 way. I beat myself for a couple of hours over this routine to convert a string to Title Case. I used this with VB3:

```
Function TCase(StrInp As String)
Dim strout$
Dim x%
StrInp$ = LCase$(StrInp$)
strout$ = UCase$(Left(StrInp$, 1))
For x% = 2 To Len(StrInp$)
    Select Case Asc(Mid$(StrInp$, x, 1))
        Case Is < 97, Is > 122
            strout$ = strout$ & Mid$(StrInp$, x, 1)
        Case 97 To 122
            Select Case Asc(Mid$(StrInp$, x - 1, 1))
                Case 97 To 122
                    strout$ = strout$ & _
                        Mid$(StrInp$, x, 1)
                Case Else: strout$ = _
                    strout$ & UCase$(Mid$(StrInp$, x, 1))
            End Select
    End Select
Next
TCase = strout$
End Function
```

In VB4, one line replaces the whole function:

```
StrConv(string, vbProperCase)

'Where vbProperCase is a built in constant in VB4
```

—Jason Lee Stenklyft

VB3

DEAL WITH THE DATACONTROL AT RUN TIME AND DESIGN TIME

One of the greatest features of data-aware controls is that you simply connect them to a DataControl and select the fields you need. You have just created a database program. It can't get any easier than that. The only problem is that if you share this program, nine times out of 10 it won't work. Why? Because the database name and directory structure are hard coded into the DatabaseName field in the DataControl.

Hard-coding the database name into the DataControl is a major programming no-no. Instead, you fill this field at run time. Then, before you can ship (or even send the program out to beta testers), you must remove the database names from your controls. Then you add them back during design, and so on. What a headache.

I don't have time for that. I created a nice Sub that does the work for me. Before I give it to you, I need to review a few idiosyncrasies of loading a DataConrol. With my method, you leave the original database names in the DataControl. The problem with this is that as soon as a form loads, it initializes the DataControl. It tries to load the database and record set and causes an error. To prevent this, set the Enabled property of the DataControl to false. Use this code in each form that uses a DataConrol:

```
Private Sub Form_Initialize()
    SetDataDBName frmIn:=ME,
sDBName:="C:\MYPROG\DATA\MYDB.MDB"
End Sub

'Calls the following Sub:
Public Sub SetDataDBName(frmIn As Form, sDBName As String)
Dim I As Integer
'On Error needed to prevent problems
'with refreshing bad values in the
'RecordSource property
On Error Resume Next
```

```
    'Search through form's controls
    For I = 0 To frmIn.Controls.Count - 1
        'See if the control is a datacontrol
        If TypeOf frmIn.Controls(I) Is Data Then
            frmIn.Controls(I).DatabaseName = sDBName
            frmIn.Controls(I).Enabled = True
        'Refresh if there is something
        'in the recordsource property
            If Len(frmIn.Controls(I).RecordSource) _
                > 0 Then
                frmIn.Controls(I).Refresh
            End If

        End If
    Next I

End Sub
```

That's it, easy and painless!

—David McCarter

VB4

TALK ABOUT REUSABLE CODE!

In VB4 you can create a single Splash Screen or About Screen that works with any program and automatically keeps itself up to date using the new properties of the App Object.

Create your form and make an attractive arrangement of label controls on it. Then add code such as this to the Load Event:

```
Label1.Caption = App.ProductName
Label2.Caption = App.FileDescription
Label3.Caption = "Version " & App.Major & "." _
    & App.Minor & "." & App.Revision _
    & " (" & Format$(FileDateTime(App.Path & "\" & _
    App.EXEName & ".EXE"), "Short Date") & " - " & _
    (FileLen(App.Path & "\" & App.EXEName _
    & ".EXE")) & " bytes)"
Label4.Caption = App.LegalCopyright
Label5.Caption = App.CompanyName
Label6.Caption = App.Title
Label7.Caption = App.LegalTrademarks
Label8.Caption = App.Comments
```

For graphics you can use your company's logo or put a blank PictureBox on the form, create the graphic as a separate file, and add this code to the Load Event:

```
Picture1.Picture = LoadPicture(App.Path & "\logo.bmp")
```

There are two cautions, however:

• Until the first time you compile your code, there will be no EXE file, so the FileLen and FileDateTime functions will fail. To avoid this problem, create a dummy file with the project name EXE.
• If any of the referenced App properties are not set the form will load fine in the development environment but will generate the run time error "Resource with identifier 'Version' not found" when compiled and run as an EXE (the error will say "Version" no matter which property is not set).

App properties are set from the not-too-obvious location of the Options button of the Make EXE menu selection of the File menu. Once you set the properties they will be saved with the project and do not need to be set each time you make the EXE file.

—Daniel R. Nolte

VB4

PASSING A CONTROL ARRAY

Working with control arrays in VB3 was frustrating, but with VB4 you can pass a control array as an argument to a function. Simply specify the parameter type as Variant:

```
Private Sub Command1_Click(Index As Integer)
    GetControls Command1()
End Sub

Public Sub GetControls(CArray As Variant)
    Dim C As Control
    For Each C In CArray
        MsgBox C.Index
    Next
End Sub
```

Also, VB4's control arrays have LBound, Ubound, and Count properties:

```
If Command1.Count < Command1.Ubound - _
    Command1.Lbound + 1 Then _
    Msgbox "Array not contiguous"
```

—William Storage

VB4

THE STARTMODE PROPERTY OF THE APP OBJECT

The StartMode setting in the Options dialog of the Tools menu in VB4 determines only whether an application with no startup form continues to run after Sub Main has completed. This allows testing of OLE Automation servers. But the StartMode property of the App object allows you to determine in code whether an application shows any visible interface to users. A single VB executable file can be both an invisible server and a normal application, just like Excel. The Sub Main procedure lets you test the value of App.StartMode to decide whether or not to show a form. If the application is started directly by a user, the StartMode is vbSModeStandalone. If started by a client application, the StartMode is vbSModeAutomation.

```
Sub Main
    If App.StartMode = vbSModeStandalone Then
        frmMain.Show
    Else
        'OLE server action...
    End If
```

—William Storage

VB4

PROPERTIES COLLECTION OF DATA ACCESS OBJECTS

The Properties collection of many data access objects is very helpful when debugging. Execute this code from the debug window:

```
For i = 0 to Recordset1.Properties.Count- 1:Debug.Print _
    Recordset1.Properties(i).Name &
Recordset1.Properties(i):Next
```

—William Storage

VB4

ADD PROPERTIES TO DATA ACCESS OBJECTS

How many times have you wished field objects had a "RequiredIfCondition1" or other user-defined property? You can add one easily:

```
Set NewProperty = Field1.CreateProperty("FieldNote")
NewProperty.Type = dbText
 Field1.Properties.Append NewProperty
```

—William Storage

VB4 WIN32

USE THE SLEEP API FUNCTION INSTEAD OF DOEVENTS

When in NT or Windows 95, use the Sleep API function instead of DoEvents. DoEvents does this:

```
while (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE)) {
  TranslateMessage(&msg);
  DispatchMessage(&msg);
}
```

DoEvents spends part of its time watching for other messages in the same process. This behavior has no value in a preemptive-multitasking operating system. Sleep yields more efficiently to other processes. Sleep is declared as:

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

and can be called as:

```
Sleep 0&
```

—William Storage

VB4

YOU CAN'T CREATE AN OLE DLL WITH A PUBLIC CLASS SET TO CREATABLE -- SINGLE USE

Only one copy of a DLL can exist in memory at a time. For this reason, VB doesn't allow you to create DLLs exposing single use instancing classes. An EXE, however, allows you to create OLE servers that can be instanced multiple times for multiple clients. If you need a new instance of the server for each client you must create an out-of-process server in an EXE file instead of a DLL.

—A. Nicklas Malik

VB4

USING THE FOR-EACH SYNTAX ON A PRIVATE COLLECTION OBJECT

Creating an object that looks and acts like a collection is a good way to implement a form of inheritance in VB4, where inheritance is not provided by the language. Creating properties for the Count and Item properties of your collection is not difficult. Likewise, creating an Add and Remove method is fairly easy. You can restrict the type of data added to a collection, and you can encapsulate all of the object handling in your class. However, there are two problems.

• You can expose the Item property, but you can't make the Item the default property. Instead of writing:

```
val = MyCollection("Fubar")
```

you must always write:

```
val = MyCollection.Item("Fubar")
```

• There is no way to allow a VB user to use the For-Each syntax and still keep your encapsulation complete. If you are willing to break your encapsulation, you can provide an Items property, like this:

```
Property Get Items() As Collection
    Set Items = collMyInternalCollection
End Property
```

Users can iterate your collection object like this:

```
For Each obj in MyCollection.Items
    ' Do something
Next
```

The problem with this method is that your users can now invoke the Add and Remove methods directly on the exposed collection. You have lost control of the data, and your encapsulation is broken. If you are willing to trust that your users will not use either the Add or Remove properties on this object, then this is a possible workaround.

—A. Nicklas Malik

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

VB4

ALIGN ALL YOUR CONTROLS ON A FORM

To line up controls on a form in VB4, select all the controls you want to line up. Press F4 to bring up the properties window. Then double-click on the property title of the property you want to set. This could be, for example, the word "Left" in the properties dialog. The value that appears is the value from the first control you selected.

—A. Nicklas Malik

VB4

WHEN DOES A FORM-LEVEL OBJECT INSTANCE CEASE TO EXIST?

You have an object variable declared in the Declarations section of a form. In the Form_Load event, you create an instance of a class and assign it to the variable. However, when your form unloads, the object class' Terminate event does not fire because the form has not yet ceased to exist. Simply unloading a form does not cause the form object to terminate. As long as the form object exists, it references the class. Set the form-level variable to nothing to cause it to unload.

As a way to remember this, treat a form object like any other object. Assign it to a form variable when you want to create it. This makes obvious the need to delete the reference when you are done.

Thus, instead of writing:

```
MyForm.Show vbModal
Set MyForm = Nothing
```

you should write:

```
Dim FormRef as MyForm
Set FormRef = New MyForm
FormRef.Show vbModal
Set FormRef = Nothing
```

This makes it clear that you are dropping an object reference in code.

—A. Nicklas Malik

VB4

RENAME THE "PRINTER" MODULE

Do you have a module or class of the same name as a system-provided object? If so, VB will use your object, not its own. In VB3, where you couldn't define your own objects, this was not a problem, but in VB4 it becomes an issue. Look for a module called "printer" and rename it.

—A. Nicklas Malik

VB4

GETTING A COUNT OF THE ROWS AFFECTED BY A SQL STATEMENT

SQL Server does not return a result set for action queries (SQL insert, update, or delete statements). If you were to use the SQLExecute method and execute the statement directly, RDO would catch the returning row count and make it available to you. However, since the statement is executed in a stored procedure, the row count information is not returned to ODBC. Hence, the Remote Data Object (RDO) can't return this information.

If you run a stored procedure that contains a mix of select and action statements, and an action statement fails, an error is returned to RDO. The error comes back to VB in the form of a trappable data error, which you can handle with the On Error syntax. The only thing that is not available is the actual number of rows affected by the action statement because the SQL Server does not return this information to RDO.

However, if your system must to know the actual number of rows affected, you can put:

```
Select @@ROWCOUNT
```

after any action statements in the stored procedure. This produces a one-column, one-row result set containing the number of rows affected by the action statement. DBLib and ODBC API require the same procedure as RDO.

—A. Nicklas Malik

VB4

NEW DECLARATION POSSIBILITIES FOR VB4

Add Optional parameters to your procedure calls. Both Functions and subs can now use the Optional keyword in the declaration to indicate that the following parameter is optional:

```
Function mfbCheckDBStatus(Optional _
    vroTest As Variant) As Boolean
```

This function can also be called without the parameter:

```
mfbCheckDBStatus
```

If you need to check for the existence of an optional parameter, use the IsMissing function to test the parameter.

—Crescent Tech Support

VB4

WHY DOES THE TYPENAME OF A FORM RETURN THE FORM'S NAME, AND NOT ITS TYPE?

You create a routine that uses the Typename() function to find out the type of an object, and based on that type, performs certain actions. However, when used in a Form object, the value returned by Typename() is not the generic object class "Form." It is, instead, the specific object class "Form1."

Try running this sample:

```
Private Sub Form_Load()
    Dim frm as Form
    set frm = Me
    Debug.Print TypeName(frm), frm.Name
    '<< prints: Form1
Form1
End Sub
```

You expect TypeName(frm) to resolve to "Form" instead of "Form1" but it doesn't. That is because the type of the object, in this case, is the form class itself, "Form1." That you assigned it to a Form object does nothing except to prevent you from calling any of the "Form1" object's methods and properties directly in code. Similarly, a control never gives a type name of "Control." Instead, the type name is always something like "CommandButton" or "PictureBox."

Interestingly, while the TypeName function returns the most specific name, the If-TypeOf syntax will match either the generic class type or the specific object class. In other words, both of these If statements would return True:

```
Set frm = Me
If TypeOf frm Is Form1 Then Debug.Print "Form1"
If TypeOf frm Is Form Then Debug.Print "Form"
'<< both stmts are true
```

—A. Nicklas Malik

VB4

MINIMIZING ALL WINDOWS IN WIN95

In Windows 95, only the VB windows with the property ShowInTaskbar set to True are minimized if you select Minimize All Windows from the task bar context menu. This is the way Win95 handles the window class. Forms with ShowInTaskbar=False work like property pages. You can easily demonstrate this.

Bring up the Display properties dialog from the control panel or by right-clicking on the desktop. Do a Minimize All. The property page remains. If Win95 were to minimize the window, it would sit on top of the task bar, not in it. Win95 provides this functionality, while VB4 just sets the bit.

The way that standard Win95 applications handle this is by having a main window that shows in the taskbar. All auxiliary windows are owned by the main window. When an owner window is minimized, owned windows will hide, not minimize.

You can use code to make a VB window be owned by another VB window, but there are implications to this solution. Owned windows are always on top of the owner window.

On form load of each child window, do a GetWindow API call, using GW_OWNER on the form's window handle (hwnd) to get the original owner window handle. Save this value for later use. Proceed to make the subwindow "owned by" the main window by calling SetWindowLong, like this:

```
iret =  SetWindowLong(Form2.hwnd, _
    GWL_HWNDPARENT, _
    Form1.hwnd)
```

On form unload of the main window restore all remaining (owned) subwindows back to being owned by their original owner window. Otherwise, you may get an Invalid Page Fault error.

—A. Nicklas Malik

VB4

DO YOU HAVE WRITE ACCESS TO A DRIVE WITHOUT OPENING A FILE ON THAT DRIVE?

Assume that you have the full path name of a file, and you need to be able to tell if the drive is writeable (that is, if it is a CD-ROM or a network drive to which you have write access). Checking the attributes of the file is not enough, because the file may not be marked read-only, but the drive may be a read-only drive. However, if you open the file for Write or Append inside VB, it will reset the file's date stamp, which you don't want.

To determine if a file is writeable without writing to it, get the attributes of the file with GetAttr, and then attempt to change them with SetAttr. This operation will fail if you do not have write permission on the drive. On the other hand, if you are worried about a file being locked for writing (like an EXE is while it is running under Windows), you should use the OpenFile() API call with the OF_WRITE switch. This call will generate an error if the file cannot be opened for writing, but will not change the time stamp if you simply close it right away.

—A. Nicklas Malik

VB4

DON'T REMOVE CONTROLS USING TOOLS-REFERENCES

Use the Tools-Custom Controls dialog instead of Tools-References to remove controls. The project file will show references to all custom controls in the project's toolbox, even if they are not used by any window. Using the Custom Controls dialog removes the control from the toolbox, which removes the reference.

—A. Nicklas Malik

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

VB4

MessageBox From OLE Server Comes Up Behind the App

You have a VB4-based OLE Automation server that has no main UI, and displays forms that are parented into Excel in response to various automation calls. Sometimes you need to display a message box instead of a form. If you haven't yet parented any forms into Excel, the message box frequently appears behind Excel.

To make sure that the message box appears where the user can see it—if all your forms are modal—the best way is to compile the VB4 OLE Automation server as an in-process server (DLL). The server becomes part of Excel's process space, and all forms and message boxes are automatically parented as you would expect them to be.

If your forms are modeless, you cannot use the DLL approach. Your OLE Automation server must be an EXE. The following workaround should do the trick.

Create a hidden, modeless form and use SetParent to make it a child of Excel's main window. Make this the first thing you do when Excel gets an instance of your object. Create a Public (visible throughout your project, but not outside) method for the hidden modeless form. The Public method takes the same parameters as a message box, and its job is to call MsgBox with those parameters. When your server needs to display a message box when no other form is parented to Excel, call this method. Excel should never hide the message box.

—A. Nicklas Malik

VB4

Create Custom Properties for Forms or Class Modules

Have you ever wanted to push a variable onto a form without using a tag? Property functions let you create custom properties for forms or class models. Use PropertySet, PropertyGet, and PropertyLet statements to manipulate custom properties.

For example, you want to send a key value to a form so that the key is there in time for an SQL query to use it in a condition. Include a variable declaration for the property as a form-level variable:

```
Private msKey as String
```

Then add two procedures for PropertySet and PropertyLet for the form:

```
Public Property Let GetKey(vNewValue)
        msKey = vNewValue
End Property
Public Property Get GetKey()
    GetKey = msKey
End Property
```

You can now set the property from elsewhere by using:

```
Form1.GetKey = "David"
```

In your form, use the variable you declared for any usage in your code. Property Procedures can be a bit more complex. But, the point is that the VB4 has become much more adaptable to your needs. Customize it to fit.

—Crescent Tech Support

VB4

See the Methods Exported By an OCX

You can use the code window to see the events for an OCX, and the properties window to see the properties of an OCX. To see the OCX's methods, use the Object Browser like this:

1. Start VB.
2. Press F7 to view code.
3. Press F2 to get the object browser.
4. Select an OCX from the libraries/projects drop down menu.
5. See the object's methods displayed in the Methods window.

—A. Nicklas Malik

VB4

Determine When an App is Complete

In VB3, you call GetModuleUsage() to determine when an app you started with the Shell command was complete. However, this call does not work correctly in the 32-bit arena of Windows NT and Windows 95.

To overcome this obstacle, use a routine in both 16- and 32-bit environments that will tell you when a program has finished, even if it does not create a window.

The IsInst() routine uses the TaskFirst and TaskNext functions defined in the TOOLHELP.DLL to see if the instance handle returned by the Shell function is still valid. When IsInst() returns False, the command has finished.

You can call it in a loop:

```
hInst = Shell("foobar.exe")
Do While IsInst(hInst)
    DoEvents
Loop

Function IsInst(hInst As Integer) As Boolean
    Dim taskstruct As TaskEntry
    Dim retc As Boolean

    IsInst = False
    taskstruct.dwSize = Len(taskstruct)
    retc = TaskFirst(taskstruct)
    Do While retc
        If taskstruct.hInst = hInst Then
            ' note: the task handle is: taskstruct.hTask
    IsInst = True
            Exit Function
        End If
        retc = TaskNext(taskstruct)
    Loop
End Function
```

—A. Nicklas Malik

VB4

HANDLING ERRORS IN THE FORM_LOAD ROUTINE

In VB3, the PostMessage API can cancel an error during Form_Load. The form unloads if you send a WM_CLOSE message to the loaded window in the error handler of the Form_Load routine. It is not easy to find out from the calling routine exactly why the form unloaded.

In VB4, you can create a property on your form to indicate success or failure, and unload the form from the calling procedure depending upon the value of that property:

```
Public SuccessfulLoad As Boolean
' creates the property Form1.SuccessfulLoad

Private Sub Form_Load()
    SuccessfulLoad = True
    If An Error Occurs Then
     SuccessfulLoad = False
    End If
End Sub
```

In the calling procedure:

```
Sub LoadTheFubarForm()

    Dim MyForm As Form1

    Set MyForm = New Form1
    Load MyForm
    If MyForm.SuccessfulLoad Then
     MyForm.Show vbModal
    End If
    Unload MyForm
    Set MyForm = Nothing
End Sub
```

—A. Nicklas Malik

VB4

SUPPORT THE FULL IMAGELIST API IN VB4

To draw a selected transparent image in VB4 from an Imagelist control, as you do in C++, follow these steps.

The ImageList_ API functions have many features that weren't passed through in the control. In order to support the full API, the ImageList control exposes the property hImageList, which returns a handle you can use when calling the API functions.

In the ListImage.Draw method, you can specify imlNormal, imlTransparent, imlSelected, or imlFocus. In the underlying API ImageList_Draw function, however, these are bit flags, and there is an additional flag to draw a mask.

This function draws an image from an image list with any combination of draw flags:

```
Sub DrawImage(img As ImageList, vIndex As Variant, _
    hDC As Long, x As Long, y As Long, _
    Optional vDraw As Variant)

    If IsMissing(vDraw) Then _
        vDraw = ILD_NORMAL Or ILD_TRANSPARENT

    ImageList_Draw img.hImageList, _
        img.ListImages(vIndex).Index - 1, hDC, _
        x / Screen.TwipsPerPixelX, _
        y / Screen.TwipsPerPixelY, vDraw
End Sub
```

These functions are documented on the MSDN. Take a look. You may find many other uses for them.

—A. Nicklas Malik

VB4

PLACING A COMBO BOX ONTO A TOOLBAR

To put a combo box on a toolbar, create a place holder and position the combo box above the place holder in the z-order. You can't place the combo box inside the place holder. Instead, follow these steps:

- Create a button with the PlaceHolder style.
- Show the form.
- In the Form_Load event set the Top and Left properties of the combo box to the same value as the PlaceHolder button.
- Set the z-order of the combo box to zero to bring it to the front.
- In the Form_Resize event, make sure the Top and Left properties of the combo box are the same as the PlaceHolder button.

```
Private Sub Form_Load()
    Dim btnX As Button

    Me.Show
    Set btnX = Toolbar1.Buttons.Add()
    btnX.Style = tbrSeparator
    Set btnX = Toolbar1.Buttons.Add()
    btnX.Style = tbrPlaceholder
    btnX.Key = "combo"
    btnX.Width = 2000

    With Combo1
        .ZOrder 0
        .Width = Toolbar1.Buttons("combo").Width
        .Top = Toolbar1.Buttons("combo").Top
        .Left = Toolbar1.Buttons("combo").Left
    End With
End Sub
```

—A. Nicklas Malik

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

VB4

Numeric Strings as Keys in a Treeview Control

To use numeric strings as keys in a treeview control, the keys must be strings containing a non-numeric character, not just digits. A handy workaround is to append the string "_" to the end of the string of digits, which you can now use as a key. Append to the end because you can use the Val() function to get the numeric value back out without doing any parsing.

—A. Nicklas Malik

VB4

Load New OCXs, Error-Message Free

Normally, to load a new OCX, you select Custom Controls from the VB4 Tools menu. Then, in the Custom Controls dialog, click on the browse button, select the OCX file, and click on OK to install it.

However, if the OCX does not run, it is probably missing the DLL files it needs. OCX controls created using VC++ 2.x need these files, which VB4 does not install: MFC30.DLL, MFCO30.DLL, and OC30.DLL. Make sure they're in your path and registered correctly. They should be supplied by the control.

If the OCX was created using VC++ 4.0, the files MFC40.DLL and OLEPRO32.DLL already should be installed on your system by VB4.

—A. Nicklas Malik

VB4

Error Handling in a Class Module

To signal errors from within a class method, use the Error.Raise method and bubble it back up to the server. Check the docs for Error.Raise and the books online.

One caveat is that errors in Class_Initialize and Class_Terminate *do not* bubble back up. Like errors in an event procedure of a form module, they're not in the client's call tree. If they are untrapped, the OLE server will terminate abruptly.

On an out-of-process server, your client will be left with invalid references. If your server is in-process (an OLE DLL), the situation is much more serious because your client will also terminate abruptly. (When you're in the same process, your fatal errors are your client's fatal errors.)

Therefore, always protect Class_Initialize and Class_Terminate with bulletproof error-trapping, and never raise errors in them.

—A. Nicklas Malik

VB3

Searching for 16-bit DLLs in Windows 95

Windows 95 uses an extra step when searching for 16-bit DLLs. A new registry key, "Known16DLLs," in Windows 95 under \\HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\SessionManager functions in a similar way to KnownDLLs in Windows NT.

Known16DLLs contains numerous subkeys, named after 16-bit DLLs, whose value is the name plus the DLL extension. If a 16-bit DLL is in this list, Windows 95 will first check the Windows\System directory for the 16-bit DLL and load it if it finds it. If the DLL is not in the Windows\System directory, a normal search (CurDir, windows, system, path) begins.

Windows 95 puts a subkey for a 16-bit DLL in Known16DLLs the first time it loads it from Windows\System. But it won't remove the subkey should the 16-bit DLL be no longer found in the Windows\System directory.

To summarize, when a 16-bit DLL is loaded from the Windows\System directory it goes into this list. Whenever an attempt is made to load it in the future, Win95 will see it in the list and will search Windows\System directory first. The only way to ensure the expected search for this 16-bit DLL is to remove the relevant subkey of Known16DLLs if it exists.

Special thanks to Oliver Hodgkins, Microsoft Customer Support, England.

—A. Nicklas Malik

VB4

Setup Wizard Complains: Not Enough Info in OLE Server

The SetupWizard directly supports only self-registering OLE servers. The setup toolkit doesn't directly support a server that requires a REG file.

A workaround is use the Add Files button to add an REG file. The setup toolkit will register those keys on the user's machine. This is a partial solution. You can only use relative paths, so the server directory must be on the PATH. Also, the setup toolkit doesn't know how to uninstall these changes made by a REG file.

—A. Nicklas Malik

VB4

Check Available Drive Space

The setup kit supplied with VB has a DLL with a useful function that can be called from your own applications. The function, GetFreeDiskSpace, returns the amount of free space available on a drive. Using this function in conjunction with the VB FileLen or LOF functions, you can test whether there is enough space available on a drive before you try to copy or save a file.

—Crescent Tech Support

VB4

PRINTING FROM THE RICHTEXT CONTROL

The on-line documentation on using the RichText control to print tells how it is supposed to work. It does not mention two basic problems. First, the RichText control doesn't work with the common dialog hDC. Second, to skip the error generated when using the Printer object hDC, you must use On Error Resume Next.

These code fragments show how to print properly from the RichText control:

```
On error resume next
printer.print ""
richtext.selprint printer.hdc
printer.enddoc
```

This wrapper function uses both the CommonDialog control and the Printer object:

```
Function FilePrintDlgProc(rprnDlg As _
    CommonDialog, rRTF As _
    RichTextBox) As Boolean
    On Local Error GoTo Error_Handler:
    With rprnDlg
        .CancelError = True
        .Flags = cdlPDReturnDC + cdlPDNoPageNums
        If rRTF.SelLength = 0 Then
            .Flags = .Flags + cdlPDAllPages
        Else
            .Flags = .Flags + cdlPDSelection
        End If
        .ShowPrinter
        On Local Error Resume Next
        Printer.Print ""
        rRTF.SelPrint Printer.hDC
        Printer.EndDoc

        FilePrintDlgProc = True

    End With
    Exit Function
Error_Handler:
    If Err <> cdlCancel Then
        MsgBox "Error " & Err & "; " & Error
    End If

End Function
```

—Steven Mitchell

VBA

BINDING WITH MICROSOFT WORD TYPE LIBRARY

The benefit of early binding is that performance improves and code does not need to be localized for each international version of Word. This is true only for the function that is being called, not the strings that are passed as parameters. Performance gains are two to 10 times faster, so mileage will vary.

Word is one of the easiest Office applications to completely early bind, because it has so few exposed objects. To early bind the Word Basic object, follow these steps:

- Select References from the Tools menu.
- Select Microsoft WordBasic 95 Type Library or browse for WB70EN32.TLB. This file can be obtained from Microsoft on Compuserve, Internet, or MSN.
- In the declarations section of your code add:

```
[Dim|Private|Public|Global] objReference as _
        Word.WordBasic
```

- Initialize the objReference variable by using GetObject or CreateObject:

```
Private Sub Command1_Click()
    Dim objReference As Word.WordBasic
    Set objReference = CreateObject("Word.Basic")
    With objReference
        .FileNewDefault
        .Insert Text:="Hello from Me to you."
        .InsertPara
    End With
End Sub
```

—Steven Mitchell

VB4

EARLY BINDING WITH MICROSOFT EXCEL

Early binding can be done with Excel 5.0 and 7.0 by taking advantage of two bugs in Excel. The first bug cannot do direct early binding. The second bug is an unexposed bug in which to do early binding. Be forewarned that using this method may not be compatible in future versions of Excel.

Follow these steps to early bind with the Excel object:

- Select References from the Tools menu.
- Select Microsoft Excel 5.0 or Excel 7.0 Object Library:

```
[Dim|Private|Public|Global] objXLRoot _
    as [_ExcelApplication]
[Dim|Private|Public|Global] objXLApp as Excel.Application
```

- To initialize the objXLRoot use the New method:

```
Set objXLRoot = New [_ExcelApplication]
```

- This complete code fragment illustrates the technique:

```
Private Sub Command2_Click()
    Dim objXL As Excel.Application
    Dim objXLRoot As [_ExcelApplication]
    Dim objWKB As Excel.Workbook
```

```
    Set objXLRoot = New [_ExcelApplication]
    Set objXL = objXLRoot.Application
    objXL.Visible = True

    Set objWKB = objXL.Workbooks.Add
    objWKB.Worksheets(1).Range("A2").Formula = "Hello"
    objWKB.Parent.ActiveCell.Formula = "Why"
End Sub
```

—Steven Mitchell

VBA

PROGRAMMATICALLY CLOSE THE BINDER OBJECT

Because of customer complaints and confusion, Microsoft Office applications will begin to implement this strategy for programmatically closing Office applications. First, hide the object by setting Object.Visible to false. Then let the object variable go out of scope or set it to Nothing.

```
Private Sub Command3_Click()
    Dim objBinder As OfficeBinder.Binder
    Dim objWord As Word.WordBasic

    Set objBinder = CreateObject("Office.Binder")
    objBinder.Visible = True
    Set objWord = CreateObject("Word.Basic")
    With objWord
        .FileNewDefault
        .FormatStyle Name:="Heading 1", Apply:=True
        .Insert "Really cool tip from VBPJ"
        .InsertPara
        .FileSaveAs "c:\VBPJ Tip.DOC"
        .FileClose
    End With
    Set objWord = Nothing
    With objBinder
        .Sections.Add filename:="c:\VBPJ Tip.doc"
        .Sections(1).Name = "VBPJ Tip"
        .SaveAs filename:="c:\VBPJ.obd", _
            saveOption:=bindOverwriteExisting
        .Visible = False
    End With

    Set objBinder = Nothing 'Close the Binder Object
End Sub
```

—Steven Mitchell

VB3

CHANGE YOUR EDITOR FONT

VB 4.0 lets you pick the font you want for your editor. You can do the same thing in VB 3.0 as well, but you're font choices are limited, and it'll involve a little tinkering with Windows itself.
VB 3.0 uses Windows' Fixed System font. In order to change this you need to find another fixed font. The WINCIMTE.FON font, which comes with CompuServe Information Manager is my favorite. It's small, yet very readable.

To change your editor font, follow these steps:

WINDOWS 95:
1. Run Control Panel, then Fonts. Click Add New Font from the File menu, and locate WINCIMTE.FON in the CSERVE\WINCIM directory. When installing, make sure it's copied to the Windows\System directory.
2. Restart your computer in MS-DOS mode. Change to the Windows\System directory. Copy VGAFIX.FON to VGAFIX.SAV. Copy WINCIMTE.FON to VGAFIX.FON.
3. Restart your computer. Now every program that uses the VGA Fixed System font (including Cardfile, Notepad and—you guessed it—VB 3.0) will have the new, smaller font.

WINDOWS 3.X:
1. Use Fonts from the Control Panel to install the C:\CSERVE\WINCIM\WINCIMTE.FON font.
2. Exit to DOS and perform the tasks in Step 2 for Windows 95.
3. Restart Windows. Now every program that uses the VGA Fixed System font will have the new, smaller font.

Note that the font must be a Fixed font, or VB won't like it. Also note that if you're running at high resolution with large fonts you may need to replace 8514FIX.FON instead of VGAFIX.FON.

—Barry Seymour

VB3

VB3 KEYBOARD SHORTCUTS

Take a few moments to memorize these keystroke combinations and you'll find yourself rocketing through your project! Key combinations separated by commas mean they're to be performed sequentially.

VB 3.0 Keyboard Shortcuts
Alt F,V,F5—Save and Run.
Alt+Dash—Access system menu of current editor window or form. This shortcut leads to others:
Alt+Dash, N—Minimize Window.
Alt+Dash, M—Maximize Window.
F3—Repeat last search. If you didn't have a "last search," the search dialog opens automatically.

Editor Keys
Ctrl-Up—Move to previous sub in editor window.
Ctrl-Dn—Move to next sub in editor window.
Home, Shift-end—Select an entire line of text.
Ctrl-Home, Ctrl+Shift+End—Select all text in the current sub.
Ctrl-F—Find.
Ctrl-R—Replace.

—Barry Seymour

VB4

VB 4.0 KEYBOARD SHORTCUTS

Note that some key combinations are different from those for VB 3.0:

Ctrl-T—Custom Controls.
Ctrl-E—Menu Editor.
Ctrl-F—Find (Unchanged from VB3).
Ctrl-H—Replace (H? Yes, H! Go Figure!).

—Barry Seymour

VB3 VB4

A REPLACEMENT FOR TABS

You say you want to use a tab control, but you have so many topics the tabs will be unreadable? Try using a list box in conjunction with a control array of picture controls. The list box will contain entries the user can choose; the picture controls will be containers for the various form subsections desired.

To demonstrate this, create a form with a list box (List1) on the left and a Picture box (Picture1) on the right. Set the Index property of Picture1 to zero, making it a control array. Then place this code in Form_Load:

```
Private Sub Form_Load()
    Dim x As Integer
    For x = 0 To 15
        List1.AddItem "Picture1(" & x & ")"
    If x > 0 Then Load Picture1(x)
    Picture(x).AutoRedraw = True
            Picture(x).AutoRedraw = True
            Picture(x).Visible = True
            Picture(x).Left = Picture1(0).Left
            Picture(x).Top = Picture1(0).Top
            Picture(x).Width = Picture1(0).Width
            Picture(x).Height = Picture1(0).Height
            Picture1(x).Print "This is picture " & x
    Next x
    Me.Show: Me.Refresh
    List1.ListIndex = 0

End Sub
```

In VB 4.0, you could use this syntax:

```
With Picture1(x)
    .AutoRedraw = True
    .Visible = True
    .Left = Picture1(0).Left
    .Top = Picture1(0).Top
    .Width = Picture1(0).Width
    .Height = Picture1(0).Height
End With
```

Note that for a real application you wouldn't dynamically create picture controls. You'd create them at design time and fill them with the controls you need. To demonstrate the concept, the example loads the control array at Form_Load.

Place the following code in List1_Click:

```
Picture1(List1.ListIndex).ZOrder
```

Whenever the user clicks an item in the list, the relevant picture control will pop to the top of the stack, becoming visible. This provides tab-like functionality without the cost of a VBX or extra memory, and lets you create items with as many pictures as you want, unfettered by tab width or tab caption readability!

—Barry Seymour

VB4

USING COMPONENTS

Creating a new project with a lot of old components? In VB 4.0 you can drag and drop files into the project window to add them to the project. Use File Manager or Explorer to drag and drop forms, modules, classes, or resource files onto the project window. Drag and drop OCX files (32-bit) or VBX files (16-bit) onto the toolbox to add controls to your project.

—Barry Seymour

VB4

COMPILE ON DEMAND

The new version of Visual Basic will compile only those portions of code it expects to run. This speeds load time, but the code may not be fully checked. To fully compile your project prior to running it, with all the syntax checking you've become used to, press Ctrl+F5 instead of F5 to run your project. To set VB 4.0 permanently to compile any project fully before running it, select Options from the Tools menu and click the Advanced tab. Uncheck the Compile on Demand check box to force VB to compile your app fully before running it.

—Barry Seymour

VB4

ERROR IN DBGRID DOCUMENTATION

The documentation for the DBGrid in VB4's Professional and Enterprise Editions states that the control's Text and Value properties allow you to read or set the contents of a cell. This would seem to indicate that you could use these properties to update a field in a record set to which the grid is bound. Unfortunately, that is not the case. The following code generates runtime error 438: "Object doesn't support this property or method:"

```
' This doesn't work:
DBGrid.Columns(0).Text = "Hello, world."

' Neither does this:
DBGrid.Columns(0).Value = "This is a test"
```

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

The workaround is to update the data control's record set directly. The change will be reflected automatically in the bound grid:

```
datCtl.Recordset.Edit
datCtl.Recordset.Fields(0) = "This is a test."
datCtl.Recordset.Update
```

—Phil Weber

VB4

PRINTER OBJECT QUIRK

You would think that the following code would change the current font of the default printer, but it doesn't under VB4:

```
Printer.FontName = "Arial"
Printer.FontSize = 11
Printer.Print "This is a test."
```

An undocumented feature of VB4's Printer object requires that each new page be initialized before the font can be changed. This code works as expected:

```
' Start new page
Printer.Print

' Set margins as desired
Printer.ScaleMode = vbTwips
Printer.CurrentY = 720

' Now you can set the font
Printer.FontName = "Arial"
Printer.FontSize = 11
Printer.Print "This is a test."
Printer.EndDoc
```

—Phil Weber

VB4

FIX FOR DATA CONTROL ERROR 3426

This code, which works fine under VB3, may generate runtime error 3426—"The action was canceled by an associated object"—in the 16-bit version of VB4:

```
Private Sub cmdUpdate_Click()

    ' Save contents of bound controls
    ' to underlying recordset
    datCtl.Recordset.Update

End Sub
```

The problem seems to occur because the 16-bit version of VB4, unlike VB3, does not perform an implicit Edit method when the Data control moves to a new record. The solution is to check the record set's EditMode and perform an explicit Edit method if necessary:

```
Private Sub cmdUpdate_Click()

    If datCtl.Recordset.EditMode = dbEditNone Then
        datCtl.Recordset.Edit
    End If
    datCtl.Recordset.Update

End Sub
```

Another workaround is to replace the Update method with the Data control's UpdateRecord method, which is equivalent functionally to performing an Edit followed by an Update. The drawback is that UpdateRecord does not fire a Validate event, so don't use it if you rely on that event to perform data validation.

—Phil Weber

VB4

ERROR WHEN SETTING DBCOMBO MATCHENTRY PROPERTY

When you try to set the MatchEntry property of a DBCombo control to "1 – Extended Matching"(at run time or design time), VB may report that the "Property is read-only." The problem is that Extended Matching may be used only with DBCombos whose Style is "2 – Dropdown List." Change the Style property accordingly and the error will disappear (if only Microsoft had thought to mention that in the documentation).

—Phil Weber

VB3 VB4

JUMP TO A FUNCTION WITHIN YOUR PROJECT

When you are looking at code that calls a function or procedure and you are not sure in which module the function or procedure is defined, you can highlight/procedure name and press Shift-F2. Visual Basic will open the appropriate module and display the function/procedure. This is great in large projects when you need to look for a particular module and are not sure where it is located.

—Lisa Vahey

VB3 VB4

DISPLAYING THE WINDOWS REGISTERED USER

The strings related to the registered user of a particular copy of Windows are stored in a string inside USER.EXE. You can retrieve them with code. In the general declarations section, insert:

```
Declare Function GetModuleHandle Lib "Kernel" _
    (ByVal Module As String) As Integer
Declare Function LoadString Lib "User" _
    (ByVal hInst As Integer, _
    ByVal wID As Integer, ByVal buf as Any, _
    ByVal size As Integer) As Integer
```

To get the user name and company strings into a variable, use this code:

```
Sub Form_Load ()
Dim hInst As Integer, user As String, _
    org As String, title As String, length As Integer
user = Space$(256)
org = Space$(256)
hInst = GetModuleHandle("user.exe")
length = LoadString(hInst, 514, user, Len(user))
user = Left$(user, length)
length = LoadString(hInst, 515, org, Len(org))
organization = Left$(org, length)
Debug.Print user
Debug.Print organization
End Sub
```

—Bill Reid

VB3 VB4

DRAGGING A FORM BY A CONTROL

This code is reusable and small enough to paste into whatever you're doing and instantly have a form that has no need for a title bar. In the general declarations section, insert these lines:

```
Declare Sub ReleaseCapture Lib "User" ()
Declare Function SendMessage _
    Lib "User" (ByVal hWnd As Integer, _
    ByVal wMsg As Integer, _
    ByVal wParem As Integer, lParem As Any) As Long
```

In the Mousedown event of the control, insert:

```
Sub Command1_MouseDown (Button As Integer, _
    Shift As Integer, X As Single, Y As Single)
Dim Ret&
ReleaseCapture
Ret& = SendMessage(Me.hWnd, &H112, &HF012, 0)

End Sub
```

—Bill Reid

VB3

FOR...NEXT LOOP SPEEDUP

Whenever you use For…Next loops, it is faster NOT to use the counter name after Next (of the For…Next loop). In nested loops, it produces significantly faster code.

```
Sub Form_Load ()
Dim TStart!
Dim I As Integer

TStart! = Timer
    For I = 1 To 32000
    Next I
Debug.Print Timer - TStart!

TStart! = Timer
    For I = 1 To 32000
    Next ' I ' For readability, it is better
             ' to show the counter name
             ' by commenting it out.
Debug.Print Timer - TStart!

End Sub
```

—Mansoor A. Thange

VB3

SECOND EDITABLE CODE WINDOW IN VB

Look for a thin horizontal line at the top of the client area in the VB code window that comes up when you view code for a form or a module. If you move the mouse pointer just above that line, you can drag the bar down to reveal another editing code window, initially viewing the same module of code. You also can load a different module to view both sets of code on the same screen. This is extremely useful when you need to look at code or declarations in one section of a project while writing code in another. You can use this in the same module only. Another module will have its own code window.

—Lawrence M. Rice

VB3

CREATE A CONTROLLED DOEVENTS

I needed a way to issue a DoEvents to allow Windows time to redraw controls on a form, but wanted to prevent the user from clicking some other control within my application.

I created a tiny form called F_DoEvents with the following properties:

```
ControlBox = False
MaxButton = False
MinButton = False
BorderStyle = none
```

```
Sub Form_Load
    Left = Width * -1
    Top = Height * -1
End Sub
```

This makes the form essentially invisible. The only code in the form is:

```
Sub Form_Activate
    DoEvents
    Me.Hide
End Sub
```

Now, whenever I want to issue a limited DoEvents, I code:

```
    F_DoEvents.Show 1
```

Because the form is displayed modally, the current code is suppressed while the F_DoEvents form is "displayed." This form issues the DoEvents and then hides itself, returning control to the "caller." I experienced no noticeable degradation in performance. Hiding the form is much faster than unloading it each time.

One warning: be sure to terminate your application with an END statement or explicitly unload F_DoEvents when you have finished. Unloading the main form won't remove F_DoEvents and the application will appear to hang.

—Robert W. Boyd

VB3 VB4

SELECTING THE TEXT WHEN ENTERING A FIELD

In many cases, it is best to select or highlight the full text of a field when that field gains the focus. This allows the user to simply begin typing to replace the original text with new text or to press tab to leave as is and move to the next field. This implements with very little code. In a global module, include this code:

```
Sub SetSelected()
    Screen.ActiveControl.SelStart = 0
    Screen.ActiveControl.SelLength =
Len(Screen.ActiveControl.Text)
End Sub
```

For each field, include this text:

```
Sub txtField_GotFocus()
    SetSelected
End Sub
```

—Gordy Blackwell

VB3

USING ENTER TO TAB FORWARD THROUGH USER ENTRY FIELDS

When users are comfortable using the Enter key to proceed to the next field, you can allow them to continue their old habits. You need very little coding to accomplish this. Set the KeyPreview property of the form to True, and include the following code:

```
Sub Form_KeyDown (KeyCode As Integer, Shift As Integer)
    ' To trick the system to think
    ' the Tab key was pressed
    If KeyCode = 13 Then SendKeys "{TAB}"
End Sub

Sub Form_KeyPress (KeyAscii As Integer)
    ' To eliminate the annoying beep
    If KeyAscii = 13 Then KeyAscii = 0
End Sub
```

—Gordy Blackwell

VB3

COUNT THE ROWS IN A TABLE BEFORE SELECTING THOSE ROWS

This technique easily helps you avoid an Object not Set error, and saves time by not sending unnecessary queries:

```
Sub Form1_Load ()
    Dim db As database
    Dim ds As snapshot
    Dim iNum As Integer

    Set db = OpenDatabase("c:\vb\biblio.mdb")
    Set ds = db.CreateSnapshot("Select Count (*) _
        from Authors Where AU_ID > 10")
    iNum = ds(0)
MsgBox "The number is " + Str$(iNum)
End Sub
```

—Peter Chyan

VB3

DETECT NULL OR ZERO-LENGTH FIELDS IN A DATABASE

This syntax is much faster than using an If…Then…Else construct to detect null or zero-length fields, and it will ensure that an error will not occur.

For strings:

```
Dim sVar As String
…
sVar = "" & ds!sField
```

For numbers:

```
Dim nVar As Integer
…
nVar = 0 & ds!nField
```

This assumes that ds is a dynaset and sField and nField are two fields in that dynaset.

—Hendrick H. Heimer

VB3

PRINTER OBJECT ENDDOC RESETS FONTNAME

The Printer object's EndDoc method resets the FontName property. If you want your application to print separate documents, invoking EndDoc at the end of each, set the FontName before printing the first item.

—Duanne Morse

VB3

ALLOW MULTIPLE FILE NAMES IN AN OPEN FILE DIALOG BOX

If you want to allow multiple file names in an open file dialog box (Flags OFN_ALLOWMULTISELECT), the resulting FileName string will consist of the drive and directory path followed by a space-separated list of the files selected in that directory. For example, selecting files X.TXT and Y.TXT in c:\a\b will result in a file name of "c:\a\b\ x.txt y.txt" and NOT "c:\a\b\x.txt c:\a\b\y.txt."

—Duanne Morse

VB3 VB4

PROGRAMMATICALLY CLOSE ANOTHER PROGRAM

To have a program programmatically close another program, use this code:

```
' Close an existing program

    Title = "VBApp"
    ihWnd = FindWindow(0&, Title)
    ihTask = GetWindowTask(ihWnd)
    iRet = PostAppMessage(ihTask, WM_QUIT, 0, 0&)
    MsgBox "Check it out! The VB App should be history"
```

—Douglas Haynes

VB3

SS3D2.VBX AND WIN95 GPF

Running applications written in VB3 in the WIN95 OS is not always a clean conversion. This code would work fine in Windows for Workgroups 3.11, but would GPF in WIN95. SS3D2.VBX, which is from Sheridan Software and contains SSCombo and SSList, among others. I got a GPF crash from Win95 when I placed the SSCombo on top of the SSIDXTAB.VBX upon exit. The workaround is to type in the Form_Unload event:

```
Dim rc as integer
rc = setparent(sscombo1.hwnd, form1.hwnd)
```

The workaround makes the combo a child of the form, not of the index tab, during the unload. In addition to, add in the General Declarations:

```
Declare Function Setparent Lib "User.exe" _
    (Byval hwndchild as integer, byval  _
    hwndparent as integer) as integer.
```

—Douglas Haynes

VB3

GIVE FOCUS TO ANOTHER 16-BIT APPLICATION IN NT

Note that in writing in VB3.0, Visual Basic's AppActivate statement fails to make a 32-bit application the active window under Windows NT. For example:

```
Sub Form_Load ()
    AppActivate "Notepad - (Untitled)"
End Sub
```

Visual Basic fails to give focus to the Notepad session because the 16-bit Windows subsystems may not be fully available to other 16-bit programs. To work around this, use the FindWindow and SetWindowPos Windows API functions like this:

1. Start a new project in Visual Basic. Form1 is created by default.
2. Double-click on the form to open the code window. Select (general) from the Object box. Enter the following in the (general) (declarations) window:

```
Declare Function FindWindow% Lib "USER" _
    (ByVal Class&, ByVal Caption$)
' The following Declare statement must be on one line:
Declare Sub SetWindowPos Lib "user" _
    (ByVal hwnd%, ByVal hwndAfter%, _
    ByVal x%, ByVal y%, ByVal cx%, _
    ByVal cy%, ByVal swp%)
```

3. Select Form from the Object box. Add the following code to the Form Click event:

```
Sub Form_Click ()

        Const SWP_NOSIZE% = &H1
        Const SWP_NOMOVE% = &H2
        AppActivate "Notepad - (Untitled)"
        x = FindWindow(0, "Notepad - (Untitled)")
        SetWindowPos x, 0, 0, 0, 0, 0, _
            SWP_NOSIZE Or SWP_NOMOVE
        Debug.Print Hex$(x)
        ' Print return code from
        ' FindWindow API function.

    End Sub
```

4. Start Notepad in Windows NT.
5. Start the Visual Basic program, or press the F5 key. Click on the form to activate Notepad. When finished, close the form to end the Visual Basic program.

—Douglas Haynes

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

VB3 VB4

ALLOW COMMAND LINE INI FILES FOR MULTIPLE USERS ON THE SAME PC

To allow different INI files for different users on the same machine, use the command line to specify the specific INI file path and file name. Add a check to see if the INI file exists, and if it doesn't, create it. A different icon specifying a different INI file on the command line can be set up for each user:

```
Sub Form_Click ()
Dim achIniFile As String
Dim Msg As String
    If Command = "" Then  ' If no command line.
achIniFile = App.Path & "\DEFAULT.INI"
        '    There is currently no command-line string."
Msg = "The INI file used is: '" & Command$ & "'"

    Else ' Put command line into message.
achIniFile = App.Path & "\" & Command$
        Msg = "The INI file used is: '" & Command$ & "'"
    End If
    MsgBox Msg   ' Display message.
End Sub
```

—Don Yasuda

VB3 VB4

AVOID AN ODBC ERROR REFERENCING SP_STATISTICS

After opening a record set on a SQL server, this ODBC error occurs:

```
    ODBC--call failed.
[Microsoft][ODBC SQL Server Driver][SQL Server]
    (#20001)
```

Attach the SQL Server table and open a record set on the attached table to solve the problem.

Do not use the OpenDatabase method to open the record set while a transaction is pending. SP_STATISTICS, a catalog stored procedure, retrieves information about the table on which you create the record set. SQL Server does not allow this stored procedure to run while a transaction is pending.

—Douglas Haynes

VB3 VB4

ENSURE CODE IS DELETED WHEN A CONTROL IS DELETED

When a control is deleted from a form, any code behind that control is not deleted. It still exists in the code in the General area. Such stranded code might not be a problem except that it increases the size of your EXE, consumes memory, and makes for sloppy work. After deletion, if a control is placed on the form with the same name as the deleted control, the stranded code will relocate to the new control, as long as the event name is the same. A programmer that does not want this code left in (and who would?) must manually delete each instance of stranded code.

—Douglas Haynes

VB3 VB4

CORRECTLY CONVERT SQL SERVER FLOATS BY JET

When using the SQL Server ODBC driver and SQL Server, and If ODBC prepared execution is used, certain floating-point values may be incorrectly converted. Microsoft Access and Microsoft Visual Basic commonly use the ODBC prepared execution. For example:

```
Dim db As Database
Dim ds As Dynaset

 Set ds = db.CreateDynaset("SELECT * FROM test")
ds.AddNew
ds.Fields("col1") = 3.9
ds.Update
```

A query that checks for equality of the float column to the value inserted does not show the record inserted, whereas a nonqualified query shows the record. For example, the record set for the ds1 dynaset does not show the record inserted, but ds2 dynaset will:

```
    Set ds1 = db.CreateDynaset("SELECT * FROM test _
        WHERE col1=3.9")

    Set ds2 = db.CreateDynaset("SELECT * FROM test")
```

The difference in behavior is because, in the case of prepared execution, the ODBC driver is doing the conversion to float. In the case of nonprepared execution and DB-Lib client tools, SQL Server is doing the conversion.

To work around this problem, do an explicit convert on the SQL Server using a statement similar to this:

```
    UPDATE test SET foo= (CONVERT(FLOAT, _
        CONVERT(VARCHAR, col1)))
```

You can do the same thing within a trigger to automatically update the value for all new records inserted. Please note that this problem does not occur using the pass-through mechanism because in that case, the conversion is done by SQL Server.

—Douglas Haynes

VB3 VB4

SUCCESSFULLY CLOSE ODBC CONNECTION

The Microsoft Access engine will maintain a persistent connection on an ODBC connection in order to be more efficient, even after using a Close method on a database opened with ODBC. The ODBC database process keeps running. To close the connection successfully, you must end the Visual Basic application. Even though this is by design, many times the connection is unwanted. One way to force the connection closed is to set ConnectionTimeout to the minimum setting of one second. A setting of zero indicates to never close the connection. It is defaulted to a value of 600 seconds, or 10 minutes. If a Visual Basic program does not reopen the ODBC connection after doing a Close method, a timeout occurs and the connection closes automatically. You can control the timeout period by placing the following line in your VB.INI or <vb_exe_app_name>.INI file, where x is the number of seconds:

```
[ODBC]
ConnectionTimeout=x
```

To enforce the fastest possible timeout, you can set ConnectionTimeout to one. In addition, you can add this code after you close the database to make sure the connection is terminated:

```
db.Close          ' Close database, using
                  ' database object variable (db).
Start = Timer
Do                ' This loop pauses a second
                  ' to allow a time-out
    FreeLocks     ' Tell Microsoft Access
                  ' engine that program is idle.
    DoEvents      ' Tell Windows to do any
                  ' pending events.
Loop While Timer <= Start + 1
```

This loop delays for a second after the db.Close. The FreeLocks statement tells the database engine that the user is idle. If you run the Visual Basic program with ConnectionTimeout set to one in your VB.INI or <vb_exe_app_name>.INI file, the database engine will disconnect the one-second-old connection to the server.

—Douglas Haynes

VB3 VB4

GPF IN VB.EXE

What do you do when you get a GPF in VB.EXE? Remember that Windows requires you to ensure memory integrity when calling API functions. A GPF in VB.EXE can be produced if an API is incorrectly called. An example of the error is:

```
Application error: VB caused a General
Protection Fault in VB.EXE
at nnnn:nnnn
```

or one of the following error messages:

```
- Assertion failed.
- Bad handle.
- Bad heap block.
```

This can occur if any of these conditions are passed to an API:

- Incorrect placement of ByVal in the Declare statement.
- A passed string initialized to a value that is too short to receive the return value.
- Undefined parameters in the function declaration or invocation.
- Incorrect type or length of parameters in the function declaration or invocation.

—Douglas Haynes

VB3

TO SYNC OR NOT TO SYNC (ASYNC) ODBC QUERIES

Here you are in VB3, you have loaded the compatibility layer for Jet 2.0, and you are using ODBC. In this configuration, the ODBC queries will always run in asynchronous query execution mode, no matter if DisableAsync is set to one or zero in the application's initialization file. Fortunately, the fix for this is readily available in the form of a new DLL in the Microsoft Access Service Pack, which upgrades the Jet Engine (MSAJT200.DLL) to version 2.50.1606.

When the VB compatibility layer is installed, the Jet Engine is upgraded from version 1.0 to version 2.0. If the Microsoft ODBC Desktop Database Drivers are then installed, the Jet Engine (MSAJT200.DLL) will be upgraded to version 2.50.1117, which forces a VB application into asynchronous query execution mode forever.

—Douglas Haynes

VB3

CREATE AN ACCESS/QUICKEN-LIKE COMBO BOX

This example shows how to make a combo box act like those found in Quicken or Microsoft Access, without using a third-party VBX. As each character is typed, the elements in the combo box are searched and, if a match is found, retrieved. If a match is not found, the original typed text is restored. The only code needed resides in the KeyUp event of a combo box called combo1.

The combo box is sorted alphabetically, so it stops on the first alphabetical match. I use the SendMessage API to turn the redraw of the combo box off and then on when the search is complete.

Here are the form-level declarations for the combo box:

```
Dim strCombo As String
Const WM_SETREDRAW = &HB
Const KEY_A = 65
Const KEY_Z = 90
```

```
Declare Function SendMessage Lib "User" _
     (ByVal hWnd As Integer, _
     ByVal wMsg As Integer, _
     ByVal wParam As Integer, _
     lParam As Any) As Long
```

The code in the KeyUp event looks like this:

```
Dim x%
Dim strTemp$
Dim nRet&

If Keycode >= KEY_A And Keycode <= KEY_Z Then
'only look at letters A-Z
     strTemp = combo1.Text
     If Len(strTemp) = 1 Then strCombo = strTemp
     nRet& = SendMessage(combo1.hWnd, _
          WM_SETREDRAW, False, 0&)
     For x = 0 To (combo1.ListCount - 1)
          If UCase((strTemp & _
               Mid$(combo1.List(x), _
               Len(strTemp) + 1))) =
UCase(combo1.List(x)) Then
               combo1.ListIndex = x
               combo1.Text = combo1.List(x)
               combo1.SelStart = Len(strTemp)
               combo1.SelLength = _
                    Len(combo1.Text) - (Len(strTemp))
               strCombo = strCombo & _
                    Mid$(strTemp, Len(strCombo) + 1)
                Exit For
          Else
               If InStr(UCase(strTemp), _
                    UCase(strCombo)) Then
                    strCombo = strCombo & _
                    Mid$(strTemp, Len(strCombo) + 1)
                    combo1.Text = strCombo
                    combo1.SelStart = Len(combo1.Text)
               Else
                    strCombo = strTemp
               End If
          End If
     Next
     nRet& = SendMessage(combo1.hWnd, _
          WM_SETREDRAW, True, 0&)
End If
```

—Dan Fox

VB3 VB4

MSDN IS INDISPENSABLE FOR THOSE OFF-THE-WALL ERRORS

Use the Microsoft Developer Network to search for errors that seem to defy explanation. VB4 comes with a short version of the MSDN, and subscriptions are available through Microsoft.

—Douglas Haynes

VB3 VB4

HANDLING LONG INI FILE ENTRIES

INI file entries can be so extremely long that the normal way of sizing a string before retrieving the entry may not be sufficient. This most frequently is true when retrieving the keyword names for an entire INI file section.

Many programmers code something like this:

```
IniEntry = Space$(512)' let's hope 512
                          ' is big enough
Result = GetProfileString(Section, _
     KeyWord, "", IniEntry, Len(IniEntry))
If Result > 0 Then
     IniEntry = Left$(IniEntry, Result)
```

However, there is no indication that the returned string may have been truncated. This technique will accommodate any length INI string:

```
IniEntry = ""
Do
     IniEntry = IniEntry + Space$(512)
     Result = GetProfileString(Section, _
          KeyWord, "", IniEntry, Len(IniEntry))
Loop Until Right$(IniEntry, 1) = " "
```

If IniEntry is not long enough, the rightmost character will contain a null character, so the loop will repeat until the rightmost character remains as a space.

The same technique works when retrieving all keywords of a section like this:

```
KeyWords = ""
Do
     KeyWords = KeyWords + Space$(512)
     Result = GetProfileString(Section, 0&, "", _
          KeyWords, Len(KeyWords))
Loop Until Right$(KeyWords, 1) = " "
```

This assumes that the "KeyWord" parameter for GetProfileString has been declared as:

```
ByVal lpKeyWord As Any
```

—Phil Parsons

VB3 VB4

KEEPING ACCURATE TIME IN VB

I developed a CBT project which required the user to read large amounts of text. I wanted to prompt the user to take a break after a period of time. The problem with the VB timer is that it lasts just over a minute. I used the API function GetCurrentTime(), which records the milliseconds since Windows was started.

Place a Timer control on the form that starts the application (make sure that this form remains loaded throughout the application). Place this code in the Declarations section of the form:

```
Dim Start&, Elapsed&
Declare Function GetCurrentTime& Lib "User" ()
```

The Form_Load event of this form must also contain this code:

```
Start = GetCurrentTime
```

This routine sets the Timer interval to about a minute.

```
Sub Timer1_Timer ()

    Dim MsgText$

    Elapsed = GetCurrentTime()
    ' if 10 minutes has elapsed since
    ' the program was started
    ' or the last msgbox was displayed
    If Elapsed - Start >= 600000 Then ' 10 minutes
        MsgText = "Give your eyes a rest. _
            Take a 5 minute break."
        MsgBox MsgText, 16, "Take A Break"
        ' however long the msgbox is on the screen
        ' the timer is effectively set to 0 when the
        ' user presses OK
        Elapsed = GetCurrentTime()
        Start = Elapsed
        Elapsed = 0
    End If
End Sub
```

—David Mawson

VB4

COOL SCREEN WIPES

You can achieve some cool form wipes with judicious use of the Move method. For example, to draw a curtain from right to left use this routine:

```
Sub WipeRight (Lt%, Tp%, frm As Form)
        Dim s, Wx, Hx, i
        s = 90 'number of steps to use in the wipe
        Wx = frm.Width / s    'size of vertical steps
        Hx = frm.Height / s   'size of horizontal steps
        ' top and left are static
        ' while the width gradually shrinks
        For i = 1 To s - 1
                frm.Move Lt, Tp, frm.Width - Wx
        Next
End Sub
```

Call the routine from a command button by using this code:

```
L = Me.Left
T = Me.Top
WipeRight L, T, Me
```

It is also possible to wipe a form from bottom to top, and from both sides to the middle, using similar routines.

—David Mawson

VB3

BEWARE OF DESIGN-TIME DDE LINKS

When using DDE Links at design time, the link will be temporarily cut when the program is run. Some, but not all, programs will reestablish the connection.

—Douglas Haynes

VB3 VB4

CHANGE TAB ORDER SEQUENCE

While in design mode, the tab order may get out of sequence, especially if you add a control after all the others are set in place. You can set the TabOrder sequence by selecting each control in reverse order with your mouse, and setting the TabOrder property to zero for each one. When done, you will find that all the orders are in the reverse of the order you chose: they are now in the correct sequence.

—Douglas Haynes

VB3 VB4

TAB ORDER WITH LABELS ATTACHED TO CONTROLS

In order for a label to be associated with a control (such as a text box) for using hot keys, they must be in TabOrder sequence, the label being one more than the text box.

—Douglas Haynes

VB3 VB4

PRINT A SINGLE SUB OR FUNCTION

The problem? You want to print a single sub but VB always prints all the subs that are in the current form. You could copy it to the clipboard and print it with the MS-Editor or Notepad. You could buy expensive tools to do this seemingly simple task. Here's a less annoying solution to the problem. Generate a small form. Add a command button. The caption may be "Print Sub." Then, add this code to the module:

```
Sub Command1_Click ()
     '    Make sure that VB has the focus
     AppAcitvate "Microsoft Visual Basic"
     '    Move the cursor to the upper left corner.
     '    The sub can contain a maximum of 500 lines.
     '    send SHIFT together with the DOWN key
     '    to mark the whole sub.
SendKeys "{Home}{UP 500} + ({DOWN 500}) % (EC)", True
     '    Copy SUB to the clipboard
Printer.Print Clipboard.GetText()
     '    Print Clipboard
Printer.EndDoc
'    Un-mark the SUB
SendKeys "{HOME}"
End Sub
```

Generate the EXE and run it together with VB. Move the Print-Sub form to a convenient desktop position.

If you want a single sub of your project to be printed, set the cursor to any position in this SUB and click on the Print-Sub button.

I've tested the code with VB3 Pro. In other VB Versions (such as the German Standard edition) you might have to change the "%(EC)" portion. "EC" refers to the VB menu bar.

—Andreas Schilling

VB3 VB4

SCREAM THROUGH SEARCHES WITH BYTE ARRAYS

This subroutine shows how byte arrays can speed a search though a file. The routine is called with the file name, a string to look for, a flag that tells it to use a string variable or a byte array and a flag that tells it to look for Unicode or ANSII strings (strings in VB4 EXEs are in Unicode). My test, with ibyte set to true (use byte array) took about six seconds to search though 32-bit WINWORD.EXE. The same file with ibyte set to false (use string variable) ran in about 36 seconds. The file is almost 4 MB.

```
Sub Searchfile(sFile As String, sSearch As String, ibyte _
     As Boolean, iUniCode As Boolean)
     'sFile - file name
     'sSearch - string to search for
     'ibyte - use byte array to search
     'iUniCode - look for UniCode strings
     Dim iHandle As Integer
     Dim sTemp As String
     Dim lSpot As Long
     Dim lFind As Long
     Dim sSearch1 As String
     Dim bTemp() As Byte
     'another advantage of using a byte array
     'is that we can easily look for UniCode strings
     If iUniCode Or (Not ibyte) Then
          'this line will look for unicode strings
          'when using byte arrays, regular
          'strings when using string variable
          sSearch1 = sSearch
     Else
          'this line will look for ANSII strings
          'when looking through a byte array
          sSearch1 = StrConv(sSearch, vbFromUnicode)
     End If

     iHandle = FreeFile
     Open sFile For Binary Access Read As iHandle
     If iHandle Then
          sTemp = Space$((LOF(iHandle) / 2) + 1)
          ReDim bTemp(LOF(iHandle)) As Byte
          If ibyte Then
               Get #iHandle, , bTemp
               sTemp = bTemp
          Else
               Get #iHandle, , sTemp
          End If
          Close iHandle
     End If

     Do
          If ibyte Then
               lFind = InStrB(lSpot + 1, sTemp, _
                    sSearch1, 1)
          Else
               lFind = InStr(lSpot + 1, sTemp, sSearch1, 1)
          End If
          lSpot = lFind
     Loop Until lFind = 0
End Sub
```

— MicroHelp UnInstaller 95 team

VB4

CONVERTING FILE NAMES

VB4's commands for dealing with file names (such as KILL, MKDIR, and FILECOPY) support long file names without programmer interaction. A number of the Win95 API functions will return only the short name, and you'll notice a number of short file name entries if you're digging through the registration database. Therefore, occasionally you'll need to convert a short file name into a long file name.

This function lets you pass a long file name with no ill effects. The file must exist for the conversion to succeed. Because this routine uses Dir$ and "walks" the path name to do its work, it will not impress you with its speed:

```
Function sLongName(sShortName As String) As String

    'sShortName - the provided file name,
    'fully qualified, this would usually be
    'a short file name, but can be a long file name
    'or any combination of long / short parts
    'RETURNS: the complete long file name,
    'or "" if an error occurs
    'an error would usually indicate
    'that the file doesn't exist

    Dim sTemp As String
    Dim sNew As String
    Dim iHasBS As Integer
    Dim iBS As Integer

    If Len(sShortName) = 0 Then Exit Function
    sTemp = sShortName
    If Right$(sTemp, 1) = "\" Then
        sTemp = Left$(sTemp, Len(sTemp) - 1)
        iHasBS = True
    End If
    On Error GoTo MSGLFNnofile
    If InStr(sTemp, "\") Then
        sNew = ""
        Do While InStr(sTemp, "\")
            If Len(sNew) Then
                sNew = Dir$(sTemp, 54) & "\" & sNew
            Else
                sNew = Dir$(sTemp, 54)
                If sNew = "" Then
                    sLongName = sShortName
                    Exit Function
                End If
            End If
            On Error Resume Next
            For iBS = Len(sTemp) To 1 Step -1
                If ("\" = Mid$(sTemp, iBS, 1)) Then
                    'found it
                    Exit For
                End If
            Next iBS
            sTemp = Left$(sTemp, iBS - 1)
        Loop
        sNew = sTemp & "\" & sNew
    Else
        sNew = Dir$(sTemp, 54)
    End If
MSGLFNresume:
    If iHasBS Then
        sNew = sNew & "\"
    End If
    sLongName = sNew
    Exit Function
MSGLFNnofile:
    sNew = ""
    Resume MSGLFNresume
End Function
```

— MicroHelp UnInstaller 95 team

VB4

USE A CLASS TO CONVERT CODE

In VB3 if you wanted to convert your code from one custom control to another, there were no options other than changing your code. By writing the proper class you can make use of your current syntax with a different control. This code demonstrates how to write a VB class to change standard list-box syntax to use a ListView control. MyList.AddItem and .RemoveItem methods and the .Text property will interface to the ListView Control using this class:

```
'Form1 would have a listView control on it
'dim a module wide object variable
Dim MyList As Object

Private Sub Form_Load()
    'create an instance of our class
    Set MyList = New FakeList
    'tell the class what control to use
    Set MyList.ListItem = ListView1
End Sub

'The Class code follows
'This holds the listview control
'that we will interface with
Public ListItem As ListView

Public Sub AddItem(sString As String)
    'convert AddItem to Add
    ListItem.ListItems.Add , , sString
End Sub

Public Sub RemoveItem(lItem As Long)
    'Convert RemoveItem to Remove
    ListItem.ListItems.Remove lItem
End Sub

Public Property Get Text() As String
    'Get the text from the selected item
    Text = ListItem.SelectedItem.Text
End Property

Public Property Let Text(sString As String)
    'set the text in the selected item
```

```
    ListItem.SelectedItem.Text = sString
End Property
```

—MicroHelp UnInstaller 95 team

VB3 VB4

CHOOSE COMPARES CAREFULLY

A straight ASCII compare, such as 'F "A" < "B" THEN' is much faster then using VB's StrComp("A", "B", 1). However, when you use this test to sort an array of strings, your result will not be "internationally" correct. Using an ASCII compare simply tests the string for which the ASCII values of one character come before those of another. The StrComp() function uses the current code page setting in Windows to determine which characters alphabetize before others based on the current country setting. Using the StrComp() when sorting and displaying output to your user will make your application much more internationally friendly.

If you are sorting and testing for internal purposes, an ASCII compare is much faster. Make sure you are consistent with the compares or your results might not be what you expect. Don't do a binary chop against an alphabetized array using an ASCII compare or vice versa.

On the same note, using UCase$() (or LCase$()) is very fast. However, these functions only zip through a string, setting or clearing the fifth bit of characters "A-Z," "a-z." To adjust case properly for international character sets, use StrConv(string$, vbUpperCase|vbLowerCase). Once again, for your own code's internal use, use UCase$()/LCase$() because they are faster, but for display purposes use StrConv().

During testing of UCase$() .versus StrConv(), both returned the same results under VB4. However, since StrConv() provides other services you may need (or in case this was simply a Microsoft bug), you should still use StrConv() in place of UCase$()/LCase$().

— MicroHelp UnInstaller 95 team

WIN95

TRIM A FILE PATH FOR DISPLAY IN A TEXT BOX

With the advent of long file names in Windows 95, it may be necessary to display a trimmed version of a path in a text box or label. The following function takes in a long file path and creates a trimmed version. For example, C:\MY VERY LONG DIRECTORY\AND LONG SUBDIRECTORY\AND ANOTHER SUBDIRECTORY\AND LONG FILENAME.TXT becomes C:\MY VERY LONG DIRECTORY\...\AND LONG FILENAME.TXT.

This function takes the full path to the file along with the maximum length that can be displayed in the text box, label, etc. It uses SGBkwdInstrS to find backslashes. The function is supplied below TruncatePath in the sample:

```
Private Function TruncatePath(ByVal sFileName _
    As String, iMaxLen as Integer) As String
    If Len(sFileName) Then
        Dim iPos As Integer, iPos0 As Integer, _
            iPos1 As Integer, iPos2 As Integer, _
            iPos3 As Integer, iPos4 As Integer
        iPos = SGBkwdInstrS(0, _
            Left$(sFileName, Len(sFileName) - 1), "\")
        iPos0 = InStr(sFileName, ":")
        iPos1 = InStr(sFileName, "\")
        iPos2 = InStr(iPos1, _
            sFileName, "\"): iPos2 = iPos1 + iPos2
        iPos3 = InStr(iPos2, sFileName, _
            "\"): iPos3 = iPos2 + iPos3
        iPos4 = InStr(iPos3, _
            sFileName, "\"): iPos4 = iPos3 + iPos4
        If Len(sFileName) > iMaxLen Then
            If (iPos4 <> 0) And _
                iPos4 +Len(Right(sFileName, iPos)) _
                <= iMaxLen - 2 Then
                sFileName = Left$(sFileName, _
                    iPos4) & "..." & Right(sFileName, _
                    Len(sFileName) - iPos)
            ElseIf (iPos3 > 0) And _
                iPos3 + Len(Mid$(sFileName, _
                iPos)) <= iMaxLen - 2 Then
                sFileName = Left$(sFileName, _
                iPos3) & "..." & _
                Right(sFileName, Len(sFileName) - iPos)
            ElseIf (iPos3 > 0) And _
                iPos3 + Len(Mid$(sFileName, iPos)) _
                <= iMaxLen - 2 Then
                sFileName = Left$(sFileName, iPos2) & _
                    "..." & Right(sFileName, _
                    Len(sFileName) - iPos)
            Else
                sFileName = Left$(sFileName, iPos0 + 1) _
                    & "..." & Right(sFileName, _
                    Len(sFileName) - iPos)
            End If
        End If
    End If
    TruncatePath = Left$(sFileName, Len(sFileName) - 1)
End Function

Function SGBkwdInstrS(ByVal iStart As Integer, _
    ByVal sTarget As String, ByVal SPattern As String)
    Dim IPtr As Integer, IPLen As Integer
    IPLen = Len(SPattern)
    If ((Len(sTarget) = zero) Or (IPLen = zero) Or
(Len(SPattern) > Len(sTarget))) Then Exit Function
    If (iStart = zero) Then iStart = 1
    If (iStart >= (Len(sTarget))) Then iStart =
Len(sTarget)
    iStart = Len(sTarget) - iStart + 1
    On Error Resume Next
    For IPtr = iStart To 1 Step True
        If (SPattern = Mid$(sTarget, IPtr, IPLen)) Then
            'found it
            SGBkwdInstrS = IPtr
            Exit For
        End If
    Next IPtr
End Function
```

—MicroHelp UnInstaller 95 team

VB4

UNDERSTANDING UNICODE

VB4 introduces the use of double-byte characters. Most of this is transparent to the programmer and requires no special consideration. When calling API functions or reading/writing to a file VB will handle the conversion for you automatically.

However, there may be times when you want to force a condition that goes against VB's will. For example, you might want to write Unicode to a file, pass a Unicode string to a function, or receive a Unicode string from a routine. In these cases you will have to use VB4's new Byte declaration. A String Byte can vary between one or two bytes depending upon how it is used. A Byte-byte is exactly that: one byte.

To convert a string variable into a byte array, use this code:

```
Redim MyByteArray(0 to len(MyString$)-1) as Byte
MyByteArray() = StrConv(MyString$, vbFromUniCode)
```

To convert a byte array to a string:

```
MyString$ = StrConv(BA(), vbUniCode)
```

Due to a bug or design limitation, VB4 does not allow you to convert a string to a binary array that is part of a Type structure. For example:

```
TYPE MyByteType
    Bytes( 0 to 255) as Byte
END TYPE

Dim MBA as MhByteType

MBA.Bytes() = StrConv(MyString$, vbFromUniCode)
```

returns an error. However,

```
MyString$ = StrConv(MBA.Bytes() , vbUniCode)
```

works as expected.

— MicroHelp UnInstaller 95 team

VB4

LIMITATIONS OF IMAGELIST CONTROL

Each image that is displayed in a TreeView or ListView control must first be placed in an ImageList control. If many items are placed in the control, it will run out of memory (error 7). To avoid this, reuse ImageList items whenever possible. For example, if you are representing percentages, draw 100 items, each representing a percentage, in the ImageList and reuse them. To avert memory problems when you use the ImageList control, draw the percentage item only when it is required.

— MicroHelp UnInstaller 95 team

VB4 WIN95

DOEVENTS() AND PREEMPTIVE MULTITASKING

Windows 95 is a preemptive multitasking system. As a result of this, no single application monopolizes the entire CPU at any given time. Many of the DoEvents() calls previously used in Windows 3.1 are unnecessary and can be removed. Removing unnecessary DoEvents() will increase application performance. The only DoEvents() calls your VB4 application requires are those used within your application, for example, to stop processing to offer the user the ability to cancel.

— MicroHelp UnInstaller 95 team

VB4

SPEEDY LIST BOXES

One way to speed list-box loading is to eliminate the constant redrawing required while loading. You can do this by calling the LockWindowUpdate API. LockWindowUpdate accepts an HWND as a parameter to start the lock and a zero parameter to unlock it. Only one window can be locked at a time. If LockWIndowUpdate is called with another HWND, the currently locked window will be unlocked.

To run this sample, put a command button and a list box on a form and paste in this code. Your code may not require the DoEvents within the loop that adds items, and a tight basic loop will keep the list from updating. However, there are times when the DoEvents is required.

```
Private Declare Function LockWindowUpdate Lib "user32"
(ByVal hwndLock As Long) As Long

Private Sub Command1_Click()
    Dim i%
    LockWindowUpdate (List1.hWnd)
    For i% = 1 To 1000
        List1.AddItem Str$(i%)
        DoEvents
    Next
    LockWindowUpdate (0&)
End Sub
```

—MicroHelp UnInstaller 95 team

WIN95

SHARP = DRESSED ICONS

To make icons look better, Windows 95 uses three standard-size icons; 16-by-16, 32-by-32, and 48-by-48. These three standard sizes should be included in any ICO file you create for inclusion in your program. Windows 95 resizes icons to fit the available space on caption bars, the desktop, and so forth. It most often chooses the closest size to its needs before resizing in order to minimize distortion. Good-looking icons will make your application look better to your user. You must use an editor that will support these resolutions in one ICO file. You can assign this Icon to the Icon property in a Form or MDIForm and WIN95 will use the icon of the appropriate size.

—MicroHelp UnInstaller 95 team

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

WIN95

DESIGN YOUR COLORS SO USERS CAN CHANGE THEM

Pay close attention to the color selections you make in your forms. Windows 95 gives the user more control over colors than previous versions of Windows allowed. problems arise if the user sets 3-D colors to something other than gray or sets a 3-D text color other than that used for window text (even Windows 95 does not handle this case properly in some places). When available, setting the Appearance property to 3D should handle this. However, you need to test for this by setting all your system colors to something unusual. Create and save a test scheme in your Control Panel/Display dialog so you can switch back and forth between your usual cool colors and your ugly test colors. A properly written Windows 95 program should not hard-code colors for its windows. It should allow the user to decide what is cool and what is ugly.

—MicroHelp UnInstaller 95 team

VB4 WIN95

CONTROL BOXES AND CLOSE BUTTONS

A form's control box property usually controls both the system menu (control box) and the close button (X in the top right corner) on the form. If the ControlBox Property is set to True, your form will have both a control box and a close button. You can make a dialog from a form without the control box, but with a close button by setting ControlBox to True, Icon to None and Borderstyle to 3 (Fixed Dialog).

—MicroHelp UnInstaller 95 team

VB4 WIN95

COMPARING LONG AND SHORT FILE NAMES

When you reference files in Windows 95, you may encounter situations when you are unsure whether you're working with a long or short file name. This becomes a problem if you are comparing file names or if you are searching for a specific file and do not know if it will be given in a long or short format. For example, `Dir$()` will always return a long file name, but the file you are comparing it to may be entered by a user as either a long or short name.

To compare two file names on a level field, you must change them both to short or to long. `Dir$()` returns a long file name, so you can take advantage of it as a built-in file name converter. What a plus! Pass it `C:\PRIVAT~1\MYSTUF~1\NEWTEX~1.TXT` and it will return `NEW TEXT DOCUMENT.TXT`. The only hitch is that you do not get the path, so you must pass `Dir$()` to each directory in the path, one by one. To do this, loop backwards through the full path string and let `Dir$()` convert each subdirectory. After you get the long file name from the path, call `Dir$("C:\PRIVAT~1\MYSTUF~1")` and get `My Stuff` back. With each successive call, take the long names that were returned and concatenate them into a full path string. This process will work even if you begin with a long file name and path.

—MicroHelp UnInstaller 95 team

VB4 WIN95

INSTANT ABOUT BOXES

The ShellAbout API call provides a quick and easy way to show an about box (using the standard Win95 format) without having to include an additional form in your project. The call uses four parameters: the hWnd of your main dialog, a string containing the name of your application, another string containing an optional additional line of text, and a long pointer to the handle of an icon. Start by placing this code in a BAS module:

```
Global Const GWW_HINSTANCE = (-6)

Declare Function ShellAbout Lib "shell32.dll" _
    Alias "ShellAboutA" (ByVal hwnd As Long, _
    ByVal szApp As String, ByVal szOtherStuff _
    As String, ByVal hIcon As Long) As Long
Declare Function ExtractIcon Lib "shell32.dll" _
    Alias "ExtractIconA" (ByVal hInst As Long, _
    ByVal lpszExeFileName As String, ByVal _
    nIconIndex As Long) As Long
Declare Function GetWindowLong Lib "user32" _
    Alias "GetWindowLongA" (ByVal hwnd As Long, _
    ByVal nIndex As Long) As Long
```

Add this routine to a button or menu to call your about box:

```
    Dim lRet As Long
    Dim lNull As Long
    Dim lIcon As Long
    Dim lInst As Long

    lInst = GetWindowLong_
        (Form1.hwnd, GWW_HINSTANCE)

    lIcon = ExtractIcon(lInst, "MYEXE.EXE", 0&)

    lRet = ShellAbout(Form1.hwnd, _
        "My App Name", "Copyright © 1995 _
        My Company Name" & Chr(13) & _
        Chr$(10) & "Serial # xxxxxxxxx-xxx", lIcon)
```

lRet will return true if the dialog was able to display and false if there was a problem. All of the required functions operate in both Windows 95 and Windows NT.

—MicroHelp UnInstaller 95 team

VB4

NOTIFY THE SYSTEM OF CHANGES

A new Windows 95-only API call notifies the system that you've changed something it should know about. The call, SHChangeNotify, is very handy in a number of cases. You might make this call in twenty different cases. This tip covers a few of the most important. The call itself has only four parameters:

• wEventId contains the flag identifying what has changed, such as:

SHCNE_ASSOCCHANGED	Changed a file type association.
SHCNE_ATTRIBUTES	Changed a file's attributes.
SHCNE_CREATE	Created a file.
SHCNE_DELETE	Deleted a file.
SHCNE_MKDIR	Created a new directory.
SHCNE_RENAMEFOLDER	Renamed a folder.
SHCNE_RENAMEITEM	Renamed an item in afolder.
SHCNE_RMDIR	Removed a directory.
SHCNE_UPDATEDIR	Updated the contents of a directory.
SHCNE_UPDATEITEM	Changed the properties of a printer or file.

These flags let Explorer know that something it is showing on screen might have changed and it needs to update its display.

• uFlags indicates what the next two parameters contain. Generally, you'll want to pass SHCNF_FLUSH so that the function doesn't return until it has processed the call. Instead, you may want to pass SHCNF_FLUSHNOWAIT so that the call returns immediately but the system continues to process the call in the background.

• dwItem1 is event specific, but for the flags you can pass null for both of these items.
• dwItem2 is event specific.

—MicroHelp UnInstaller 95 team

VB3 VB4

SPEED CRYSTAL REPORTS

If Crystal Reports' speed is lacking although your report contains no large graphics or large numbers of groups, change these two lines in your CRW.INI file to solve disk swapping problems:

```
MaxRecordMemory=0
MetapageSpillLimit=100
```

—Crystal, A Seagate Software Company Tech Support

VB3 VB4

AVOID ERRORS WHILE SIMULATING A DATABASE IN CRYSTAL REPORTS

When you're creating a database that simulates a client's database for reporting purposes, errors such as "MSAccess Error: Buf Too Short" or "Error Detected by Database DLL" may occur if fields expected by the database (but not in the report) are found.

Check the Verify on Every Print command under the Database menu before you ship the report. This feature causes the Print engine to check for differences in the database and to compensate for them.

—Crystal, A Seagate Software Company Tech Support

VB4

SET COLUMNS IN DBGRID AT DESIGN TIME

Set the number of columns in VB4's DBGrid at design time by right-clicking on the grid and selecting Edit from the menu. (The grid can now be edited interactively at design time.) Add columns by right-clicking on the grid again and selecting Add or Insert from the menu. When the grid is editable at design time, the columns widths can also be set interactively. Click on the form or another control to leave design-time editing.

—Apex Software Tech Support

VB4

DBGRID AUTOMATIC CONFIGURATION

If the layout of VB4's DBGrid is not changed at design time, it will automatically configure to a new record set when the Data control is refreshed with a new RecordSource.

—Apex Software Tech Support

VB4

REFERENCE DBGRID'S COLUMNS USING OBJECT VARIABLES

In VB4, DBGrid's columns can be referenced using object variables. This reduces the amount of typing required and makes code more readable. The code is also more efficient because the full name doesn't need resolved with each reference. For example:

```
Dim Col() As Column
Dim NumCols As Integer

Private Sub Form_Load()
    Data1.RecordSource = "SELECT * FROM Publishers;"
    Data1.Refresh

    NumCols = DBGrid1.Columns.Count
    ReDim Col(NumCols)
    Dim i As Integer

    For i = 0 To NumCols - 1
        Set Col(i) = DBGrid1.Columns(i)
        Col(i).AllowSizing = False 'disable user sizing
        Col(i).DividerStyle = 0 'right divider = none
    Next i
End Sub
```

—Apex Software Tech Support